Speed up big data processing via DSL

Agenda

- Background
- Big Data Process
- Domain Special Language
- Demo

Background

- Vision
  - Less domain knowledge for newbie
  - Visual environment to definite big data process logic
  - Work with variety of big data engine
- Challenge
  - Big data
  - Small team to support multiple projects
  - Offshore

Idea

- Use DSL engine to generate big data process logic

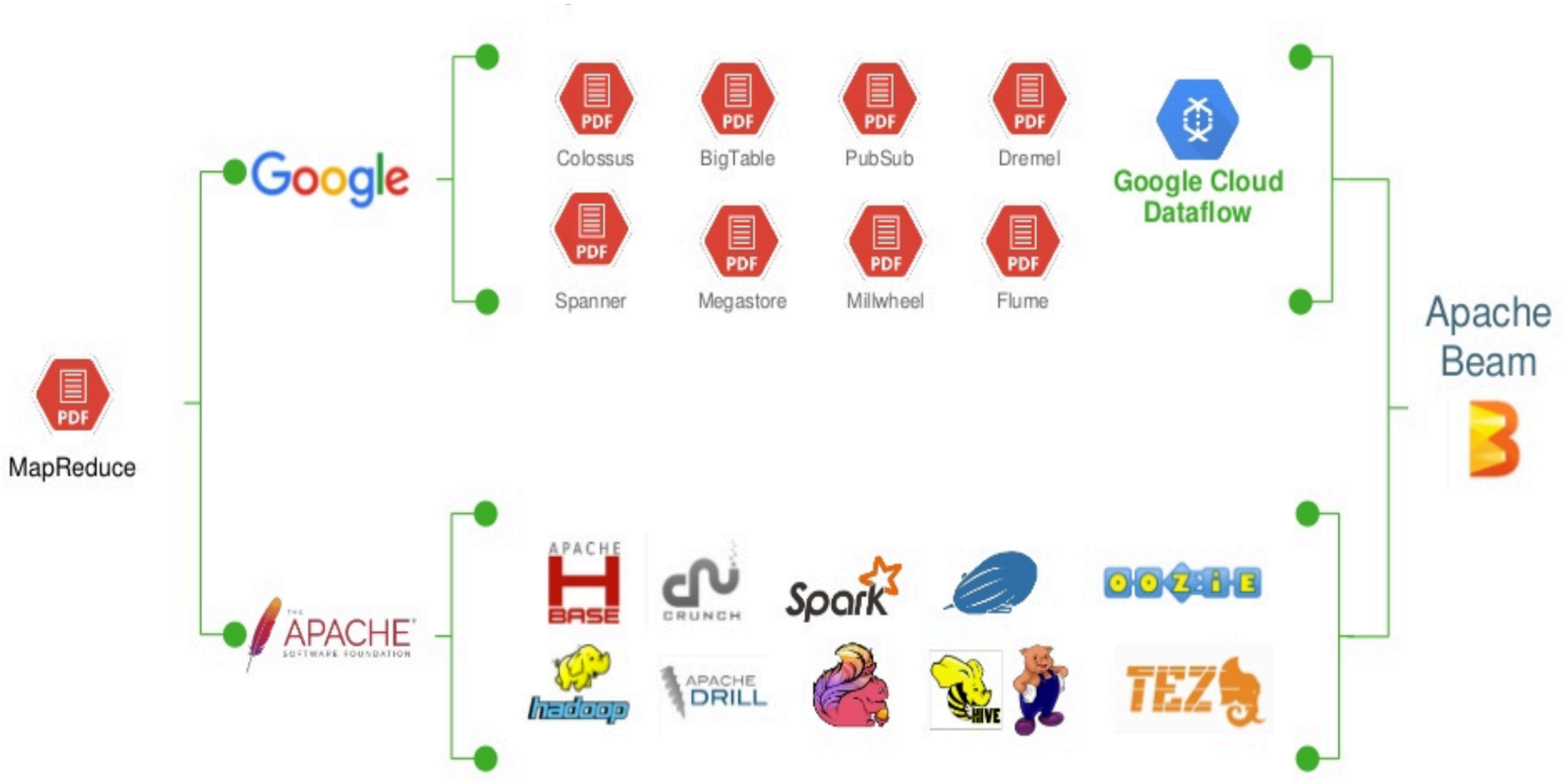
Big data Execution model

- Batch Process
  - Hadoop
  - Spark
- Stream Process
  - Storm
  - Spark Streaming
  - Flink

Big Data Program Model

- Map Reduce
- Spark
- Beam

The Evolution of Apache Beam



Beam

- An advanced unified programming model , Implement batch and streaming data processing jobs that run on any execution engine.

