Weld: A Common Runtime for Data Analytics

Shoumik Palkar, James Thomas, Anil Shanbhag*, Deepak Narayanan, Malte Schwarzkopf*, Holger Pirk*, Saman Amarasinghe*, Matei Zaharia

Stanford InfoLab, *MIT CSAIL

Motivation

Modern data apps combine many disjoint processing libraries & functions

- » Relational, statistics, machine learning, …
- » E.g. PyData stack

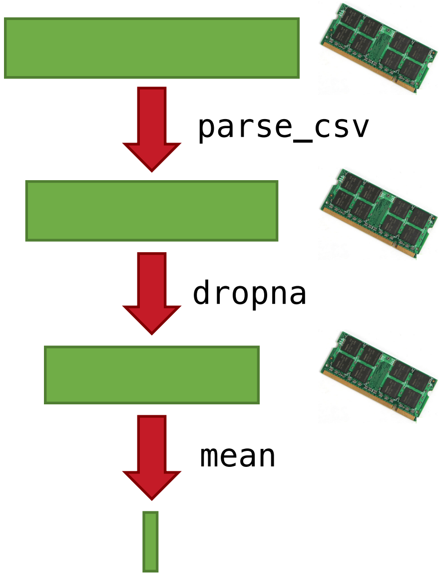
+ Great results leveraging work of 1000s of authors

– No optimization across these functions

How Bad is This Problem?

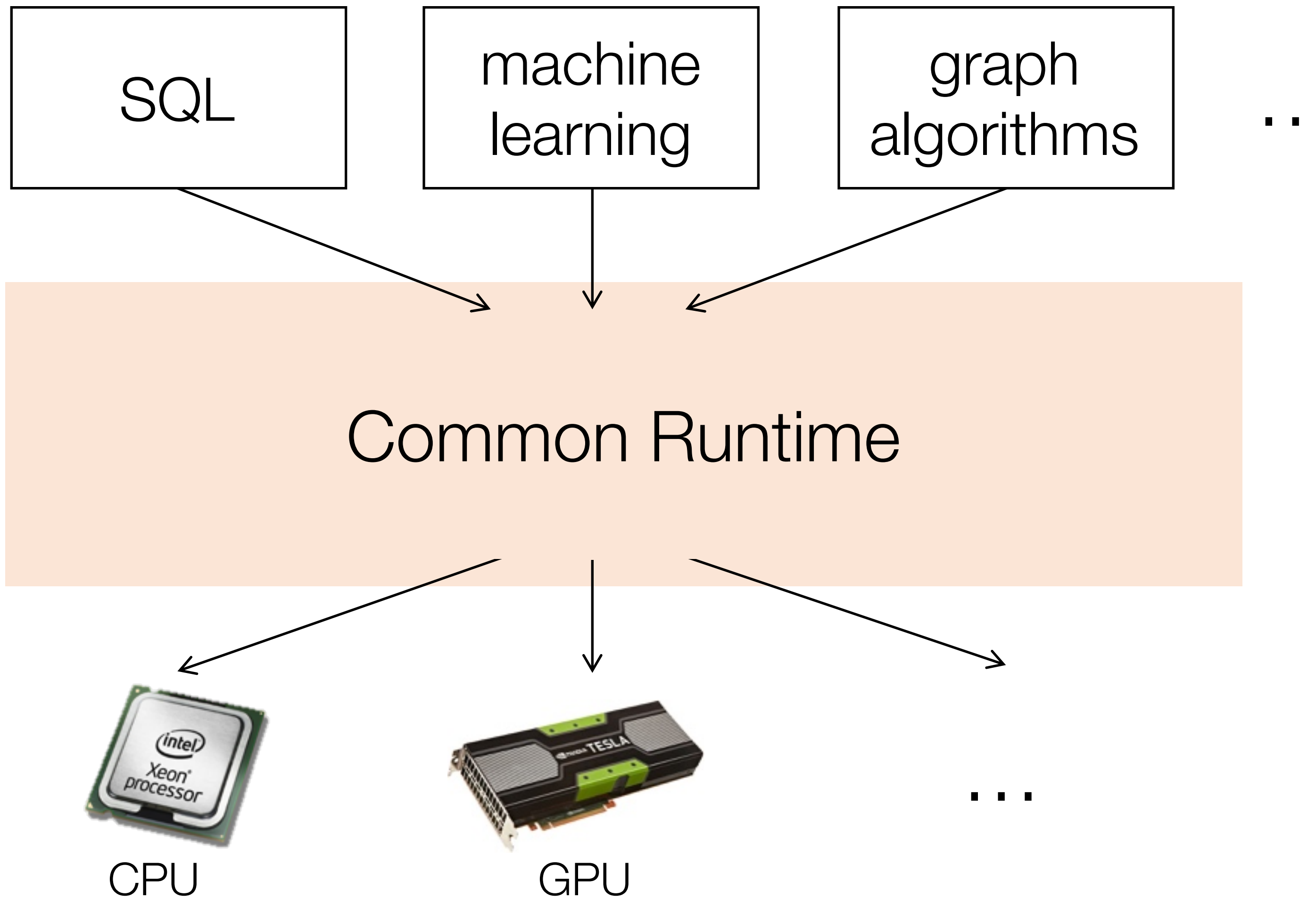
Growing gap between memory/processing makes traditional way of combining functions worse

```
data = pandas.parse_csv(string)

filtered = pandas.dropna(data)

avg = numpy.mean(filtered)
```

