23RD INTERNATIONAL EUROPEAN CONFERENCE
ON PARALLEL AND DISTRIBUTED COMPUTING

Santiago de Compostela, 29 August 2017

PiCo - **Pi**peline **Co**mposition

a Novel Approach to Stream Data Analytics

Claudia Misale, Maurizio Drocco, Guy Tremblay, Marco Aldinucci

UNIVERSITÀ DEGLI STUDI DI TORINO

UQÀM | Université du Québec à Montréal

REPHRASE

Introduction

- I wanna be a **Big Data Rockstar**
  1. pick random “Big Data” features
  2. work hard on Look and Feel
  3. find endorsements

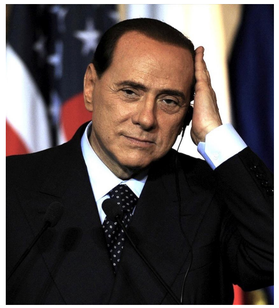
Introduction

- I wanna be a ~~Rockstar~~ **Scientist**
  - "*La mathématique est l'art de donner le même nom à des choses différentes.*"

    — Henry Poincaré

    "Mathematics is the art of giving the same name to different things."
  - a.k.a. abstraction

Example



Abstraction:
nice politicians with weird hair

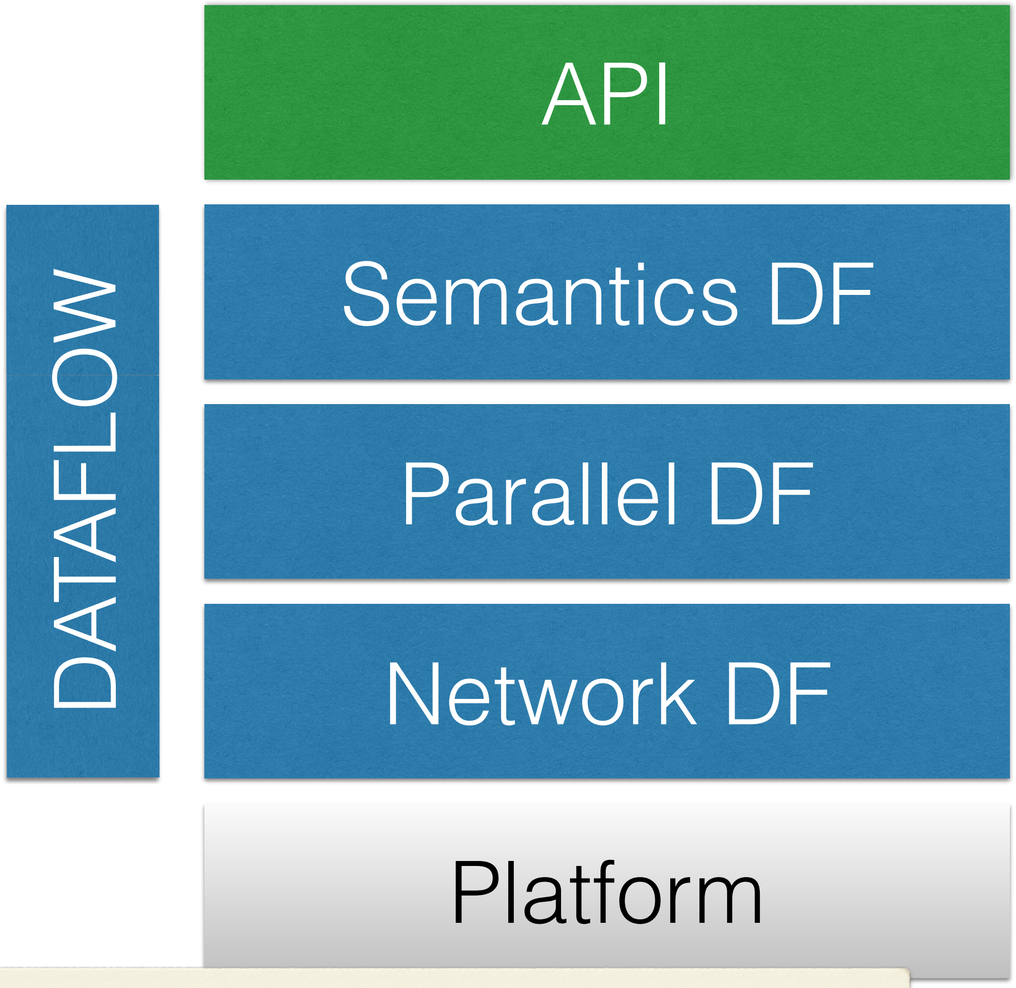
Big Data Abstraction - WHY

- Different names/metaphors for the very same concepts
  - e.g., MapReduce = FlatMap + ReduceByKey
- Mixed semantics/implementation
  - e.g., repartition primitive, first-n over unordered collections
  - sorting in MapReduce semantics [OSDI 04]
- Conclusion: hard to tell what a program means

