SPARK+AI
SUMMIT 2018

Accelerating Real Time Analytics with Spark Streaming and FPGAaaS

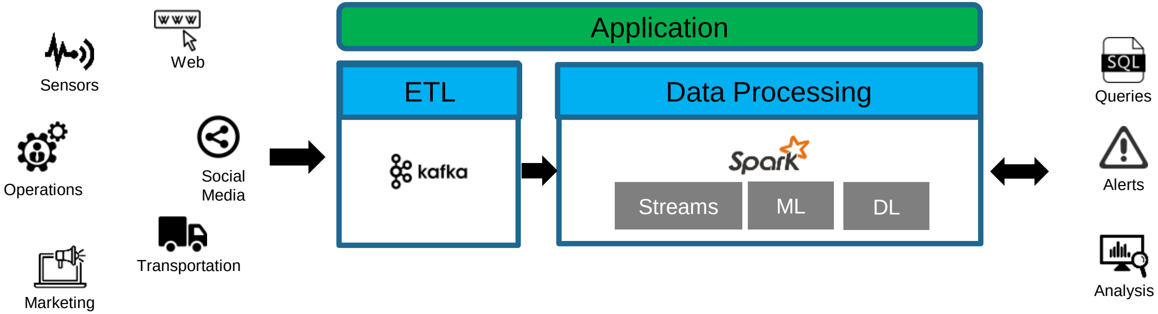
P.K.Gupta, Megh Computing

#HWCSAIS17

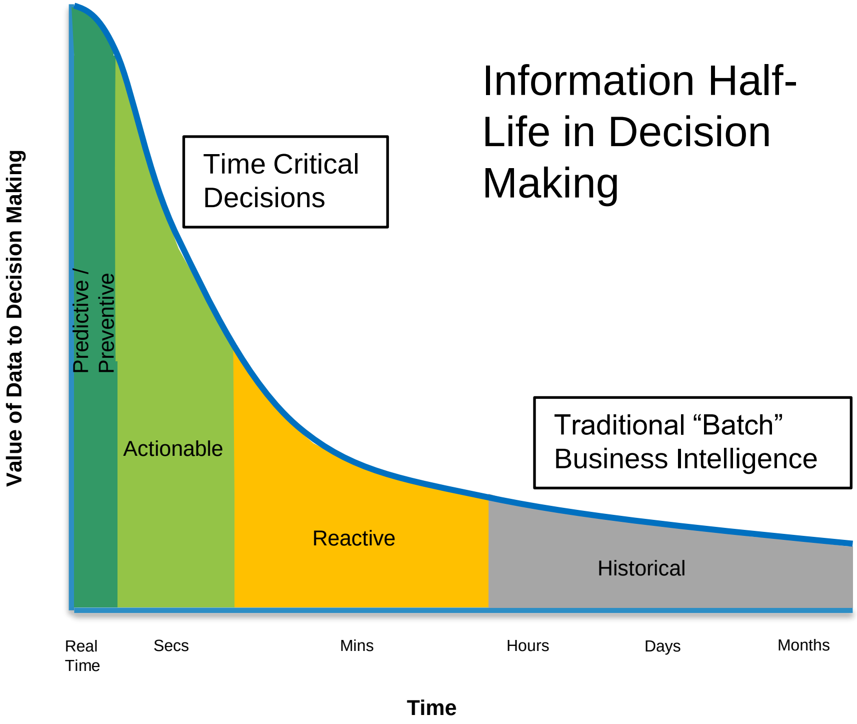
Agenda

- Using Spark Streaming for Real Time Analytics
- Why FPGA : Low Latency and High Throughput
  - Inline Processing
  - Offload Processing
- Challenges in Using FPGA accelerators
- Megh Platform
  - Arka Runtime
  - Sira AFUs
- Demo Applications
- Conclusion