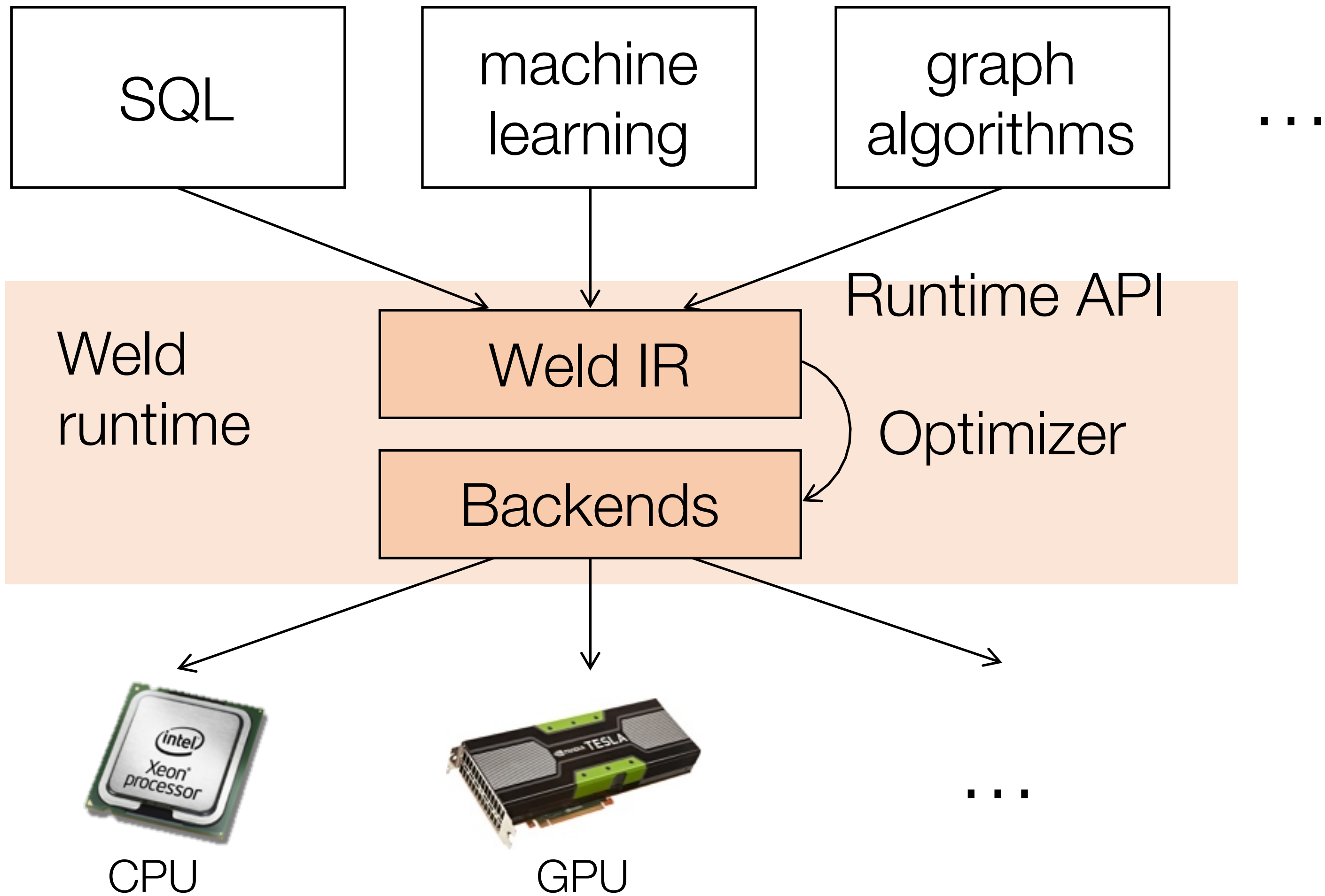


5-30x slowdowns in NumPy, Pandas, TensorFlow, etc

How We Solve This

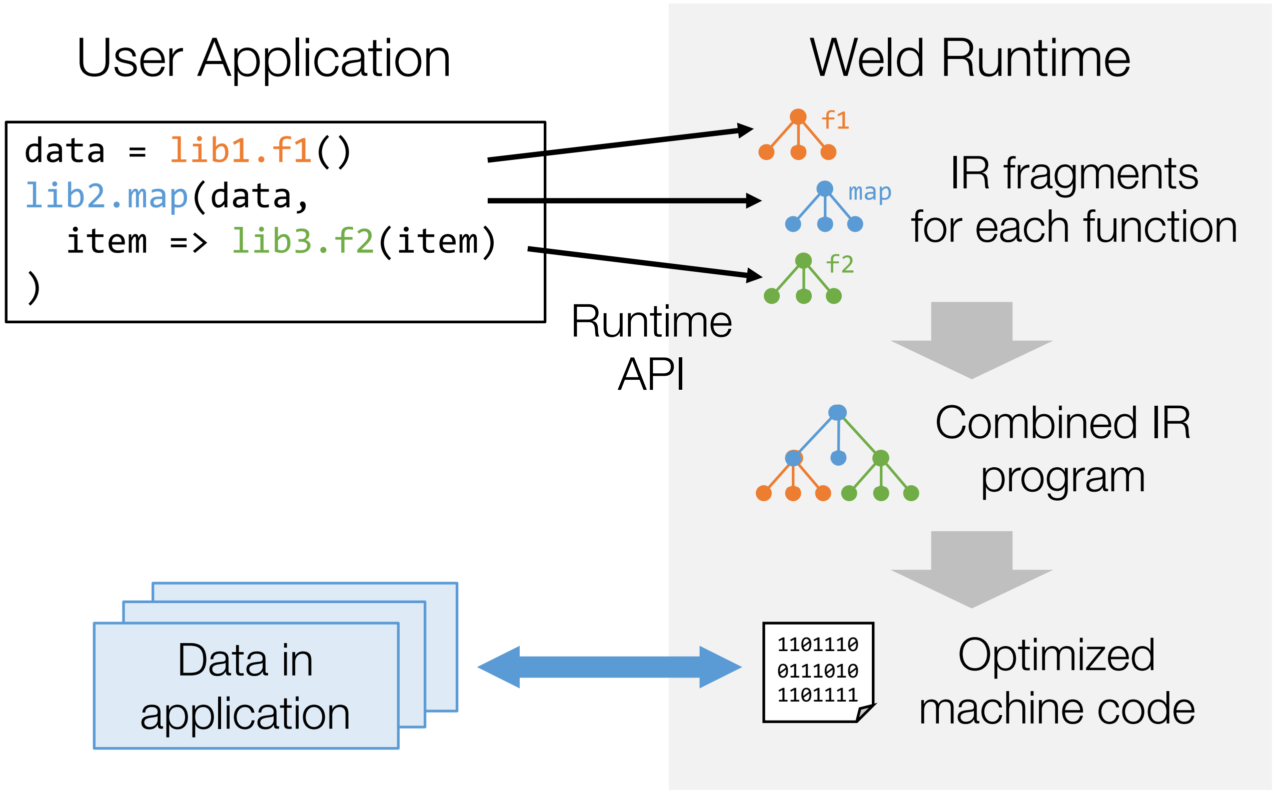


How We Solve This



Runtime API

Uses lazy evaluation to collect work across libraries



Weld IR

Designed to meet three goals:

1. **Library composition:** support complete workloads such as nested parallel calls

2. **Ability to express optimizations:** e.g. loop fusion, vectorization, loop tiling

3. **Explicit parallelism and targeting parallel hardware**

Weld IR

Small, powerful design inspired by “monad comprehensions”

Parallel loops: iterate over a dataset

Builders: declarative objects for producing results

- » E.g. append items to a list, compute a sum
- » Can be implemented differently on different hardware

Captures relational algebra, functional APIs like Spark, linear algebra, and composition thereof

Examples

Implement functional operators using builders

```
def map(data, f):
    builder = new vecbuilder[int]
    for x in data:
        merge(builder, f(x))
    result(builder)


def reduce(data, zero, func):
    builder = new merger[zero, func]
    for x in data:
        merge(builder, x)
    result(builder)
```

Example Optimization: Fusion

```
squares = map(data, x => x * x)
sum = reduce(data, 0, +)
```

```
bld1 = new vecbuilder[int]
bld2 = new merger[0, +]
for x in data:
    merge(bld1, x * x)
    merge(bld2, x)
```

Loops can be merged into one pass over data

Heterogeneous Hardware

A platform for heterogeneous computing.