[OSDI 04] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”

Big Data Abstraction - HOW

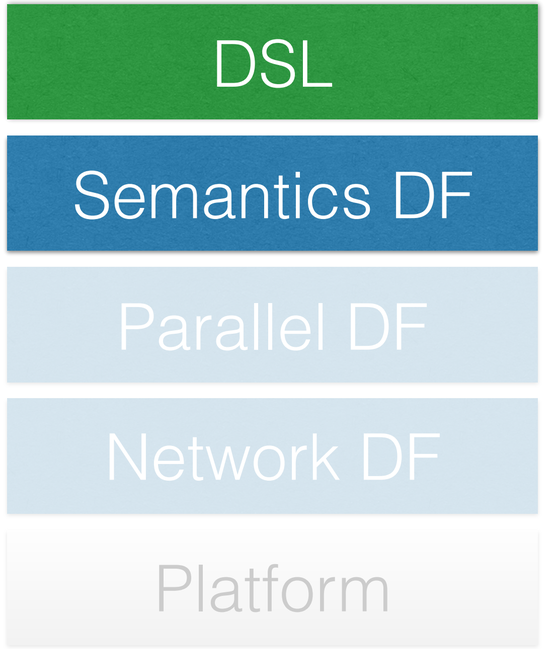
- Semantics:
  node = operation,
  edge = dependency
- Parallel:
  node = logical processor,
  edge = logical channel
- Network:
  node = OS thread, process
  edge = shmem/network channel



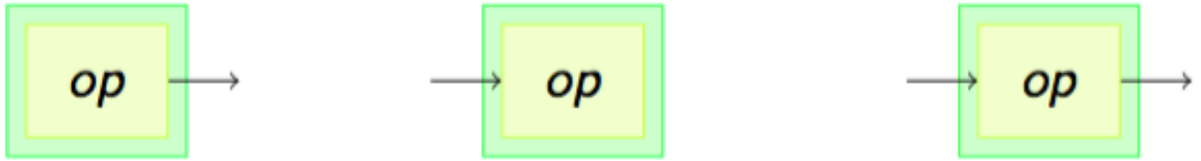
[PPL 17] C. Misale et al., “A Comparison of Big Data Frameworks on a Layered Dataflow Model”

PiCo — C++ API

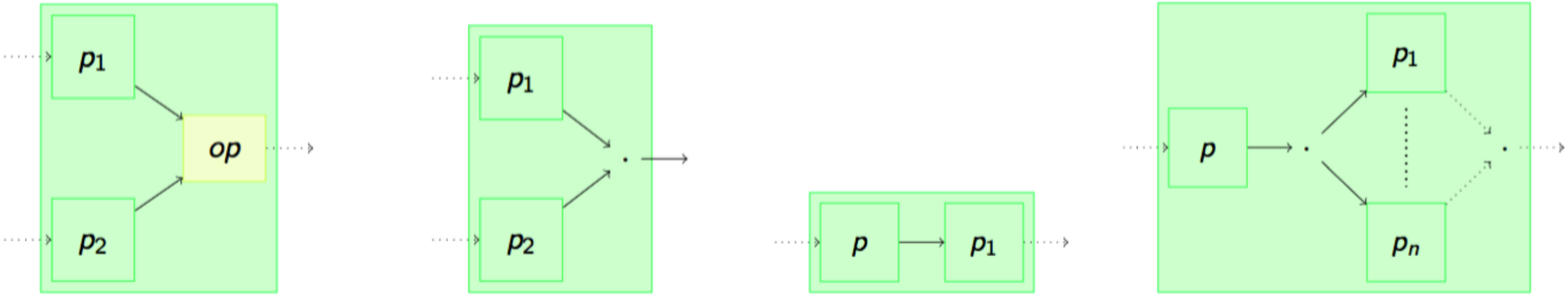
- Algebra of **Pipelines** and Operators
- Unified model for batch and stream processing
- Clear functional semantics
- RISC (vs Big Data Rockstars)



PiCo — C++ DSL (Pipelines)



