Transparent GPU Exploitation on Apache Spark

Kazuaki Ishizaki, IBM Research
Madhusudanan Kandasamy, IBM Systems

#Res9SAIS

About Me – Madhusudanan Kandasamy

- STSM(Principal Engineer) at IBM Systems
- Working for IBM Power Systems over 15 years
  - AIX & Linux OS Development
  - Apache Spark Optimization for Power Systems
  - Distributed ML/DL Framework with GPU & NVLink
- IBM Master Inventor (20+ Patents, 18 disclosure publications)
- Committer of GPUEnabler
  - Apache Spark Plug-in to execute GPU code on Spark
    - https://github.com/IBMSparkGPU/GPUEnabler
- Github: https://github.com/kmadhugit
- E-mail: madhusudanan@in.ibm.com

What You Will Learn from This Talk

- Why accelerate your workloads using GPU on Spark
  - GPU/CUDA Overview
  - Spark + GPU for ML workloads
- How to program GPUs in Spark
  - Invoke Hand-tuned GPU program in CUDA
  - Translate DataFrame program to GPU code automatically
- What are key factors to accelerate program
  - Parallelism in a program
  - Data format on memory

GPU/CUDA Overview

- GPGPU - Throughput
- CUDA, which is famous, requires programmers to explicitly write operations for
  - allocate/deallocate device memories
  - copying data between CPU and GPU
  - execute GPU kernel

```
void fooCUDA(N, float *A, float *B, int N) {
  int sizeN = N * sizeof(float);
  cudaMalloc(&d_A, sizeN); cudaMalloc(&d_B, sizeN);
  cudaMemcpy(d_A, A, sizeN, HostToDevice);
  GPUMultiplyBy2<<<N, 1>>>(d_A, d_B, N);
  cudaMemcpy(B, d_B, sizeN, DeviceToHost);
  cudaFree(d_B); cudaFree(d_A);
}
```

```
// code for GPU
__global__ void GPUMultiplyBy2(
    float* d_a, float* d_b, int n) {
  int i = threadIdx.x;
  if (n <= i) return;
  d_b[i] = d_a[i] * 2.0;
}
```

Spark + GPU for ML workloads

- Spark provides efficient ways to parallelize jobs across cluster of nodes
- GPUs provide thousands of cores for efficient way to parallelize job in a node.
- GPUs provide up to 100x processing over CPU *
- Combining Spark + GPU for lightning fast processing
  - We will talk about two approaches

* https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/

Outline

- Why accelerate your workloads using GPU on Spark
- How to program GPUs in Spark
  - Invoke Hand-tuned GPU program in CUDA
  - Translate DataFrame program to GPU code automatically
- Toward faster GPU code
- How two frameworks work?

Invoke Hand-tuned GPU program in CUDA

- GPUEnabler to simplify development
- Implemented as Spark package
  - Can be drop-in into your version of Spark
- Easily Launch hand coded GPU kernels from `map()` or `reduce()` parallel function in **RDD, Dataset**
- manages GPU memory, copy data between GPU and CPU, and convert data format
- Available at https://github.com/IBMSparkGPU/GPUEnabler

Example - hand tuned CUDA kernel in Spark

Step 1: Write CUDA kernels (without memory management and data copy)

```
__global__ void multiplyBy2(int *in, int *out, long size) {
  long i = threadIdx.x + blockIdx.x * blockDim.x;
  if (size <= i) return;
  out[i] = in[i] * 2;
}
```