One example: Creating a storage engine using FPGAs.

- Loading data from a persistent format into a memory format is often compute bound.
- Weld can accelerate these workloads

Implementation

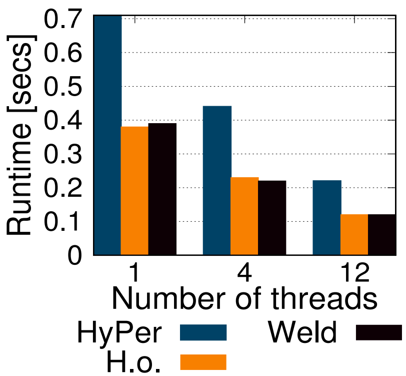
Prototype with APIs in Scala and Python

» LLVM and Voodoo for code gen

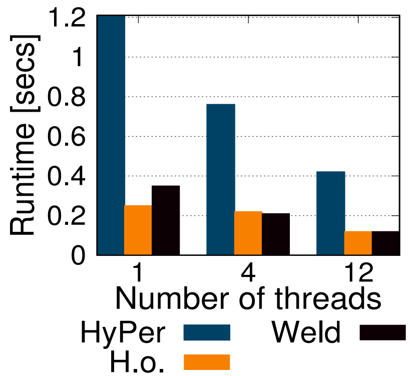
Integrations: TensorFlow, NumPy, Pandas, Spark

Results: Individual Workloads

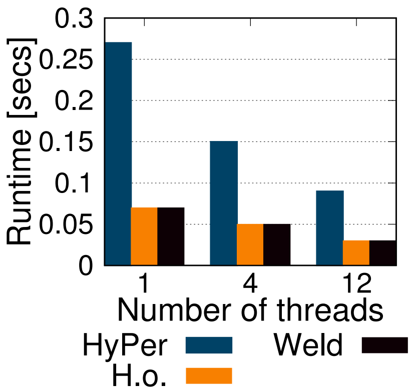
SQL (TPC-H)