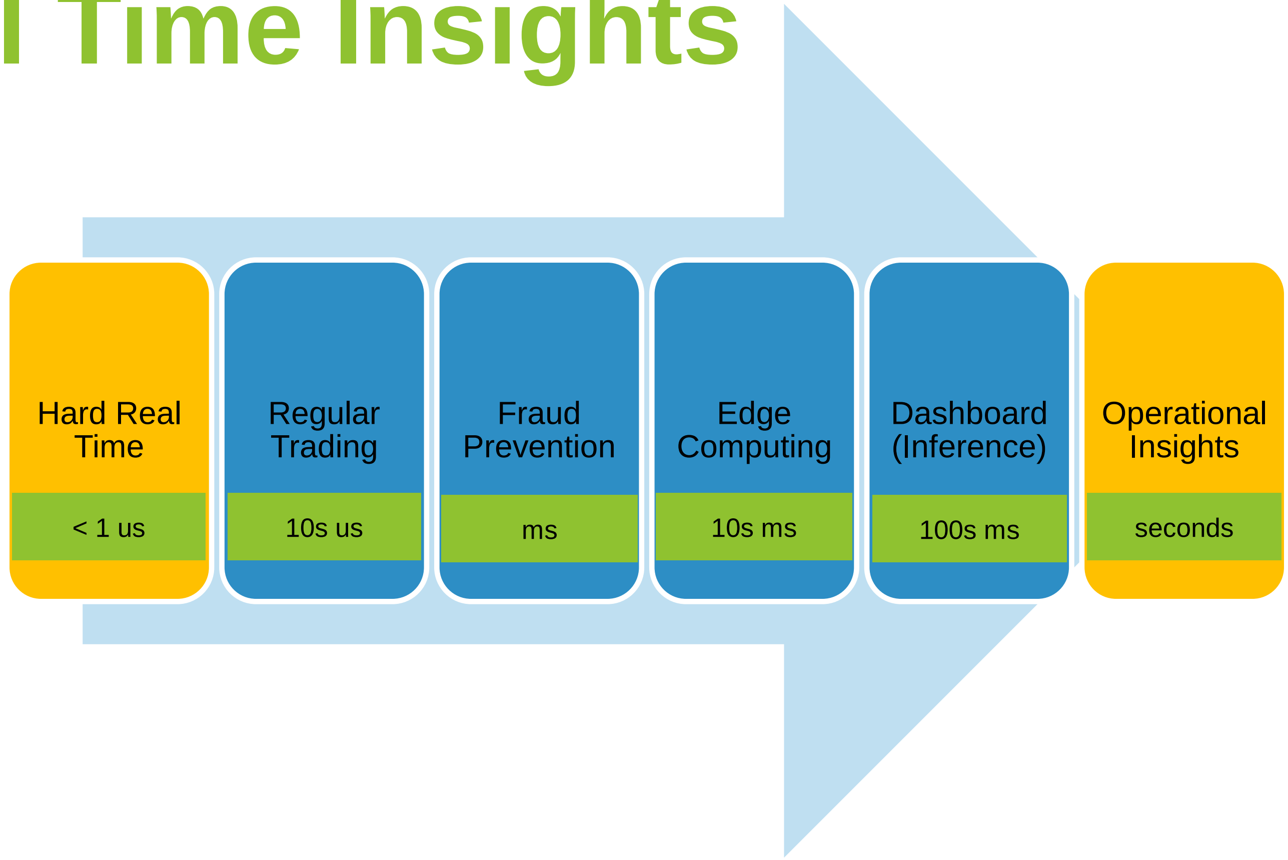
Using Spark Streaming with ML / DL for Real Time Analytics