CUDA is a programming language for GPU defined by NVIDIA

PTX is an assembly language file that can be generated by a CUDA file

Step 2: Write Spark program

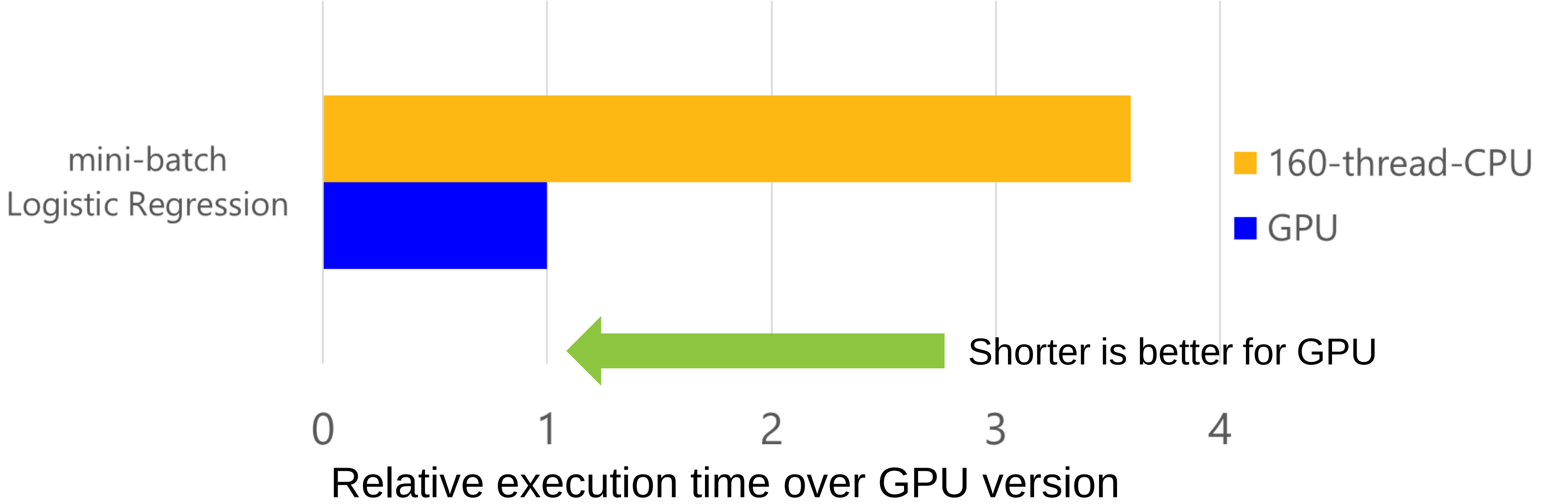
```
Object SparkExample {
  val mapFunction = new CUDAFunction("multiplyBy2", Seq("value"), "example.ptx")
  val output = sc.parallelize(1 to 65536, 24).cache
    .mapExtFunc(x => x*2, mapFunction).show }
```

Step 3: Compile and submit

```
$ nvcc example.cu -ptx
$ mvn package
$ bin/spark-submit --class SparkExample SparkExample.jar
  --packages com.ibm:gpu-enabler_2.11:1.0.0
```

Performance Improvements of GPU program over Parallel CPU

- Achieve 3.6x for CUDA-based mini-batch logistic regression using one P100 card over POWER8 160 SMT cores



IBM Power System S822LC for High Performance Computing "Minsky", at 4 GHz with 512GB memory, one P100 card, Fedora 7.3, CUDA 8.0, IBM Java pxl6480sr4fp2-20170322_01(SR4 FP2), 128GB heap, Apache Spark 2.0.1, master="local[160]", GPU Enabler as 2017/5/1, N=112000, features=8500, iterations=15, mini-batch size=10, parallelism(GPU)=8, parallelism(CPU)=320

Outline

- Why accelerate your workloads using GPU on Spark
- How to program GPUs in Spark
  - Invoke Hand-tuned GPU program in CUDA (GPUEnabler)
  - Translate DataFrame program to GPU code automatically
- Toward faster GPU code
- How two frameworks work?

“Transparent” GPU Exploitation

- Enhanced Spark by modifying Spark source code
- Accept expression in `select()`, `selectExpr()`, and `reduce()` in **DataFrame**
- Automatically generate CUDA code from DataFrame program
- Automatically manage GPU memory and copy data between GPU and CPU
- No data format conversion is required