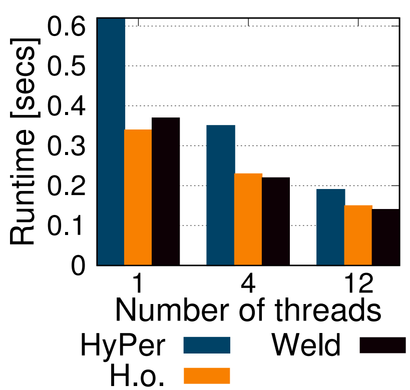
Q1



Q3

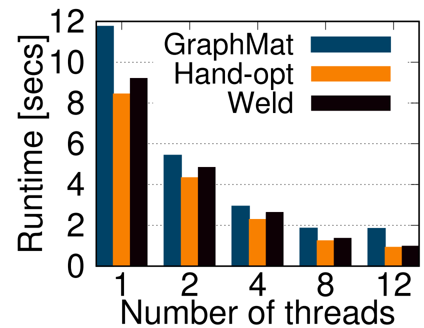


Q6

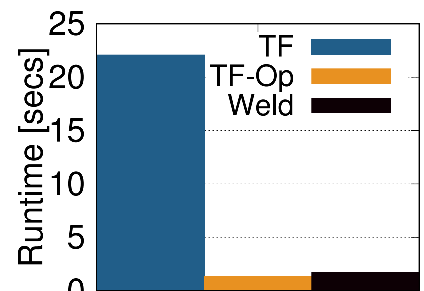


Q12

PageRank

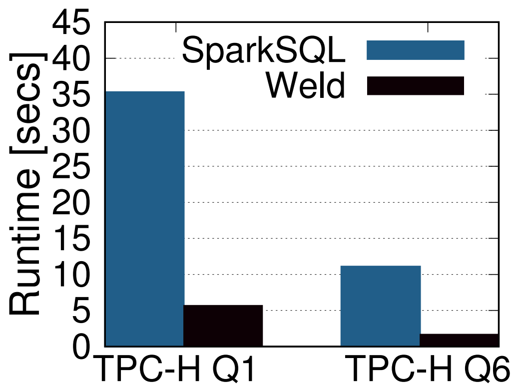


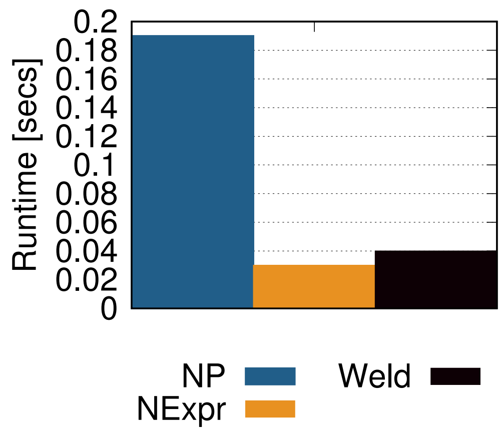
Word2Vec

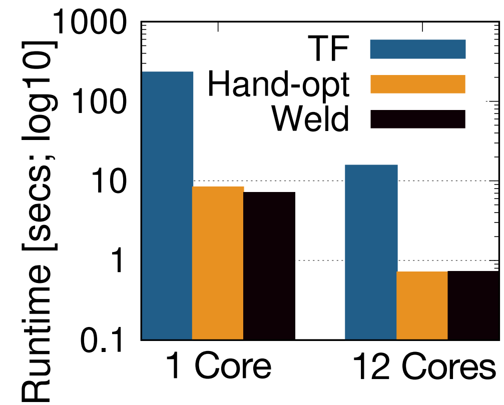


TF-Op = C++ operator

Results: Existing Frameworks







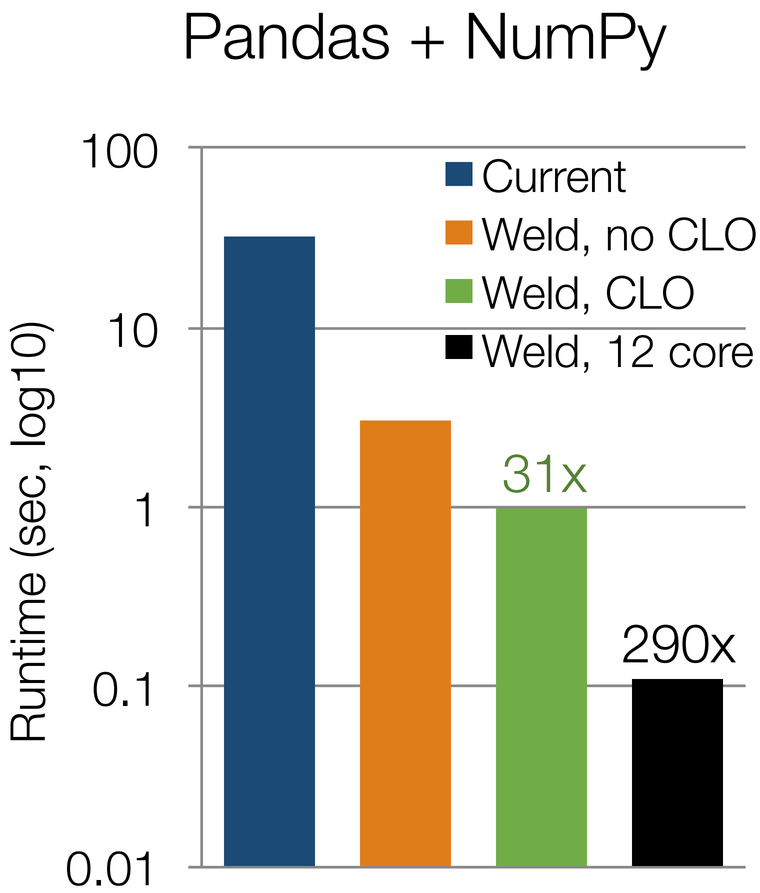
TPC-H

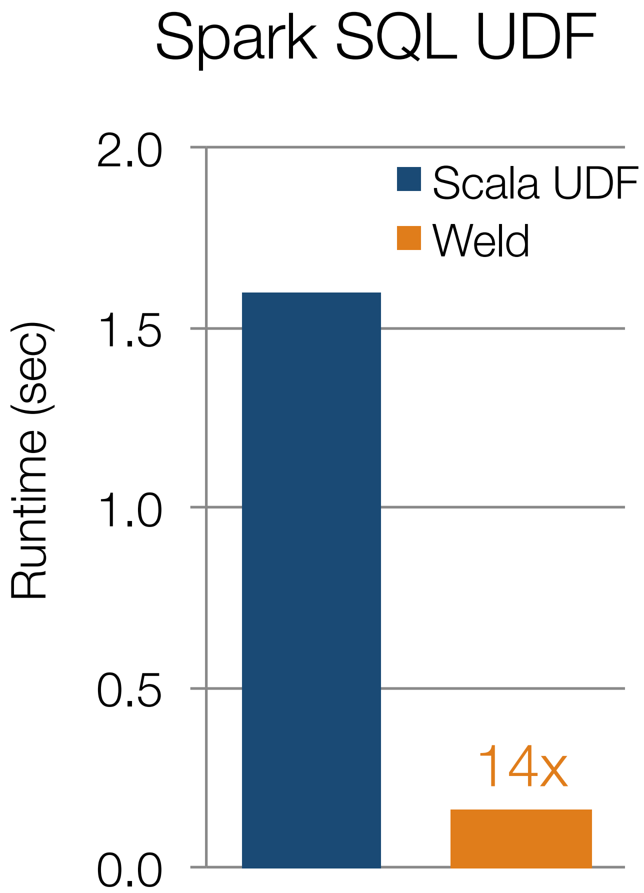
Vector Sum

Logistic Regression

Integration effort: 500 lines glue, 30 lines/operator

Results: Cross-Library Optimization





Conclusion

The way we compose software will have to change to efficiently use modern hardware

Lots of open questions and design decisions!

» Leveraging specialized hardware, domain info, …

Open source: soon!

STANFORD INFOLAB

Why Don't Compilers Solve This?

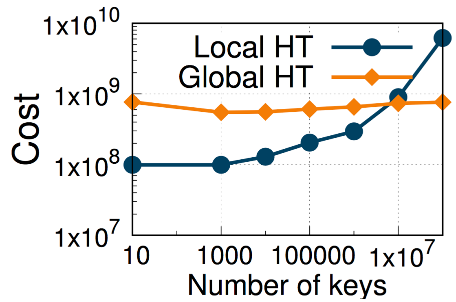
Languages and intermediate representations (IRs) make it hard to optimize across libraries

- Main abstraction is shared memory
 (must worry about aliasing, order, etc)

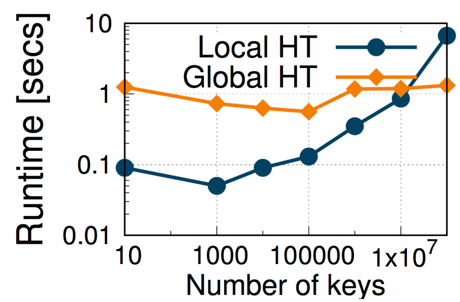
Most compilers don't model parallel operations