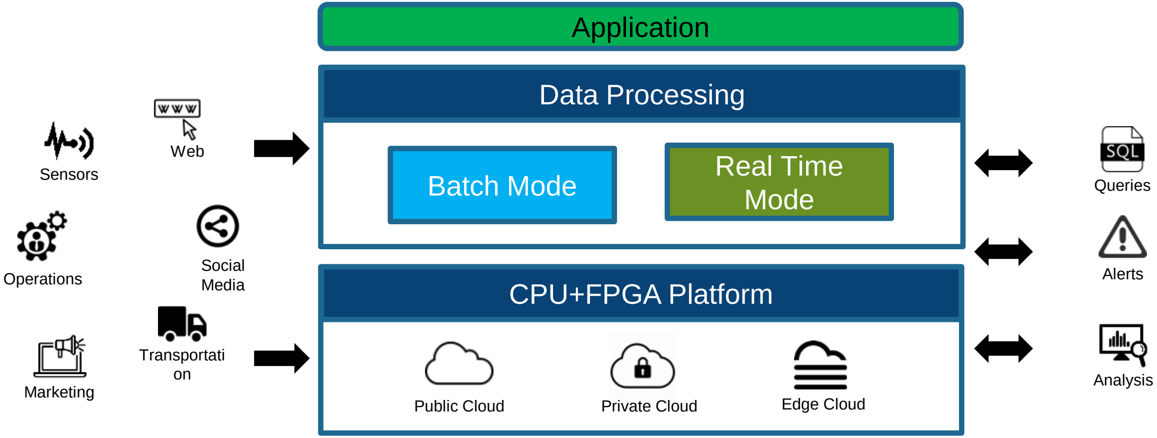
Real Time vs. Batch Insights



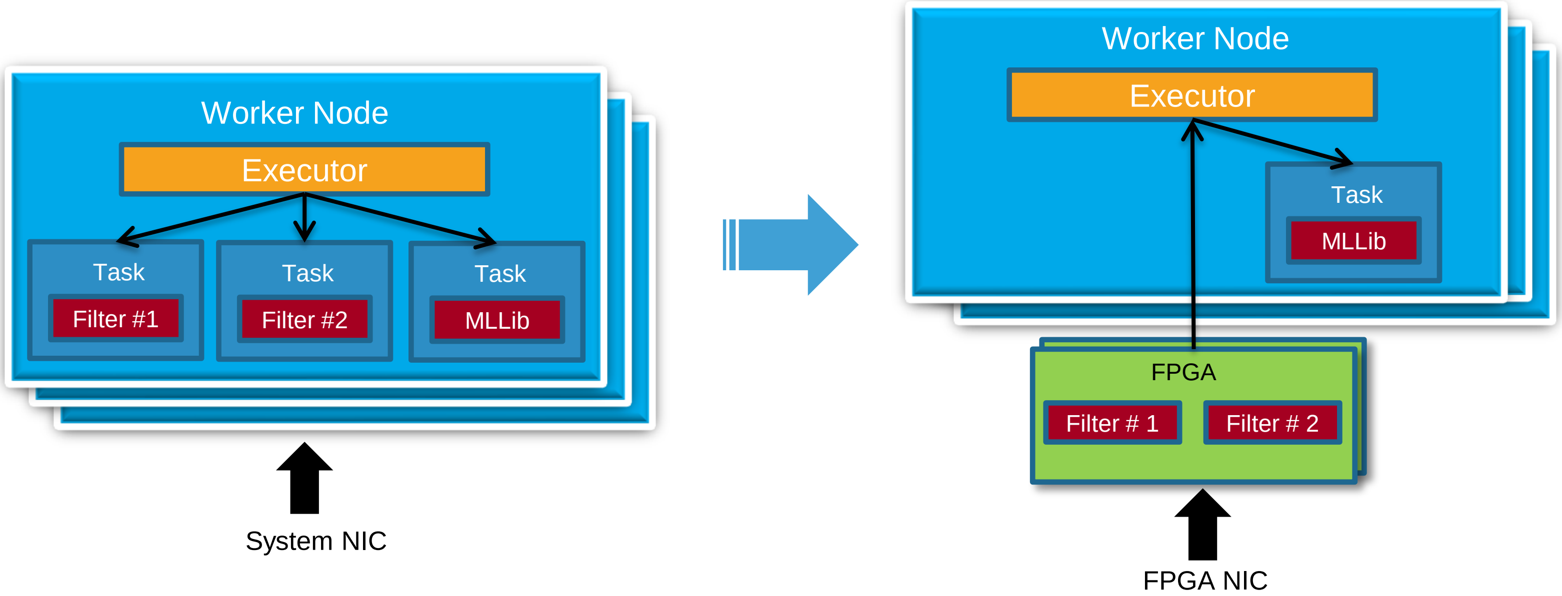
Real Time Insights



Real Time Analytics platform:
using Heterogeneous CPU+FPGA computing

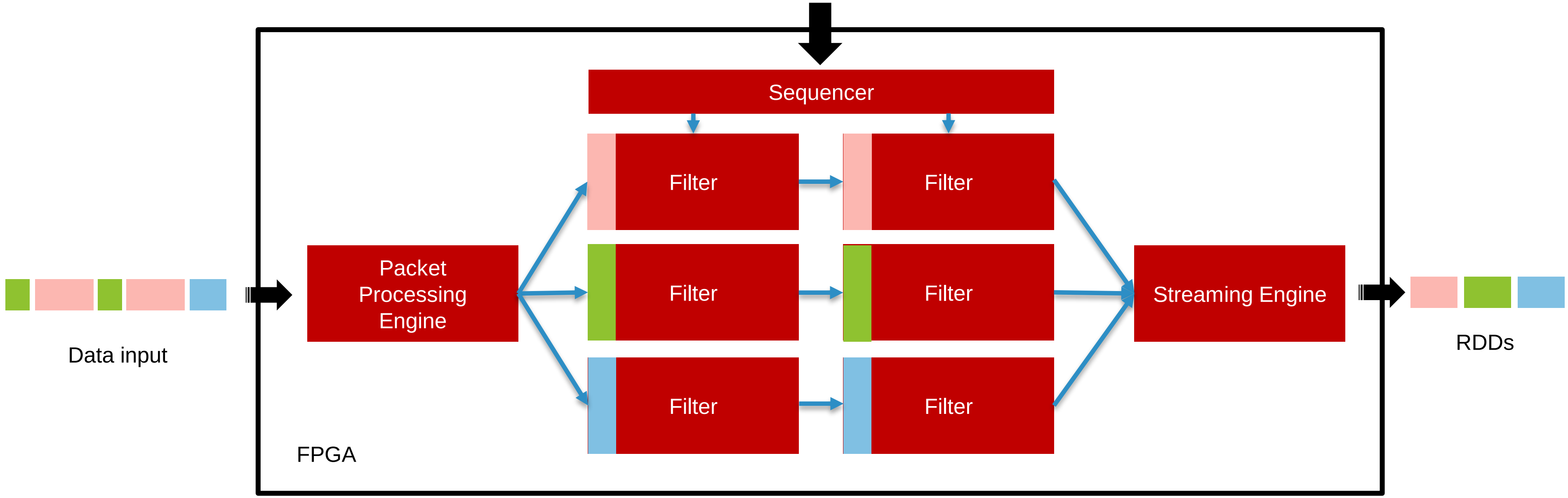


In-Line Stream Processing:
using heterogeneous CPU+FPGA platform

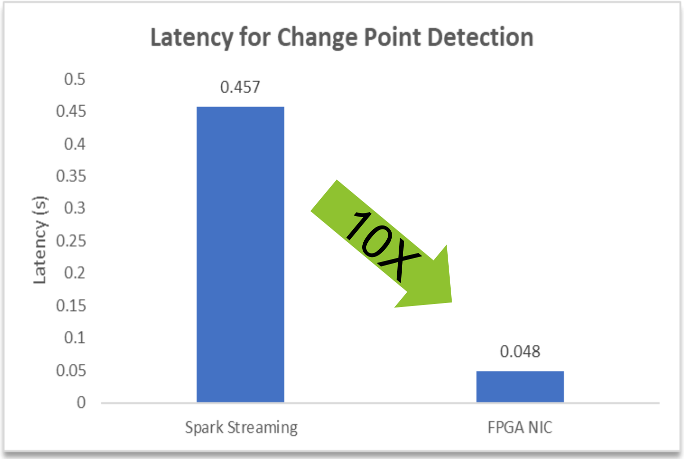


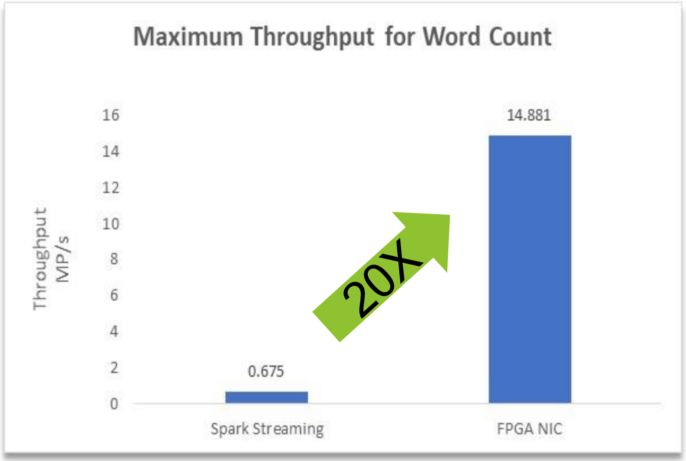
FPGA terminates Network and dynamically chains filters to provide pre-processed / low latency DStreams to SPARK apps transparently