- Available Runners:
  - Apache Spark
  - Apache Apex
  - Apache Fink
  - Google Cloud Dataflow
  - Apache GearPump

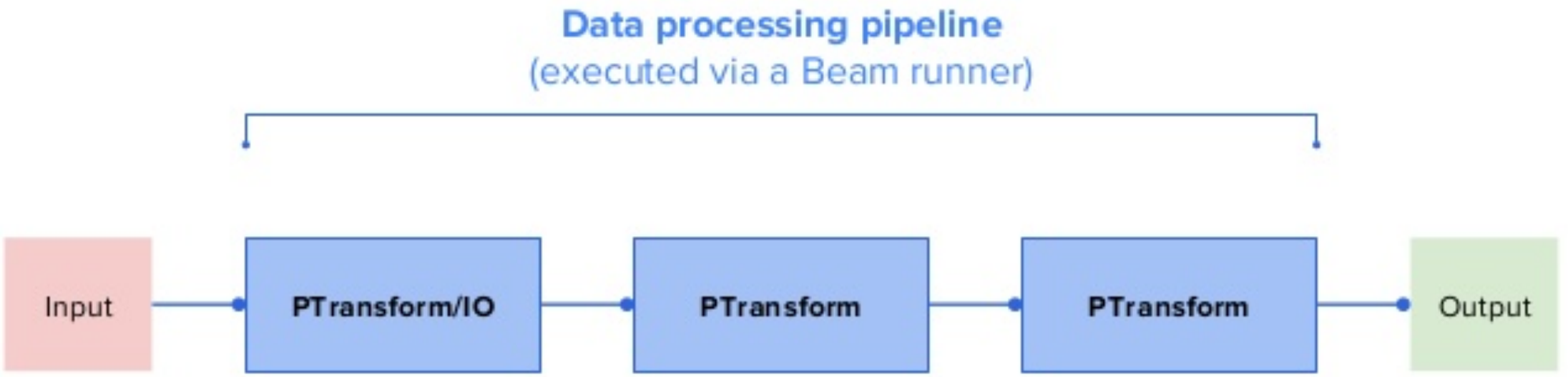
Beam

Beam Programming Model

1. **Pipelines** - data processing job as a directed graph of steps
2. **PCollection** - the data inside a pipeline
3. **Transform** - a step in the pipeline (taking PCollections as input, and produce PCollections)
    a. **Core transforms** - common transformation provided (ParDo, GroupByKey, ...)
    b. **Composite transforms** - combine multiple transforms
    c. **IO transforms** - endpoints of a pipeline to create PCollections (consumer/root) or use PCollections to "write" data outside of the pipeline (producer)

Beam

Beam Programming Model



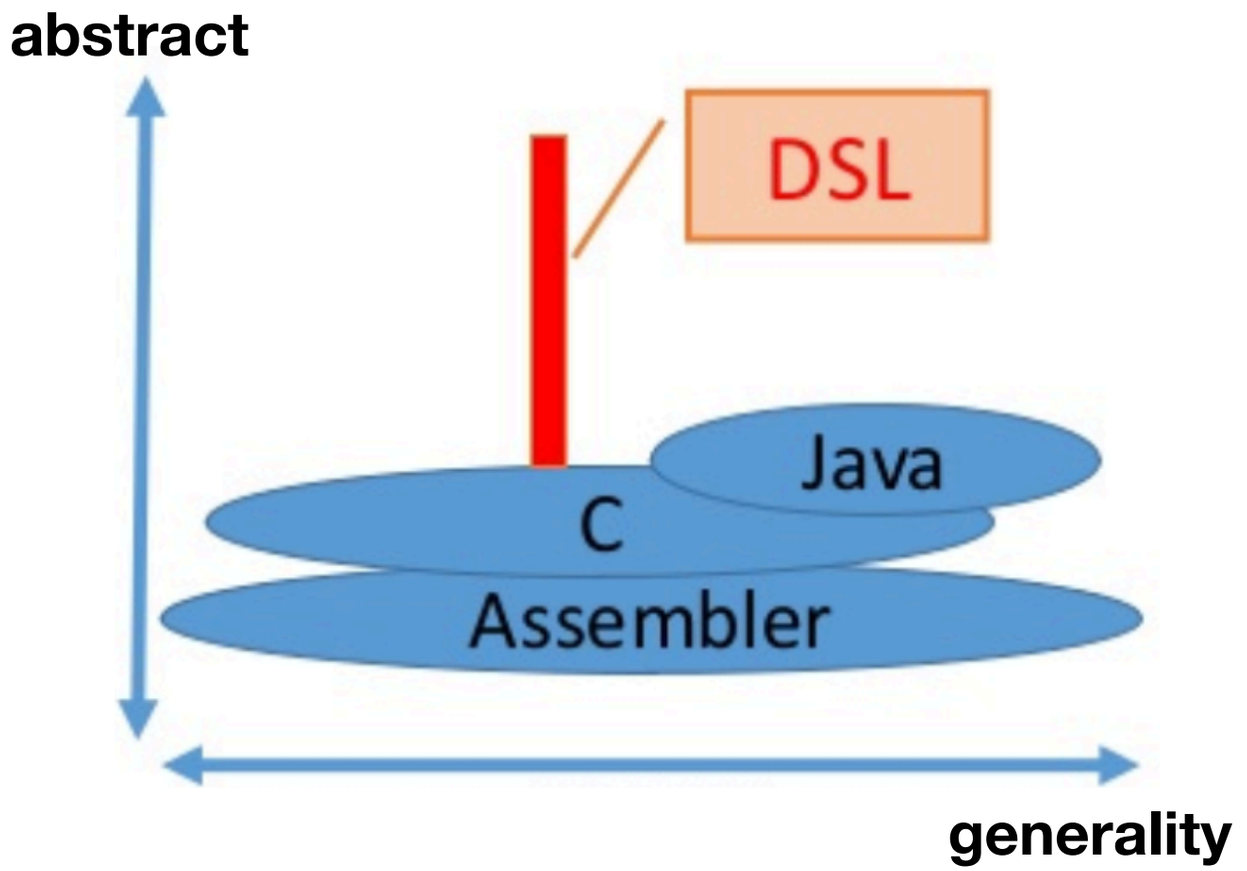
Beam

```
public static class CountWords extends PTransform<PCollection<String>,
    PCollection<KV<String, Long>>> {
  @Override
  public PCollection<KV<String, Long>> expand(PCollection<String> lines) {

    // Convert lines of text into individual words.
    PCollection<String> words = lines.apply(
        ParDo.of(new ExtractWordsFn()));

    // Count the number of times each word occurs.
    PCollection<KV<String, Long>> wordCounts =
        words.apply(Count.<~>perElement());

    return wordCounts;
  }
}
```

```
public static class FormatAsTextFn extends SimpleFunction<KV<String, Long>, String> {
  @Override
  public String apply(KV<String, Long> input) { return input.getKey() + ": " + input.getValue(); }
}

public static void main(String[] args) {
  WordCountOptions options = PipelineOptionsFactory.fromArgs(args).withValidation()
    .as(WordCountOptions.class);
  Pipeline p = Pipeline.create(options);

  p.apply( name: "ReadLines", TextIO.read().from(options.getInputFile()))
   .apply(new CountWords())
   .apply(MapElements.via(new FormatAsTextFn()))
   .apply( name: "WriteCounts", TextIO.write().to(options.getOutput()));

  p.run().waitUntilFinish();
}
```