- Makes high performance code generation for heterogeneous parallel hardware even more difficult

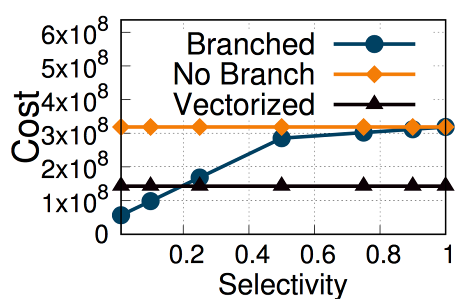
Results: Modeling Costs



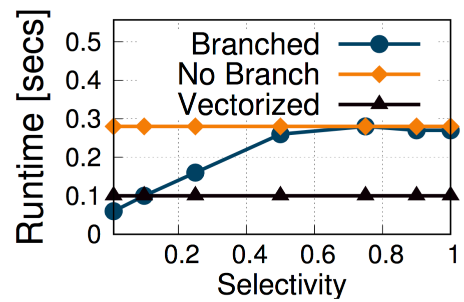
(a) TPCH Q1 Cost Estimates.



(b) TPCH Q1 Runtimes.



(e) TPCH Q6 Cost Estimates.



(f) TPCH Q6 Runtimes.

Takeaway: Cost curves *resemble* actual runtimes

Cost Model

Inspired by cost models for in-memory databases

+ modeling nested parallelism and choices among implementations of data structures

Models cache contention, costs of atomic instructions, etc.

Builders

Hardware independent and explicitly parallel

Three operators:

`merge(builder, value):` Merge a value into the builder and return a new builder

`result(builder):` destroy the builder and return a value

`for(data, builders, func):` iterate over data, potentially merging values into one or more builders in parallel

Optimizer

Cost Based Optimizer similar to an RDBMS

Builder Implementations: How to implement a particular builder (e.g., global vs. local hash tables)

Transforms: Should expressions be fused, vectorized, inlined, etc.