Example - “Transparent” GPU Exploitation

- Write Spark program in DataFrame

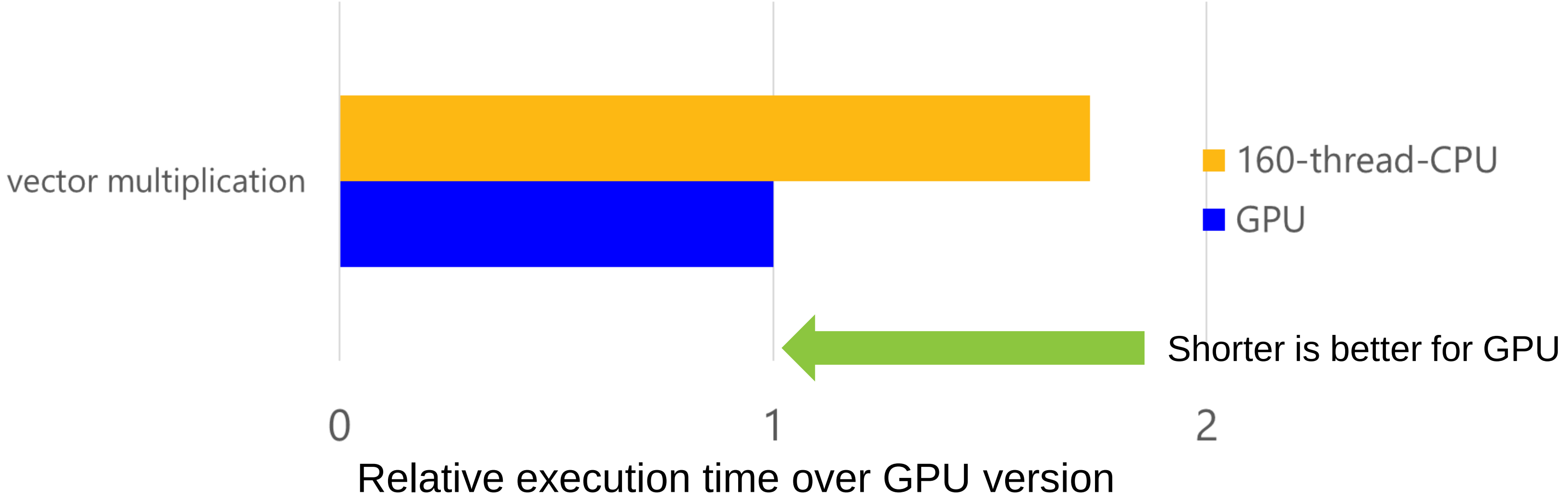
```
Object SparkExample {
  val output = sc.parallelize(1 to 65536, 24).toDF(“value”).cache
    .select($”value” * 2).cache.show }
```

- Compile and submit them

```
$ mvn package
$ bin/spark-submit --class SparkExample SparkExample.jar
```

Performance Improvements of Spark DataFrame program over Parallel CPU

- Achieve 1.7x for Spark vector multiplication using one P100 card over POWER8 160 SMT cores



IBM Power System S822LC for High Performance Computing "Minsky", at 4 GHz with 512GB memory, one P100 card, Fedora 7.3, CUDA 8.0, IBM Java pxl6480sr4fp2-20170322_01(SR4 FP2), 128GB heap, Based on Apache Spark master (id:657cb9), master="local[160]", N=480, vector length=1600, parallelism(GPU)=8, parallelism(CPU)=320

Outline

- Why accelerate your workloads using GPU on Spark
- How to program GPUs in Spark
  – Invoke Hand-tuned GPU program in CUDA (GPUEnabler)
  – Translate DataFrame program to GPU code automatically
- Toward faster GPU code
- How two frameworks work?

About Me – Kazuaki Ishizaki

- Researcher at IBM Research in compiler optimization
- Working for IBM Java virtual machine over 20 years
  - In particular, just-in-time compiler
- Active Contributor of Spark since 2016
  - 98 commits, #37 in the world (25 commits, #8 in 2018)
- Committer of GPUEnabler
- Homepage: http://ibm.biz/ishizaki
- Github: https://github.com/kiszk, Twitter: @kiszk