(a) Source (b) Sink (c) Processing



(a) PAIR (b) MERGE (a) Linear TO (b) Non-linear TO

PiCo — C++ DSL (Operators)

- Source/Sink
  - From/to file, network…
- Map/Reduce
  - and Binary variants
- Modifiers
  - Windowing
  - Partitioning (e.g., by-key)

map	unary
combine	unary
b-map	binary
b-combine	binary
emit	produce-only
collect	consume-only

Word Count in PiCo

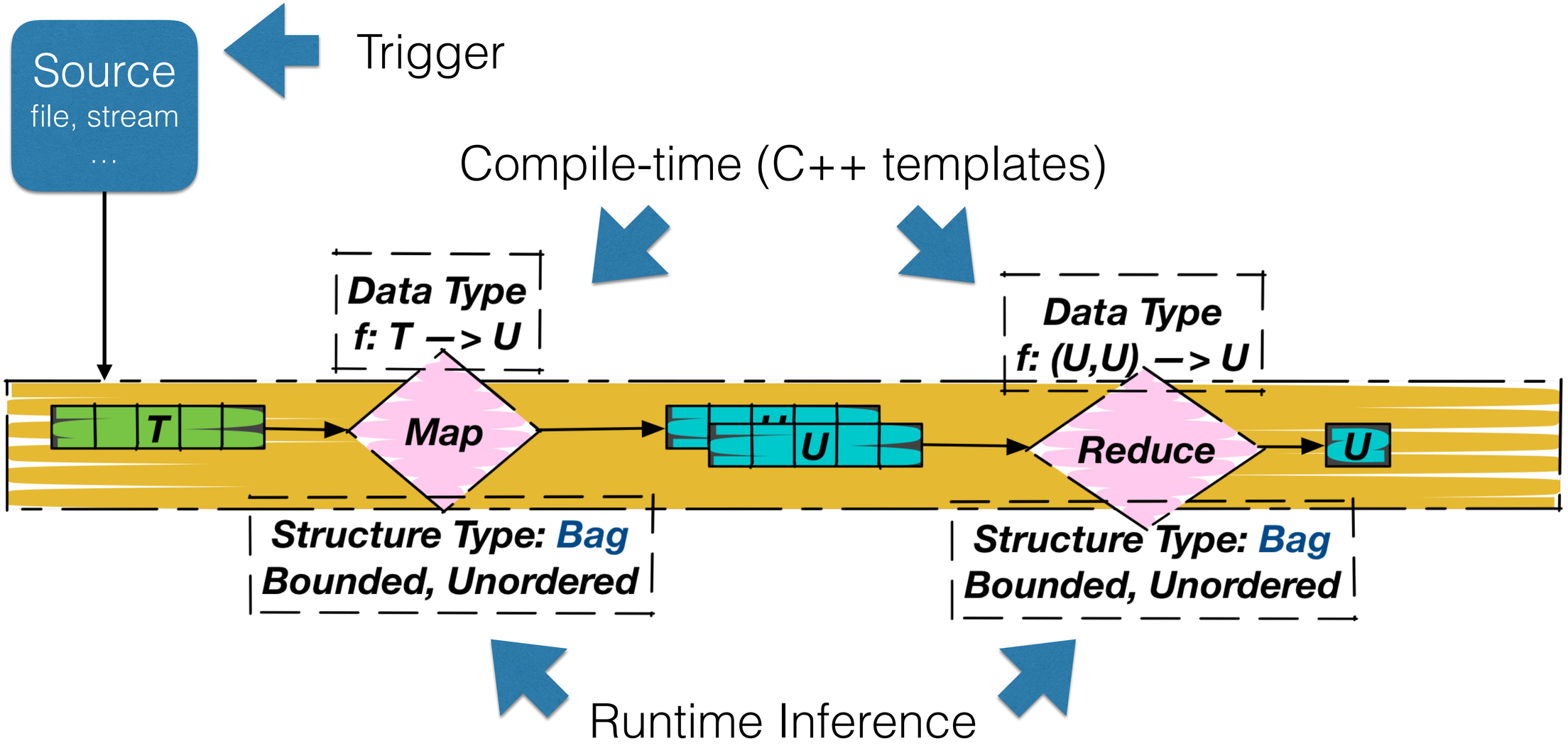
```
Pipe CountWords;

CountWords
    .add(FlatMap<string,string>(tokenizer))
    .add(Map<string,KV>([](string in){return KV(in,1);}))
    .add(PReduce<KV>([](KV a, KV b){return a + b;}))
```