Quantifies choices among optimizations using data from the program

Related Work

HyPer, LegoBase, Tupleware: target relational algebra and serial UDFs; no nested parallelism

LLVM, OpenCL: low-level shared-memory model

NESL, parallel FPs: not closed under optimizations

DSLs: Weld focuses on integration with existing libraries and cross-library optimization

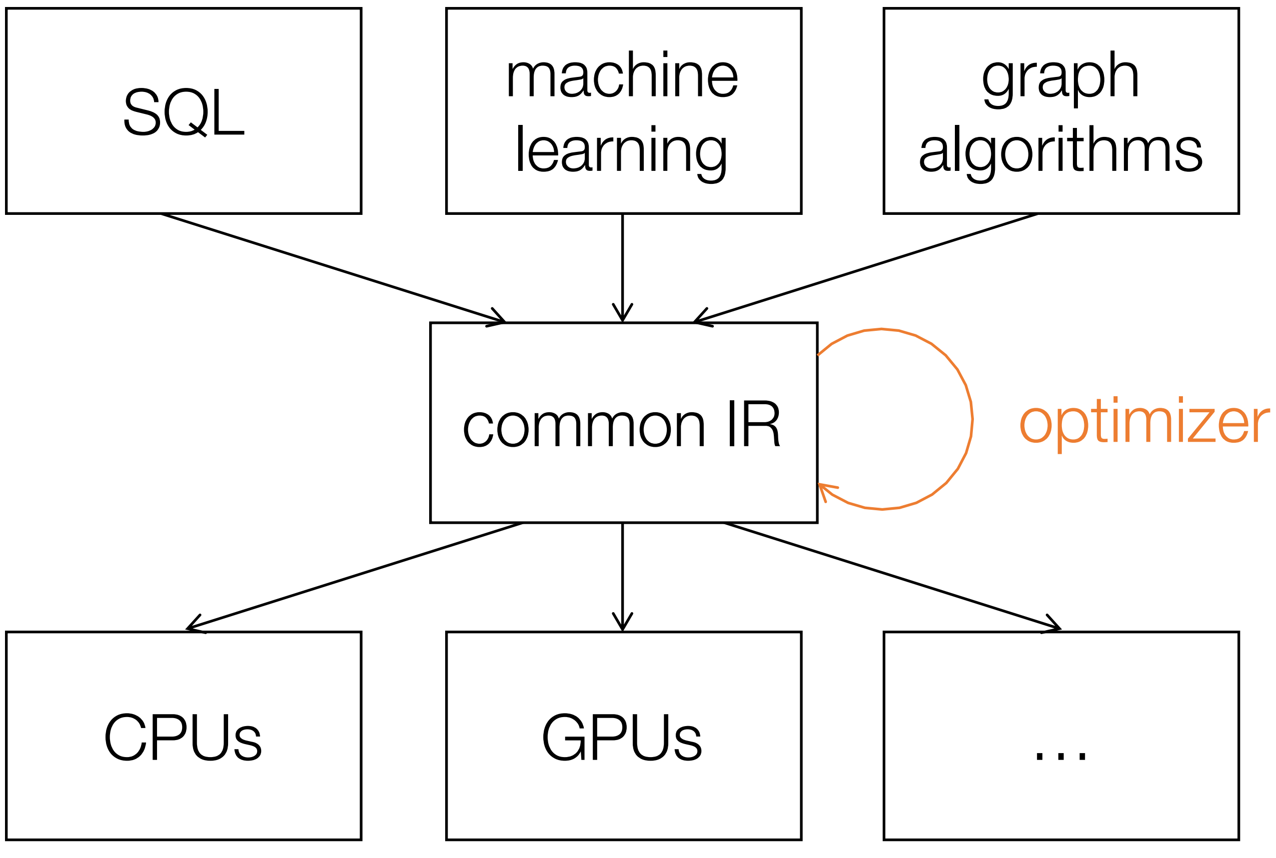
Observation

Many analytics algorithms can be written with a few “embarrassingly parallel” operators

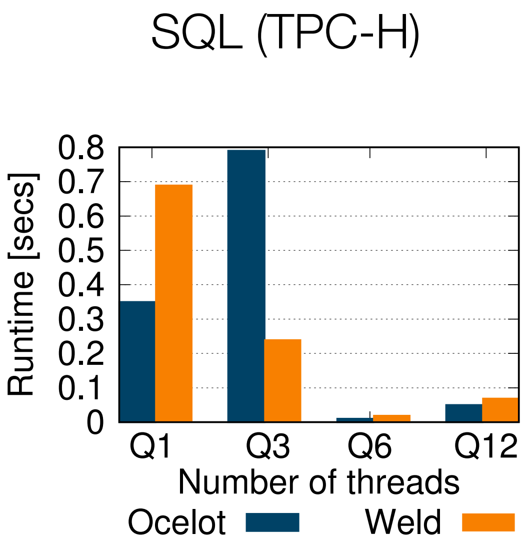
- » See how many run on MapReduce / Spark

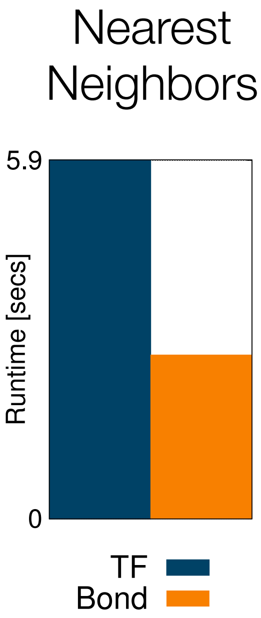
Focus on these instead of general programs

The Goal


SQL
machine learning
graph algorithms
common IR
optimizer
CPUs
GPUs
...