In-Line Stream Processing:
FPGA Architecture



In-Line Stream Performance



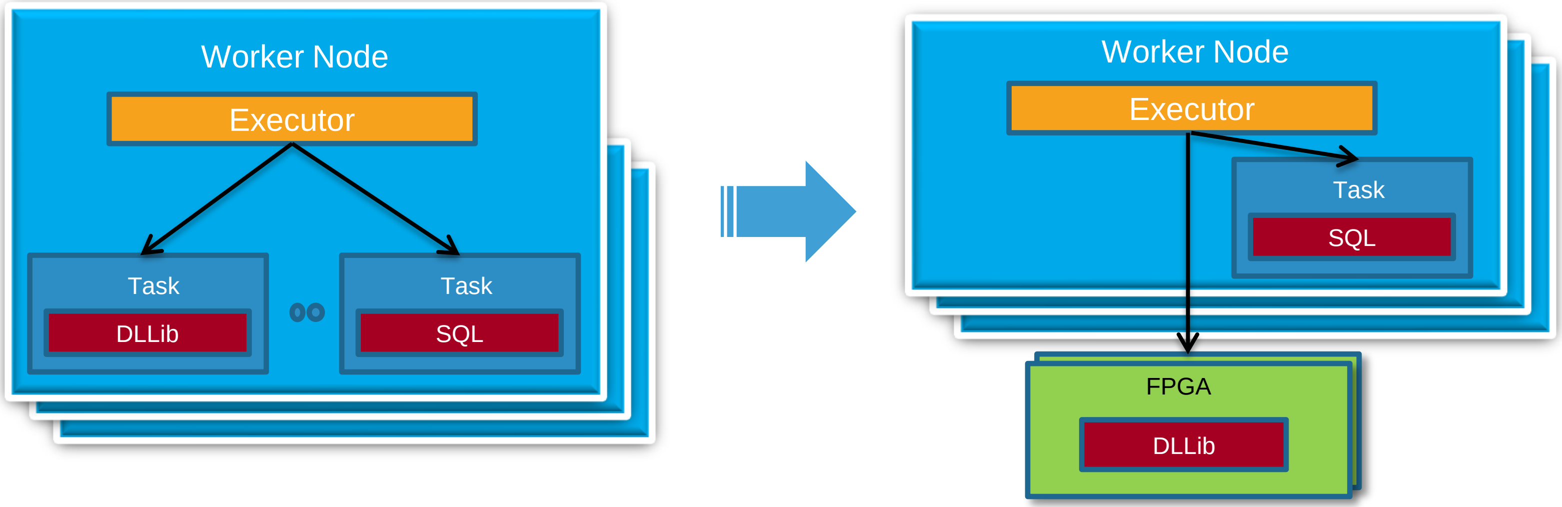


Lower Latency

Higher Throughput

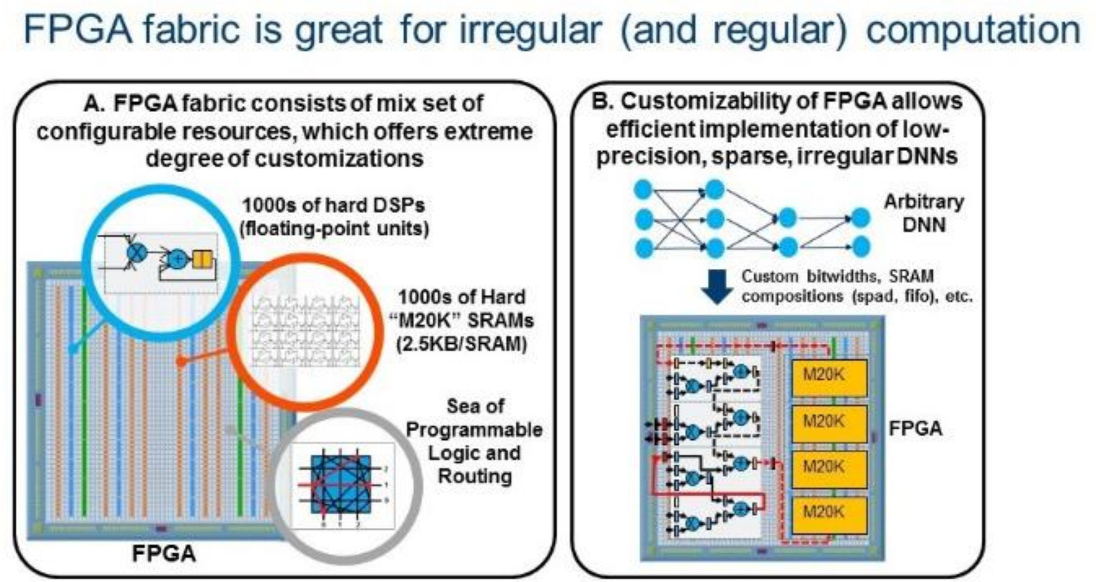
Source: An FPGA Based Low Latency Network Processing for Spark Streaming, K. Nakamura et.al. Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016

Off-load Processing (ML/DL):
using heterogeneous CPU+FPGA platform



Accelerate ML/DL algorithm transparently by providing
SPARK bindings to FPGA implementations of ML/DL libraries