Toward Faster GPU code

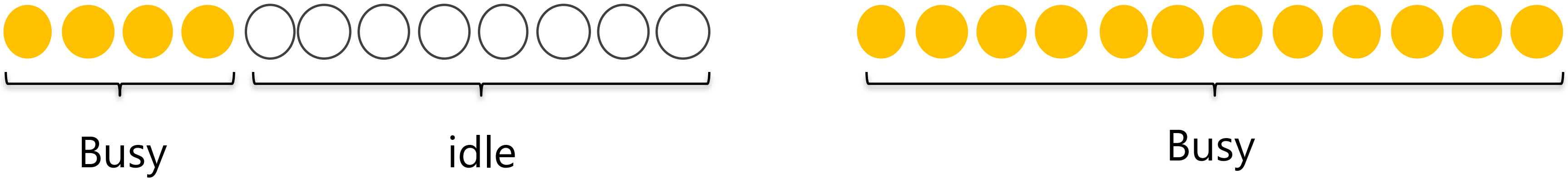
- Assign a lot of parallel computations into GPU cores
- Reduce # of memory transactions to GPU memory

Assign A Lot of Parallel Computations into GPU Cores

- Achieve high utilization of GPU

Achieve low performance

Achieve high performance

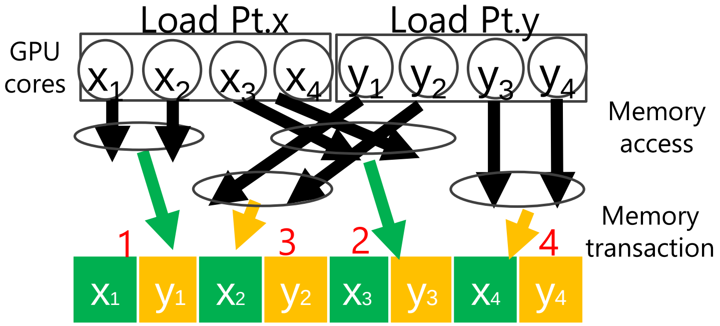


Reduce # of Memory Transactions

- Depends on memory layout, # of memory transactions are different in a program

```
Pt(x: Int, y: Int)
Load Pt.x1
Load Pt.x2
Load Pt.x3
Load Pt.x4
Load Pt.y1
Load Pt.y2
Load Pt.y3
Load Pt.y4
```

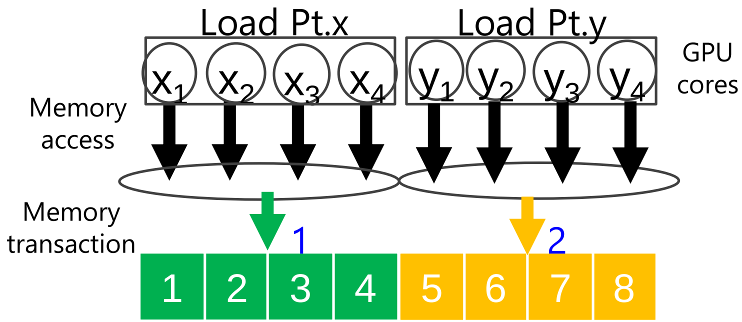
Memory access



Row-oriented layout

4 v.s. 2

memory transactions to GPU device memory



Column-oriented layout

Assumption: 4 consecutive data elements can be coalesced by GPU hardware

Toward Faster GPU Code

- Assign a lot of parallel computations into GPU cores
  - Spark program has been already written by a set of parallel operations
    - e.g. `map, join, …`
- Reduce # of memory transactions
  - Column-oriented layout achieves better performance
    - The paper* reports 3.3x performance improvement of GPU kernel execution of kmeans over row-oriented layout

* Che, Shuai and Sheaffer, Jeremy W. and Skadron, Kevin.
"Dymaxion: Optimizing Memory Access Patterns for Heterogeneous Systems", SC'11

Questions

- How can we write a parallel program for GPU on Spark?

- How can we use column-oriented storage for GPU in Spark?

Questions

- How can we write a parallel program for GPU on Spark?
  - Thanks to Spark programming model!!