Word Count in Spark

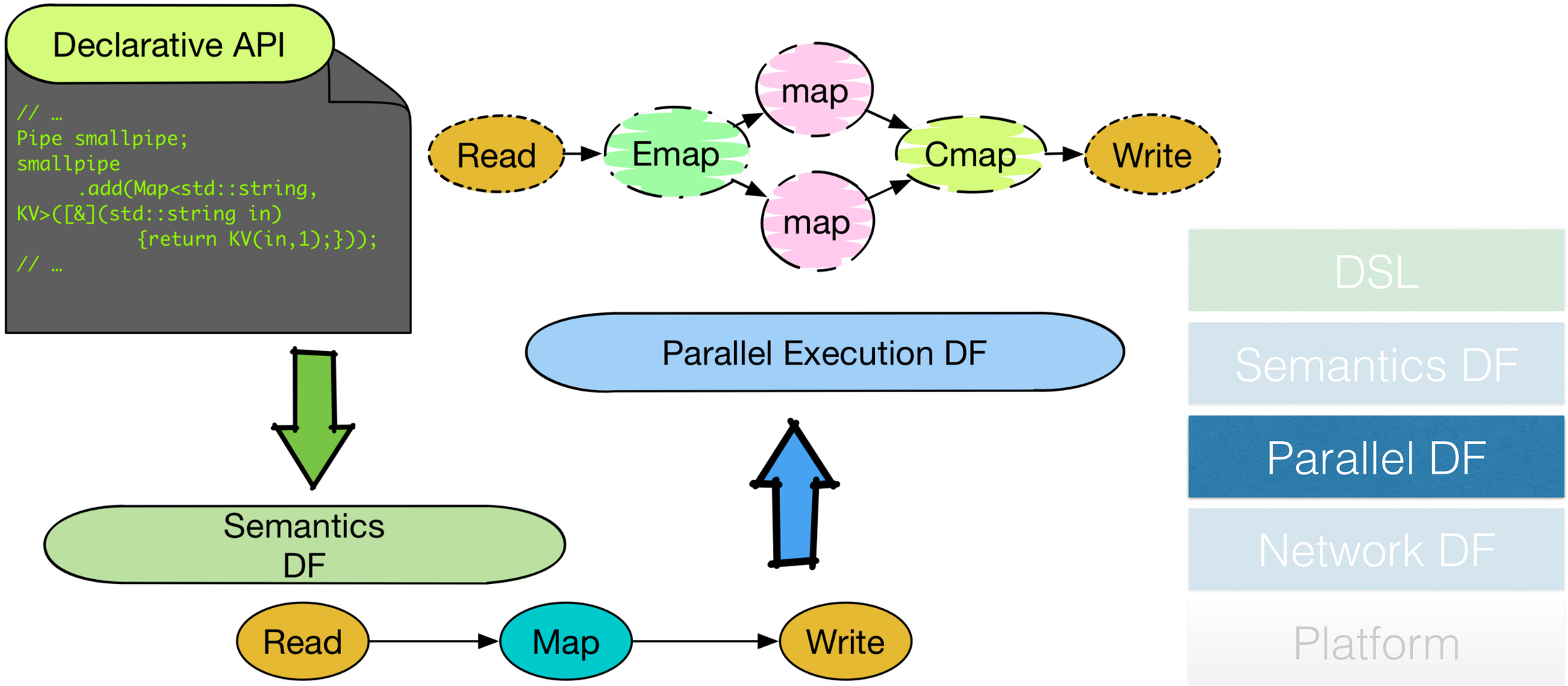
```
JavaRDD<String> textFile = sc.textFile("hdfs://...");

JavaPairRDD<String, Integer> counts = textFile

    .flatMap(tokenizer)

    .mapToPair(word -> new Tuple2<>(word, 1))

    .reduceByKey((a, b) -> a + b);
```