Off-load ML/DL Processing:
FPGA Architecture



Source: CAN FPGAS BEAT GPUS IN ACCELERATING NEXT-GENERATION DEEP LEARNING?
The Next Platform, March 21, 2017

Off-load ML/DL Performance

10X

Lower Latency

500 fps

Higher Throughput

Source: Accelerating Persistent Neural Networks at Datacenter Scale
Eric Chung, et. Al, HotChips, 2017

Challenges in using FPGA

1

Programming FPGAs

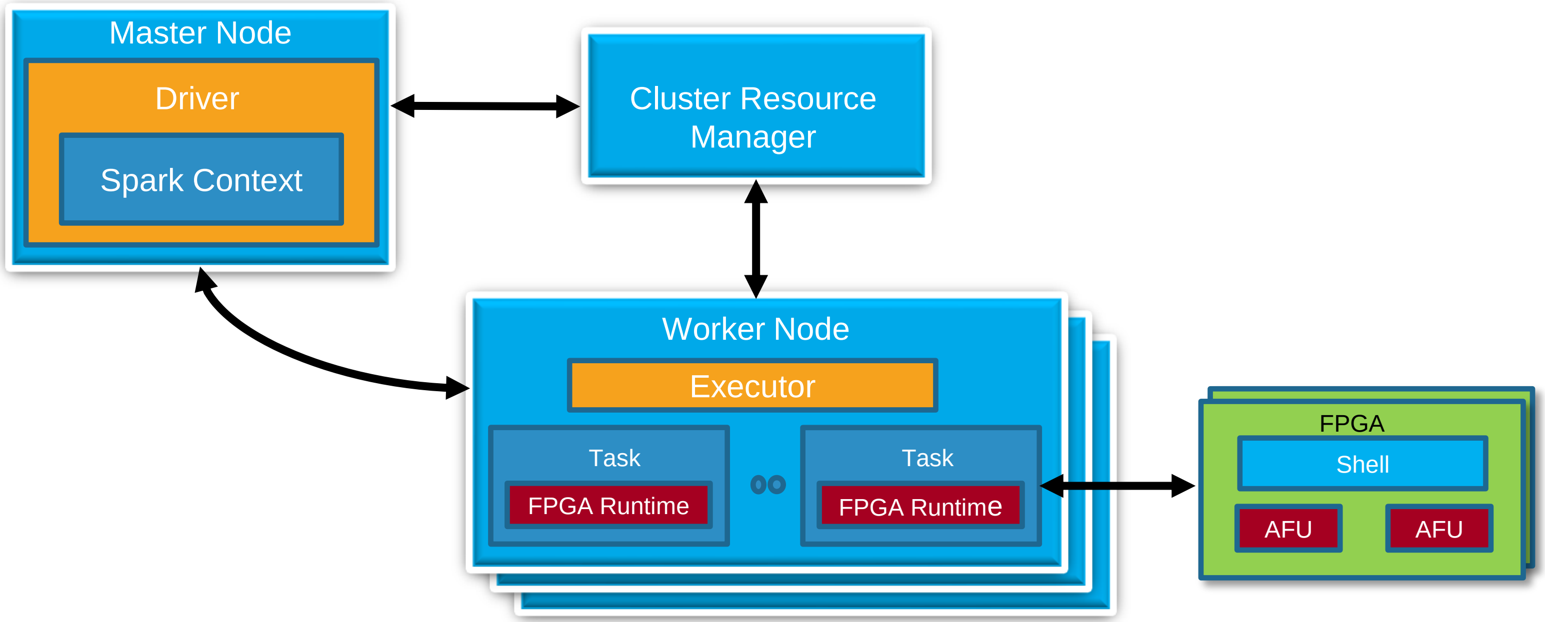
2

Managing FPGAs in the DataCenter

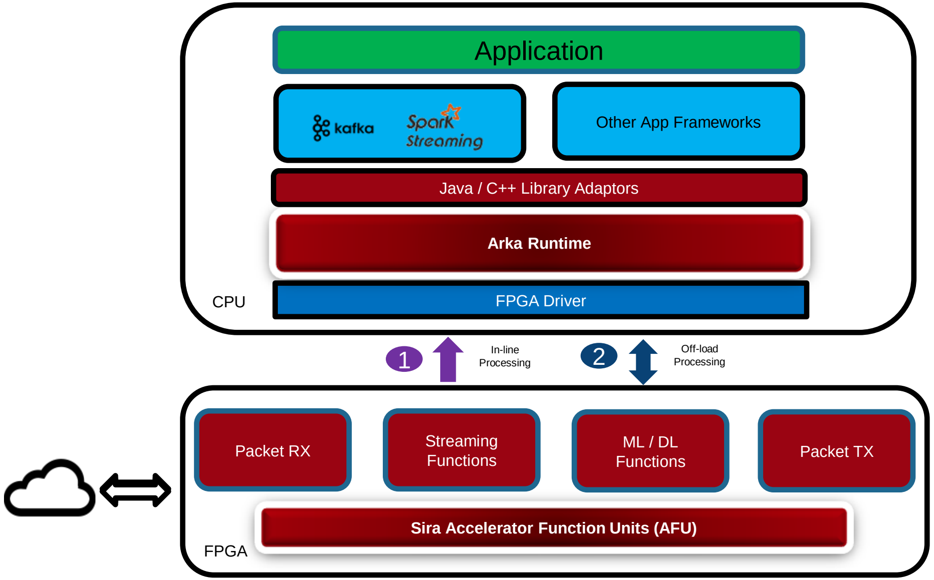
3

Integrating FPGAs into applications

Spark Streaming Architecture using CPU+FPGA platform



Megh Platform:
abstracts the complexity of the FPGA



Application:
- uses standard APIs
- And/or custom APIs

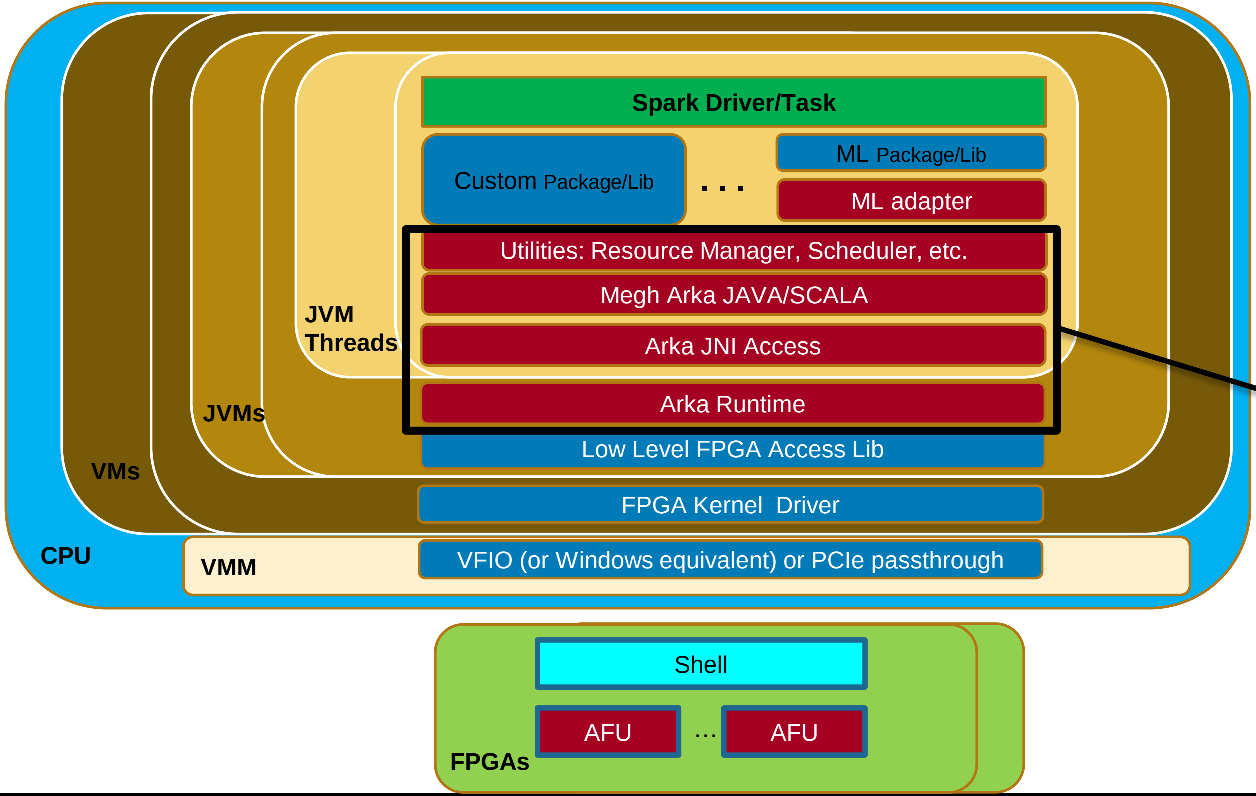
Arka Runtime:
- FPGA management
- SW fallback
- Expose AFaaS

Sira Accelerators:
- Downloaded at Runtime
- Bare Metal or Exposed to VMs via VMM