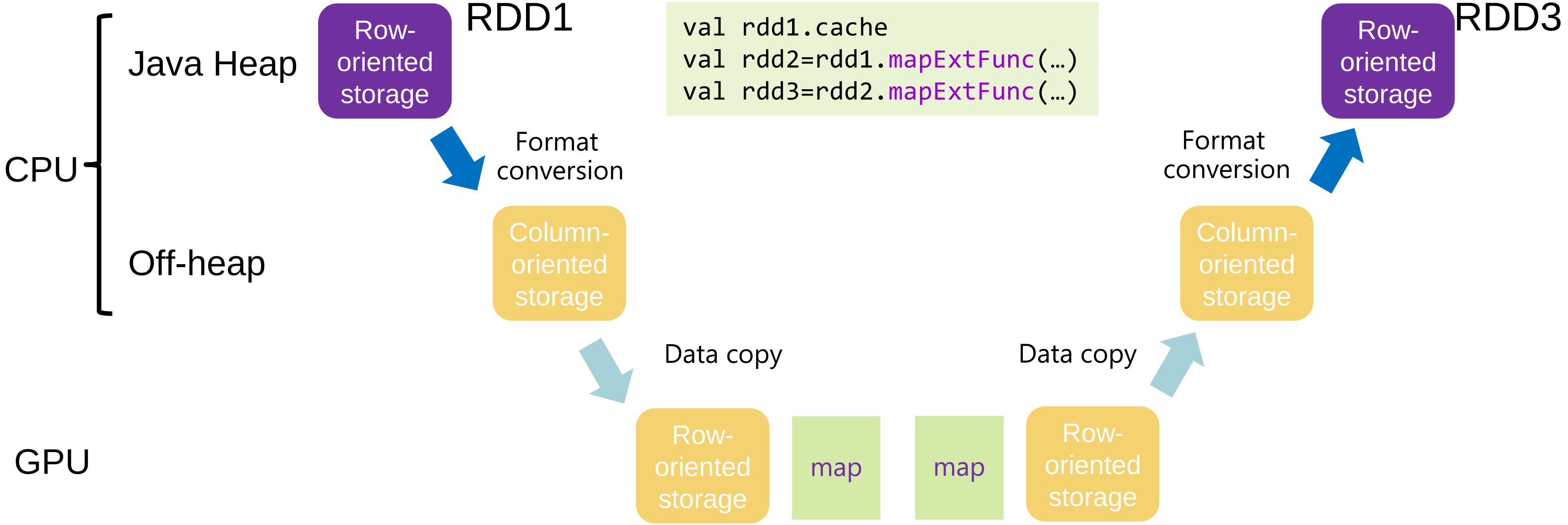
- How can we use column-oriented storage for GPU in Spark?

Outline

- Why accelerate your workloads using GPU on Spark
- How to program GPUs in Apache Spark
  - Invoke hand-tuned GPU program in CUDA (GPUEnabler)
  - Translate DataFrame program to GPU code automatically
- Toward faster GPU code
- How two frameworks work?