explicit data collections

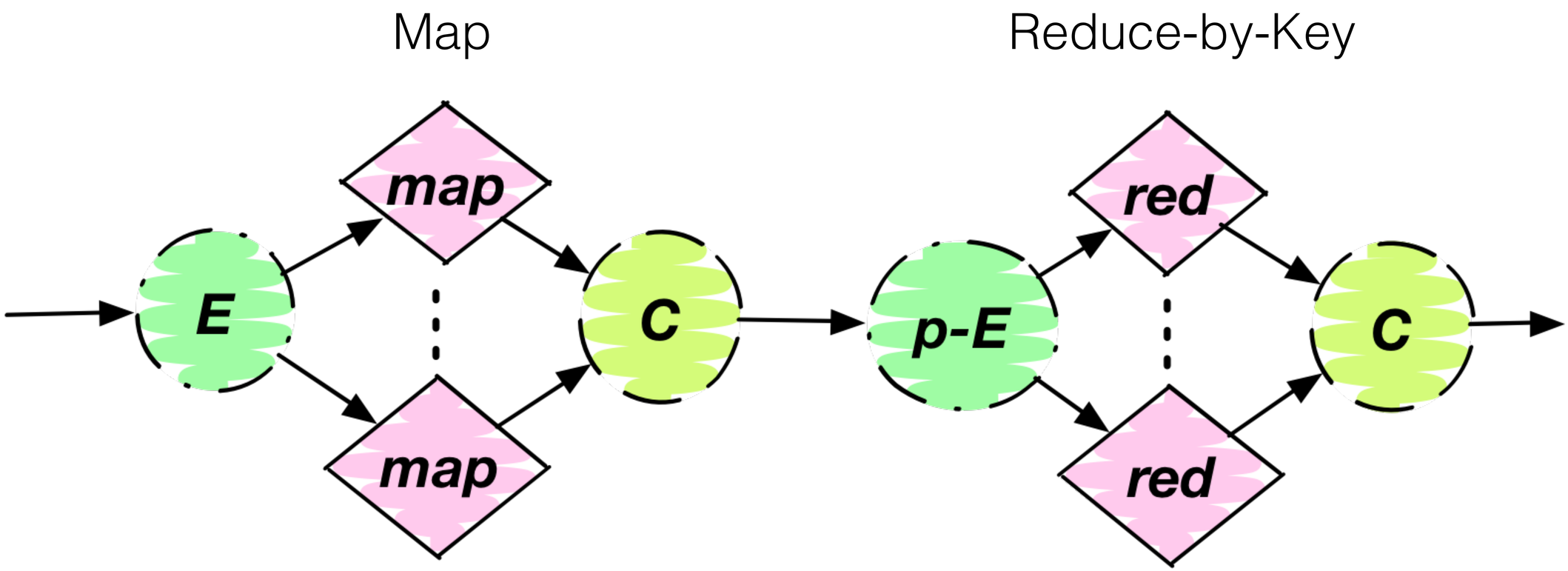
Polymorphism



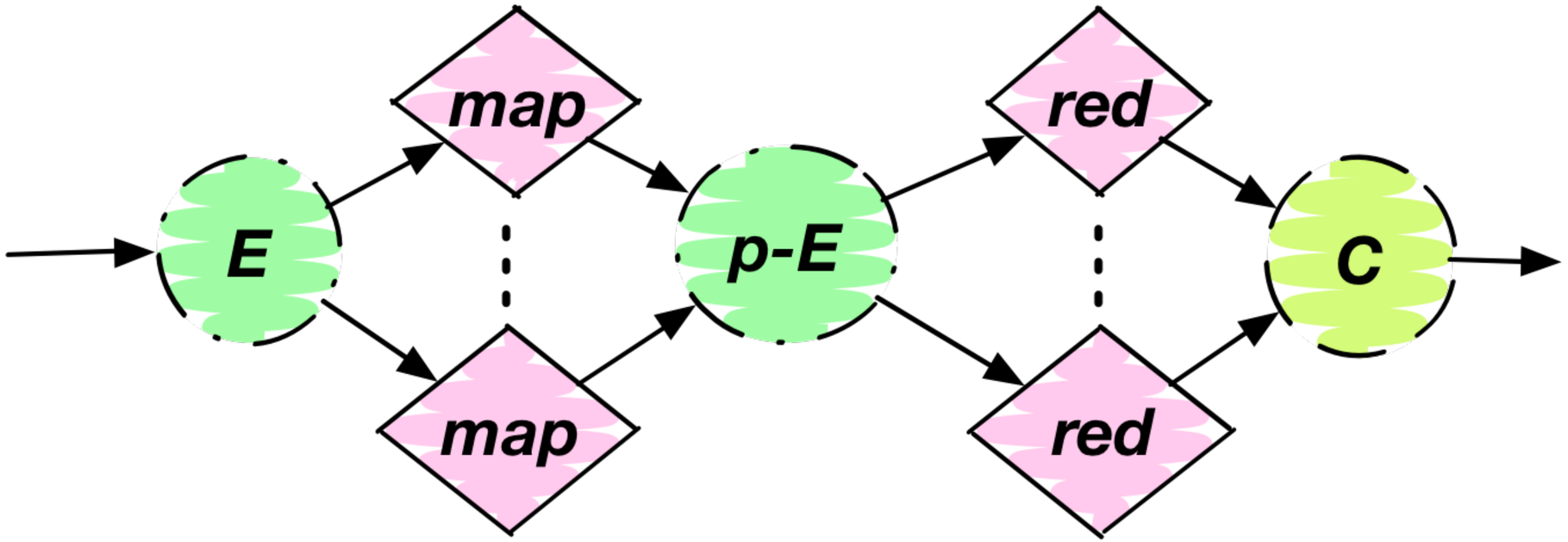
Parallelism-aware IR



IR Optimisation

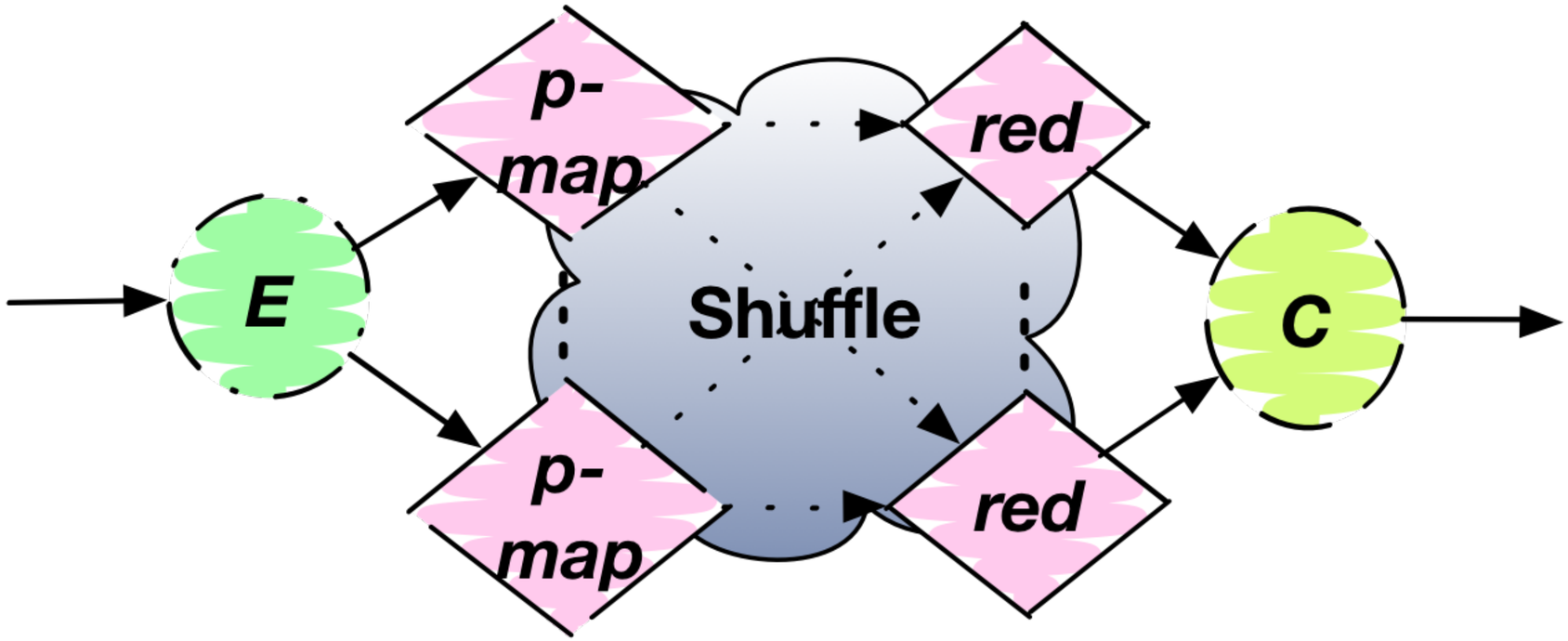


IR Optimisation



Collector-Emitter fusion

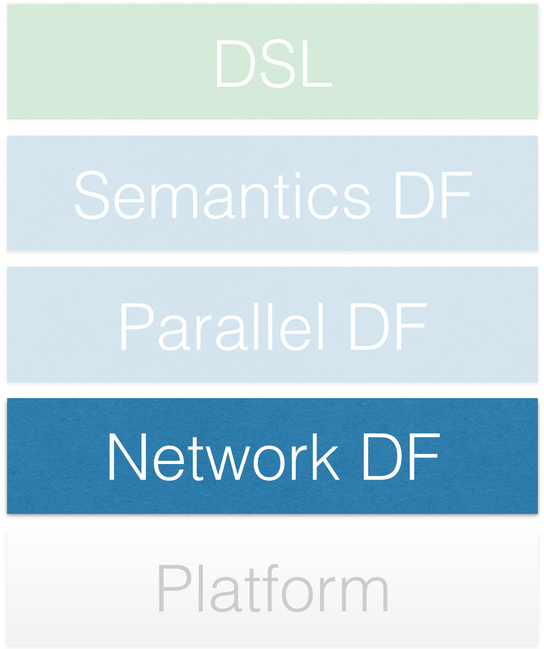
IR Optimisation



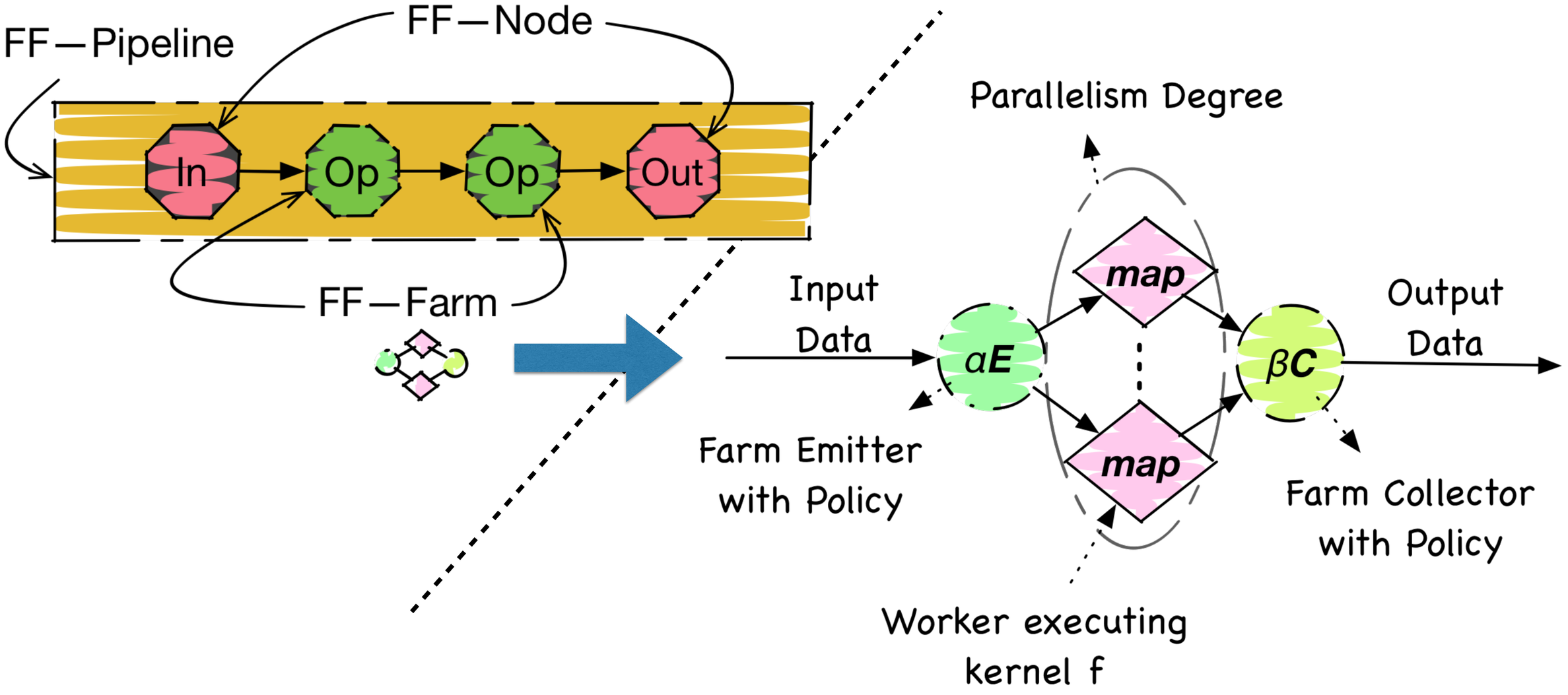
Dispatcher elimination

Shared-Memory Implementation