Infrastructure Components

Megh Components

Virtualized Real Time Analytics Stack

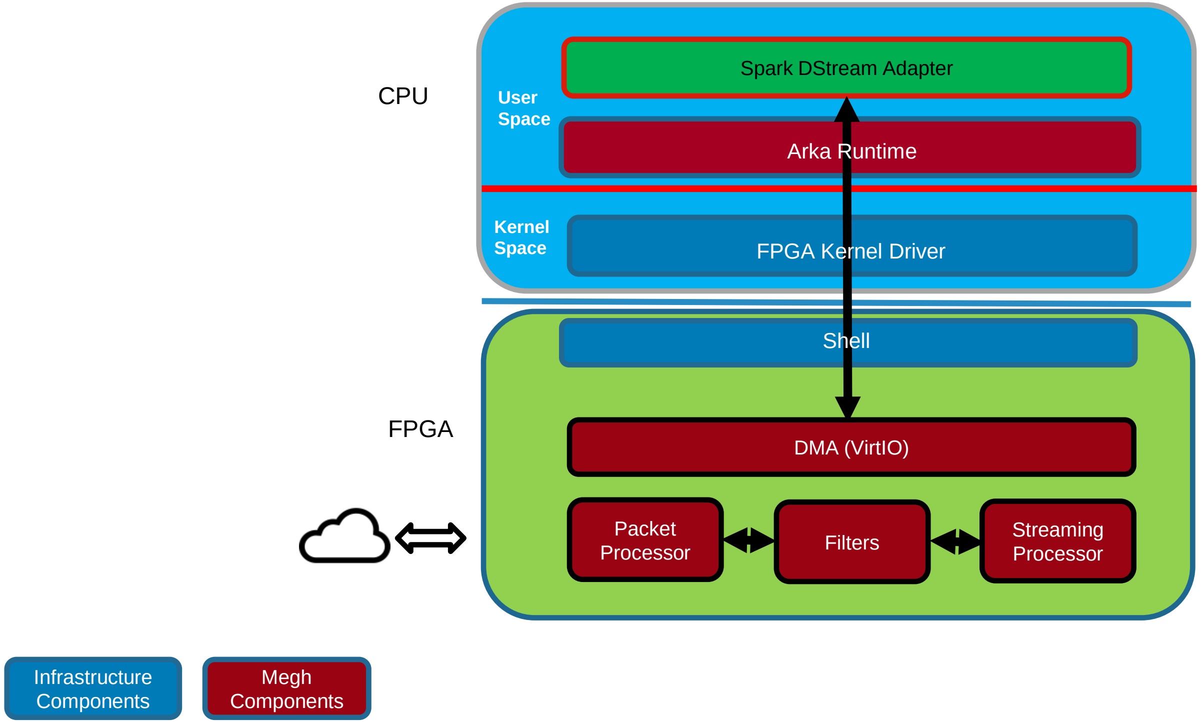


Application:
- uses standard APIs
- And/or custom APIs

Runtime:
- FPGA management
- SW fallback
- Expose AFaaS

Accelerators:
- Downloaded at Runtime
- Exposed to VMs via VMM

In-Line Processing:
Smart rx/tx adaptor architecture



- **Packet Processor:** Intercepts network packets destined to Spark
- **Filters:** Performs data cleaning, re-size, layout transforms (ETL operations)
- **Streaming Processor**: Creates D-Stream packets for Spark

Inline sample implementation

```
public final class JavaSqlNetworkWordCount {
  private static final Pattern SPACE = Pattern.compile(" ");
  public static void main(String[] args) throws Exception {
    if (args.length < 2) {
      System.err.println("Usage: JavaNetworkWordCount <hostname> <port>");
      System.exit(1);
    }
    StreamingExamples.setStreamingLogLevels();

    // Create the context with a 1 second batch size
    SparkConf sparkConf = new SparkConf().setAppName("JavaSqlNetworkWordCount");
    JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.seconds(1)); 1

    // Create a JavaReceiverInputDStream on target ip:port and count the
    // words in input stream of \n delimited text (eg. generated by 'nc')
    JavaReceiverInputDStream<String> lines = ssc.socketTextStream( 2
        args[0], Integer.parseInt(args[1]), StorageLevels.MEMORY_AND_DISK_SER);
    JavaDStream<String> words = lines.flatMap(Split2Words()); 3
    ..
}
    ..
}
```

CPU IMPLEMENTATION

1. Sets up the DStream CPU adapter connected to System NIC.
2. Configure IP/port on CPU NIC
3. etlLibCPU.jar (CPU implementation)
   - split2Words()
   - spilt2Sort()
   - split2Count()

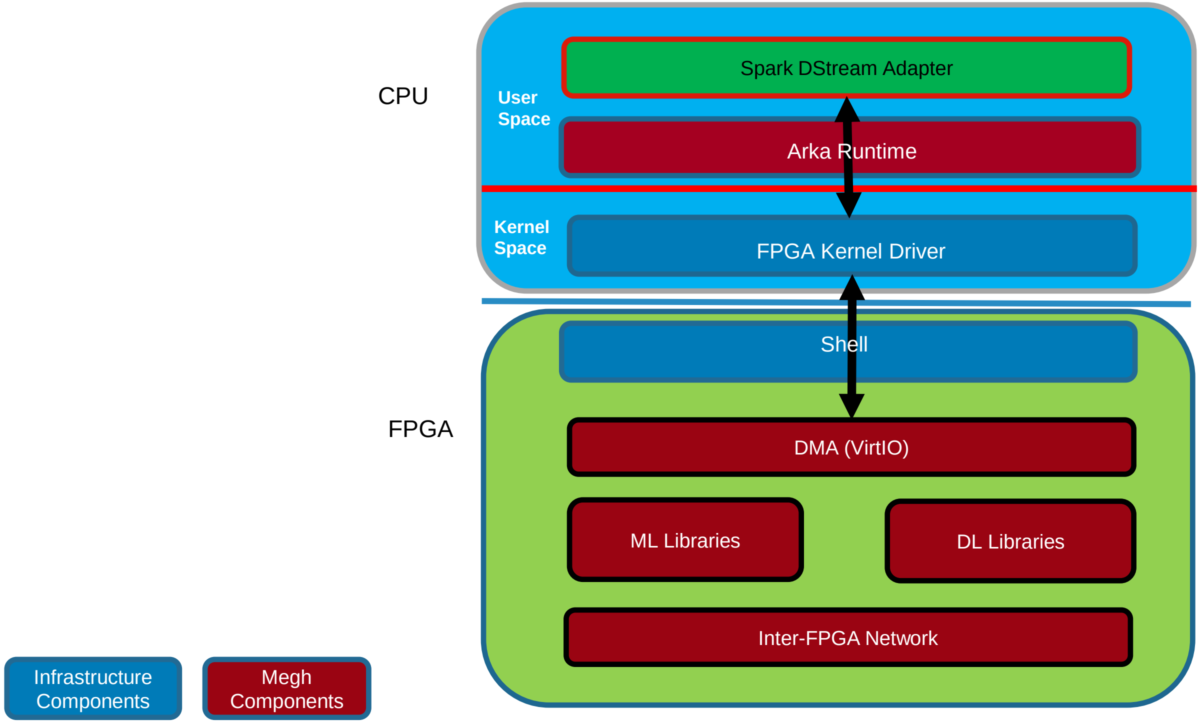
FPGA IMPLEMENATAION

1. Sets up the DStream FPGA adapter connected to FPGA NIC.
2. Configures IP/Port on FPGA NIC
3. etlLibCPU.jar(FPGA implementation)
   - split2Words()
   - spilt2Sort()
   - split2Count()

* Full implementation
https://github.com/apache/spark/blob/master/examples/src/main/java/org/apache/spark/examples/streaming/JavaSqlNetworkWordCount.java

FPGA is setup to stream and filter data - before passing it to SPARK as DStream object.