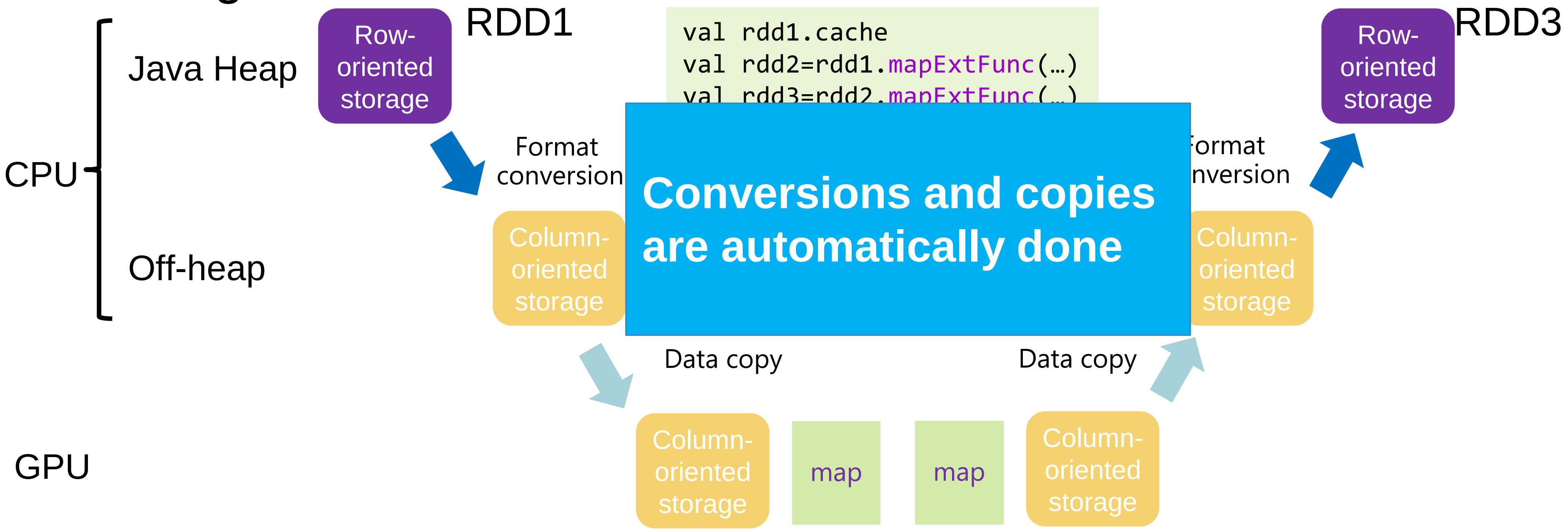
Data Movement with GPUEnabler

- Data in RDD is moved into off-heap as column-oriented storage



Data Movement with GPUEnabler

- Data in RDD is moved into off-heap as column-oriented storage



How To Write GPU Code in GPUEnabler

- Write a GPU kernel corresponds to map()

Spark Program

```
val rdd2 = rdd1
  .mapExtFunc(p => Point(p.x*2, p.y*2), mapFunction)
```

GPU Kernel

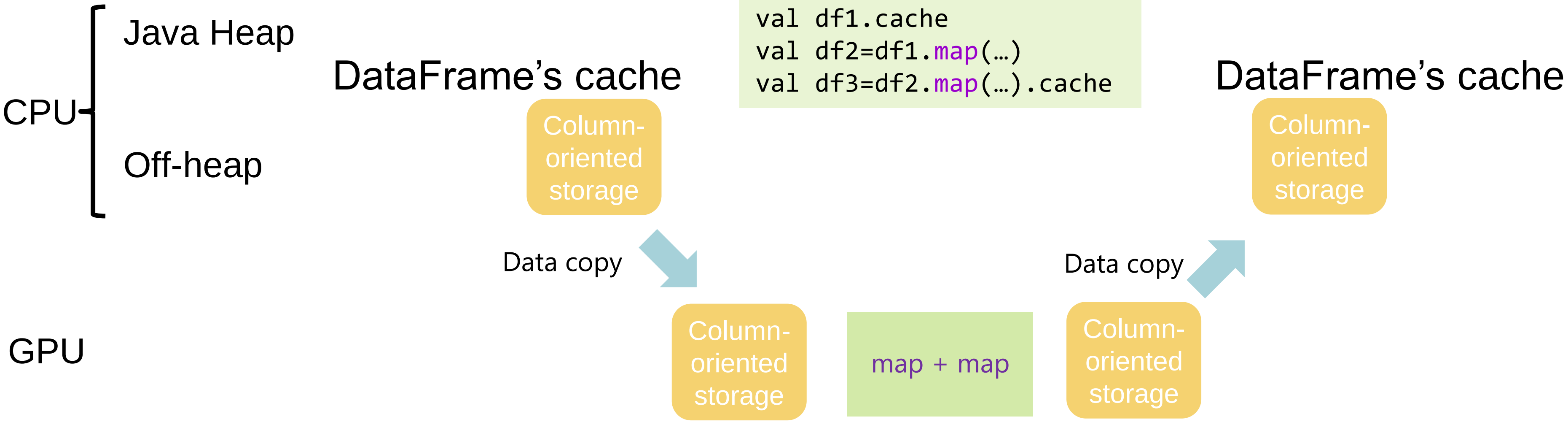
```
__global__ void multiplyBy2(int *inx, int *iny,
     int *outx, int *outy, long size) {
  long i = threadIdx.x + blockIdx.x * blockDim.x;
  if (size <= i) return;

  outx[i] = inx[i] * 2; outy[i] = iny[i] * 2;
}
```