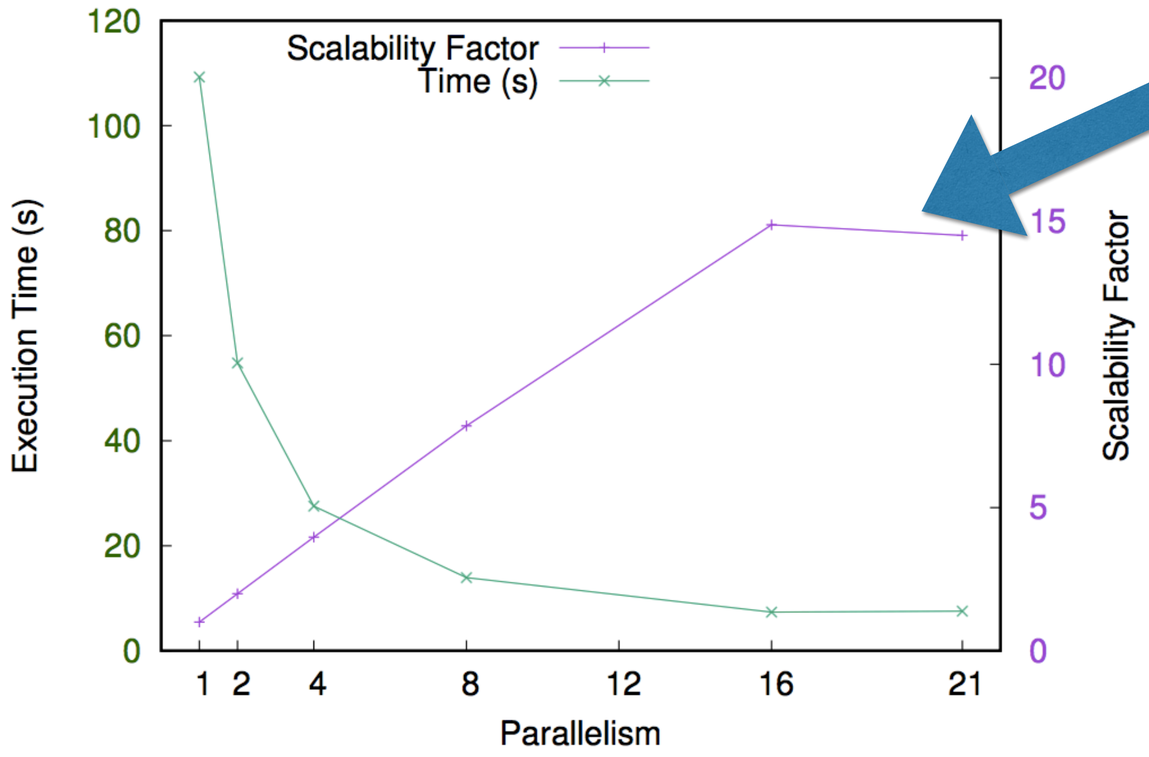
- FastFlow Networks
  - header-only C++ library
  - Streaming patterns (farm, pipeline)
  - State-of-the-art SPSC queues
- Everything is streaming
  - no tasks, no master-workers



Shared-Memory Implementation



Performances — Streaming HFT



To be improved:
Memory allocation

workers	exec. time	read from socket	map worker	map scalability
1	109184.00	3962.74	109184.00	1.00
2	54747.90	3979.72	54747.50	1.99
4	27426.10	4064.94	27423.40	3.98
8	13927.80	4245.09	13926.20	7.84
16	7386.69	6341.60	7277.45	15.00
21	7548.15	7539.83	7545.49	14.47

Performances — Streaming HFT

- Fast
  - C++ little runtime overhead
- Scalable
  - Streaming on top of FastFlow (state-of-the-art performance)
- Low memory footprint
  - Better memory management than Java

	Flink	**Spark**	**PiCo**
Min. Exec. Time	24.78 s	42.22 s	**7.35 s**
Relative Speedup	9.21	2.24	**14.87**
CPU %	14.31	10.23	**38.85**
Memory Footprint	4.88 GB	3.17 GB	**315 MB**

Future Work

- Distributed platforms
  - idea: C++ smart global pointers — i.e., my PhD thesis
- Exploit GPUs
  - idea: exploit FastFlow+CUDA and FastFlow+OpenCL API
- Better memory allocation
  - idea: exploit pattern-aware FastFlow allocator

Thank you