Off-load Processing:
Low latency off-Load of ML/DL libraries



- **Machine Learning Libraries:** Optimized libraries for K-Means, SVM, etc.
- **Deep Learning Libraries:** Optimized libraries for DNN based inference engines.
- **Inter-FPGA Network**: FPGA network for sharing FPGA resources for larger DNN topologies

Offload Sample Implementation

```
public class JavaKMeansExample {
  public static void main(String[] args) {
    SparkConf conf = new SparkConf().setAppName("JavaKMeansExample");
    JavaSparkContext jsc = new JavaSparkContext(conf);
    ..
    // Cluster the data into two classes using KMeans
    int numClusters = 2;
    int numIterations = 20;
    KMeansModel clusters = KMeans.train(parsedData.rdd(), numClusters,numIterations);
    ..
    double cost = clusters.computeCost(parsedData.rdd());
    System.out.println("Cost: " + cost);

    // Evaluate clustering by computing Within Set Sum of Squared Errors
    double WSSSE = clusters.computeCost(parsedData.rdd());
    System.out.println("Within Set Sum of Squared Errors = " + WSSSE);
    ..
    jsc.stop();
  }
}
```

mlib.jar
(CPU library implementation)
- KmeansModel.train()
- KmeansModel.computeCost()

mlibFPGA.jar
(FPGA accelerated library implementation)
- KmeansModel.train()
- KmeansModel.computeCost()

* Full implementation:
https://github.com/apache/spark/blob/master/examples/src/main/java/org/apache/spark/examples/mllib/JavaKMeansExample.java

CPU and FPGA share the same function signature -
providing application transparent acceleration by using FPGA library

numAdd Demo:
Implementation details

```
public static void main( String[] args ) throws Exception {
    System.out.println(" Java: NumAdd Spark Demo.\n");
    Long total = null;

    SparkConf sparkConf = new SparkConf().setAppName( "NumAdd" );
    JavaSparkContext ctx = new JavaSparkContext( sparkConf );
    JavaRDD<String> lines = ctx.textFile( args[0], 1 );

    JavaRDD<Long> sums = lines.map( new sumOneString() );
    total = sums.reduce( (a,b) -> (a+b) );

    System.out.println( "Total is -> " + total );
    ctx.stop();
}
```

numAdd is slight variation of the popular WordCount Sample where numbers in the files are parsed and added up using SPARK

Accelerated Operation:
sumOneString

```
AFU.Factory fpgaFactory = new AFU.Factory();
AFU wc  = fpgaFactory.createAFU("meghna");


TransferBuffer inbuf = wc.getTransferBuffer( input1.length() );
wc.queueInputBuffer( inbuf );

// Reuse buffer 1 for the output. AFU design ensures this is safe.
wc.queueOutputBuffer( inbuf );   //  Arka permits it.


wc.startFunction();           // The real work starts here


TransferBuffer obuff = wc.waitOnOutputQueue();
return ( obuff.getByteBuffer().asLongBuffer().get(0) );
```

Instantiate AFU as a Service. Enables multiple distinct implementations to co-exist and be selected dynamically: specifically, an FPGA implementation and a CPU-based fallback implementation.

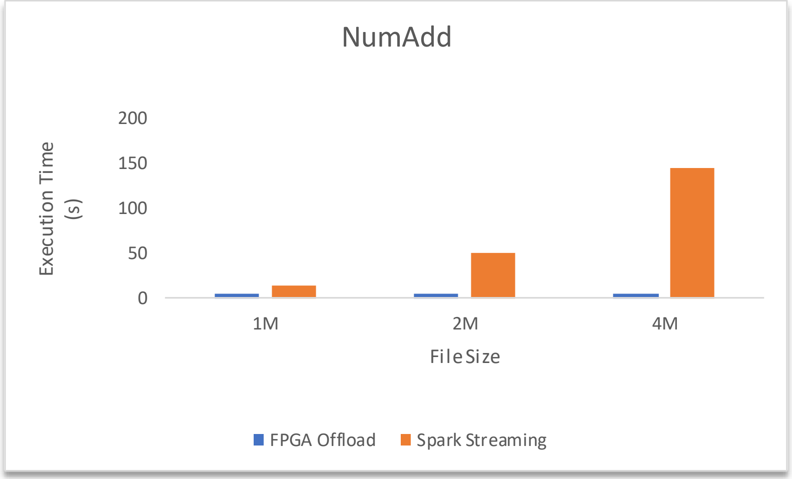
Buffer Queue based model
- (Register interface available but not shown)

AFU optimized Transfer Buffers allow for:
- Zero copy to HW. And efficient access.
- Efficient access from Java/Scala
- AFU specific implementation.
- May use direct byte buffers, SVM, Netty, Apache Arrow etc…

Start operation.

Wait for results in output queue.

Demo: NumAdd Offload Profiling



* Executor/task on the worker node restricted to 1 thread

In Summary….

- Megh CPU+FPGA platform optimized for Real Time Analytics
- Arka Runtime supports different streaming frameworks
- Sira AFUs deliver low latency and high throughput for inline and offload processing

Thank You

info@meghcomputing.com