Data Movement with DataFrame

- Cache in DataFrame already uses column-oriented storage
- We enhanced to create cache for DataFrame in off-heap
  - Reduce conversion and data copy between Java heap and Off-heap

```
val df1.cache
val df2=df1.map(…)
val df3=df2.map(…).cache
```



Data Movement with DataFrame

- Cache in DataFrame already uses column-oriented storage
- We enhanced to create cache for DataFrame in off-heap
  - Reduce conversion and data copy between Java heap and Off-heap

CPU
Java Heap
Off-heap

```
val df1.cache
```

DataFrame's cache

Column-oriented storage

Data copy

copy is automatically done

DataFrame's cache

Column-oriented storage

GPU

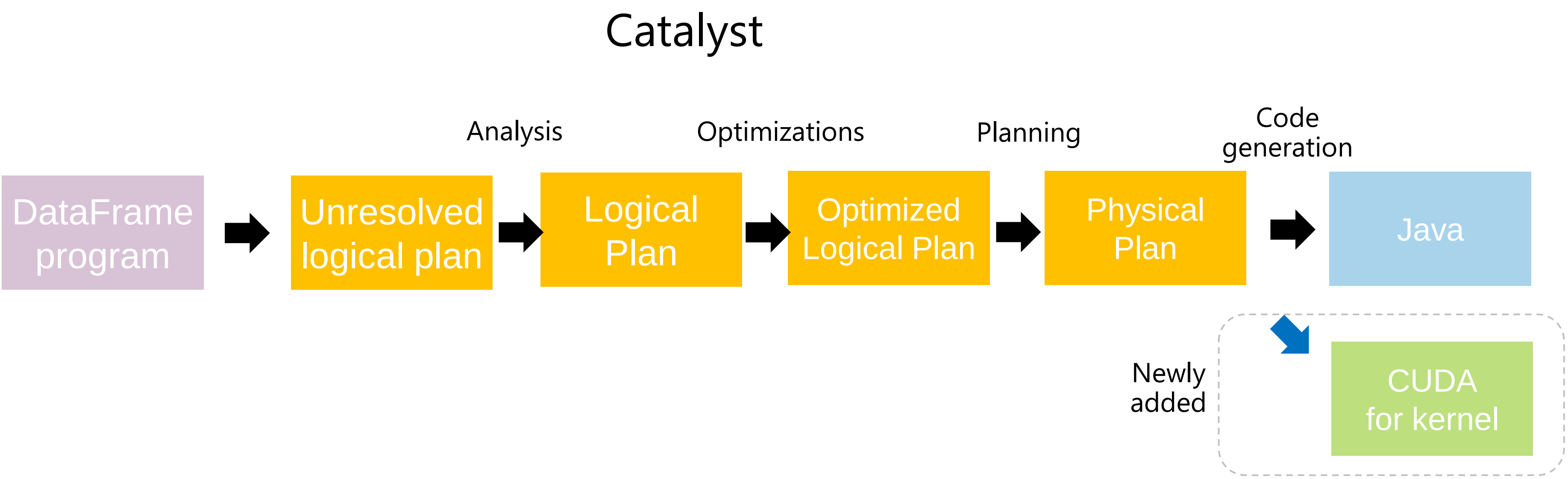
Column-oriented storage

map + map

Column-oriented storage

How To Generate GPU Code from DataFrame

- Added a new path to generate CUDA code into Catalyst



Derived Structuring Apache Spark 2.0: SQL, DataFrames, Datasets And Streaming - by Michael Armbrust

Takeaway

- Why accelerate your workloads using GPU on Spark
  - Achieved up to 3.6x over 160-CPU-thread parallel execution
- How to use GPUs on Spark
  - Invoke Hand-tuned GPU program in CUDA (GPUEnabler)
  - Translate DataFrame program to GPU code automatically
- How two approaches execute program on GPUs
  - Address easy programming for many non-experts, not the state-of-the-art performance by small numbers of top-notch programmers