Domain Special Language

- Well-known DSL : SQL, HTML, CSS, MATLAB…



How to build DSL

- Internal DSL. (ruby, groovy…)
- External DSL (antlr , EMF, xtext, …)

Xtext Framework

- A complete environment for development of textual program language and domain special language

- Implemented in Java and based on Eclipse, EMF and anltr.

Xtext Framework

- Xtext : Syntax definition language

- Xbase : base data type implement of java language

- Xtend : another JVM language used to write code generation logic

Xtext

```
grammar org.example.entities.Entities with org.eclipse.xtext.common.Terminals

generate entities "http://www.example.org/entities/Entities"

Model: entities += Entity*;

Entity: 'entity' name = ID ('extends' superType=[Entity])? '{'
        attributes += Attribute*
      '}' ;

Attribute: type=AttributeType name=ID ';';

AttributeType: elementType=ElementType (array ?='[' (length=INT)? ']')?;

ElementType: BasicType | EntityType;

BasicType: typeName=('string'|'int'|'boolean');

EntityType: entity=[Entity];
```

Xtend

```
class EntitiesGenerator extends AbstractGenerator {

    override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
        for (e : resource.allContents.toIterable.filter(typeof(Entity))) {
            fsa.generateFile("entities/" + e.name + ".java", e.compile)
        }
    }

    def compile(Entity entity){
        '''
            package entities;

            public class «entity.name» «IF entity.superType != null»extends «entity.superType.name» «ENDIF»{
                «FOR attribute : entity.attributes»
                    private «attribute.type.compile» «attribute.name»;
                «ENDFOR»

                «FOR attribute : entity.attributes»
                    public «attribute.type.compile» get«attribute.name.toFirstUpper»(){
                        return «attribute.name»;
                    }
                    public void set«attribute.name.toFirstUpper»(«attribute.type.compile» _arg) {
                        this.«attribute.name» = _arg;
                    }
                «ENDFOR»
            }
        '''
    }

    def compile(AttributeType attributeType) {
        attributeType.elementType.typeToString +
        if(attributeType.array)   "[]" else ""
    }

    def dispatch typeToString(BasicType type) {
        if(type.typeName == "string") "String" else type.typeName
    }

    def dispatch typeToString(EntityType type) {
        type.entity.name
    }
}
```

Demo

```
entity Person {
    int id;
    string name;
}

entity Group {
    Person[] members;
}
```

```
package entities;

public class Group {
    private Person[] members;

    public Person[] getMembers(){
        return members;
    }
    public void setMembers(Person[] _arg) {
        this.members = _arg;
    }
}
```

```
package entities;

public class Person {
    private int id;
    private String name;

    public int getId(){
        return id;
    }
    public void setId(int _arg) {
        this.id = _arg;
    }
    public String getName(){
        return name;
    }
    public void setName(String _arg) {
        this.name = _arg;
    }
}
```

QUESTIONS ?

THANKS !