Results on GPUs





Example Transformations

```
def query(products: vec[{dept:int, price:int}]):
  sum = 0
  for p in products:
    if p.dept == 20: sum += p.price
```

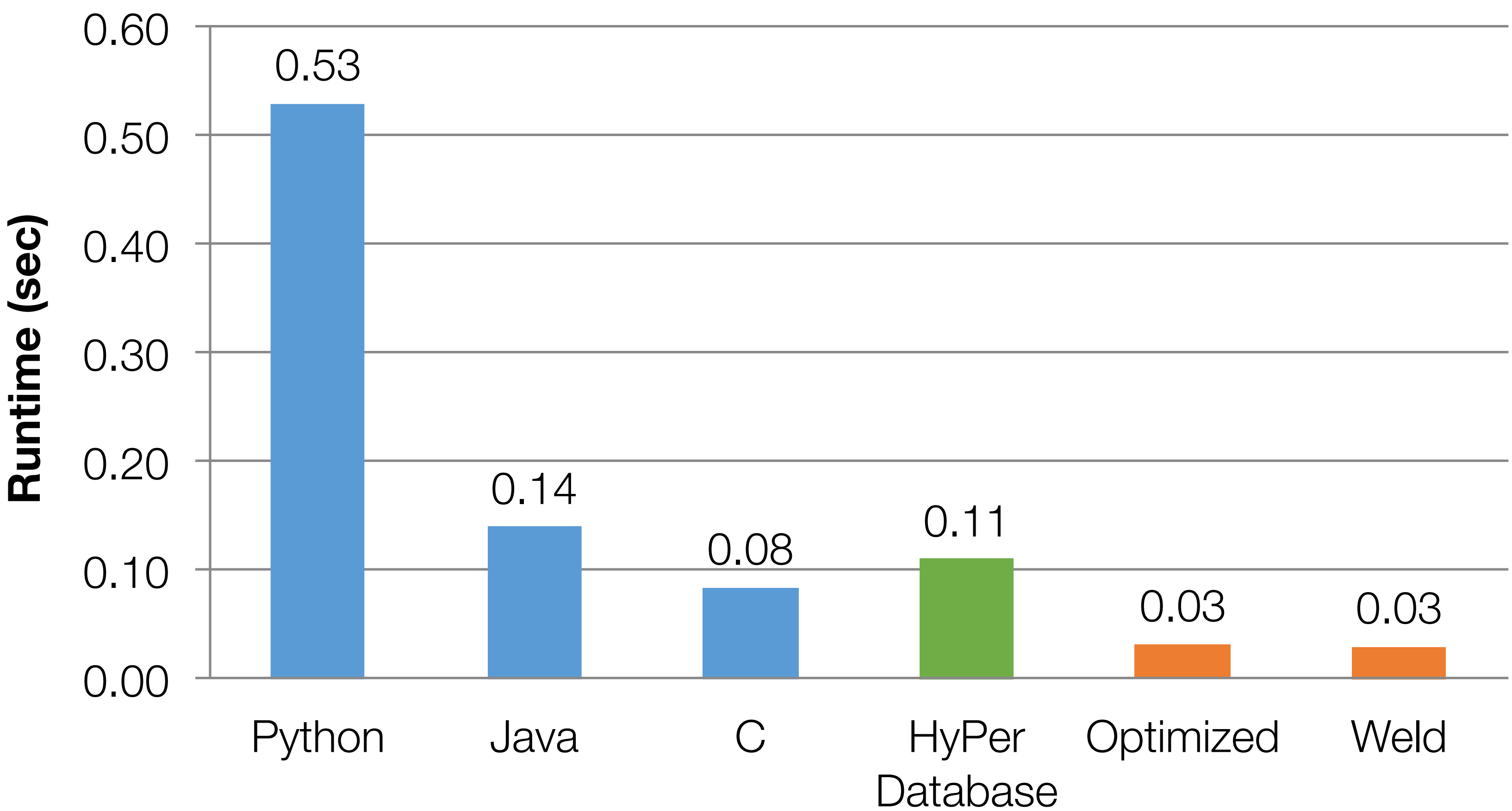
row-to-column

```
def query(dept: vec[int], price: vector[int]):
  sum = 0
  for i in 0..len(users):
    if dept[i] == 20: sum += price[i]
```

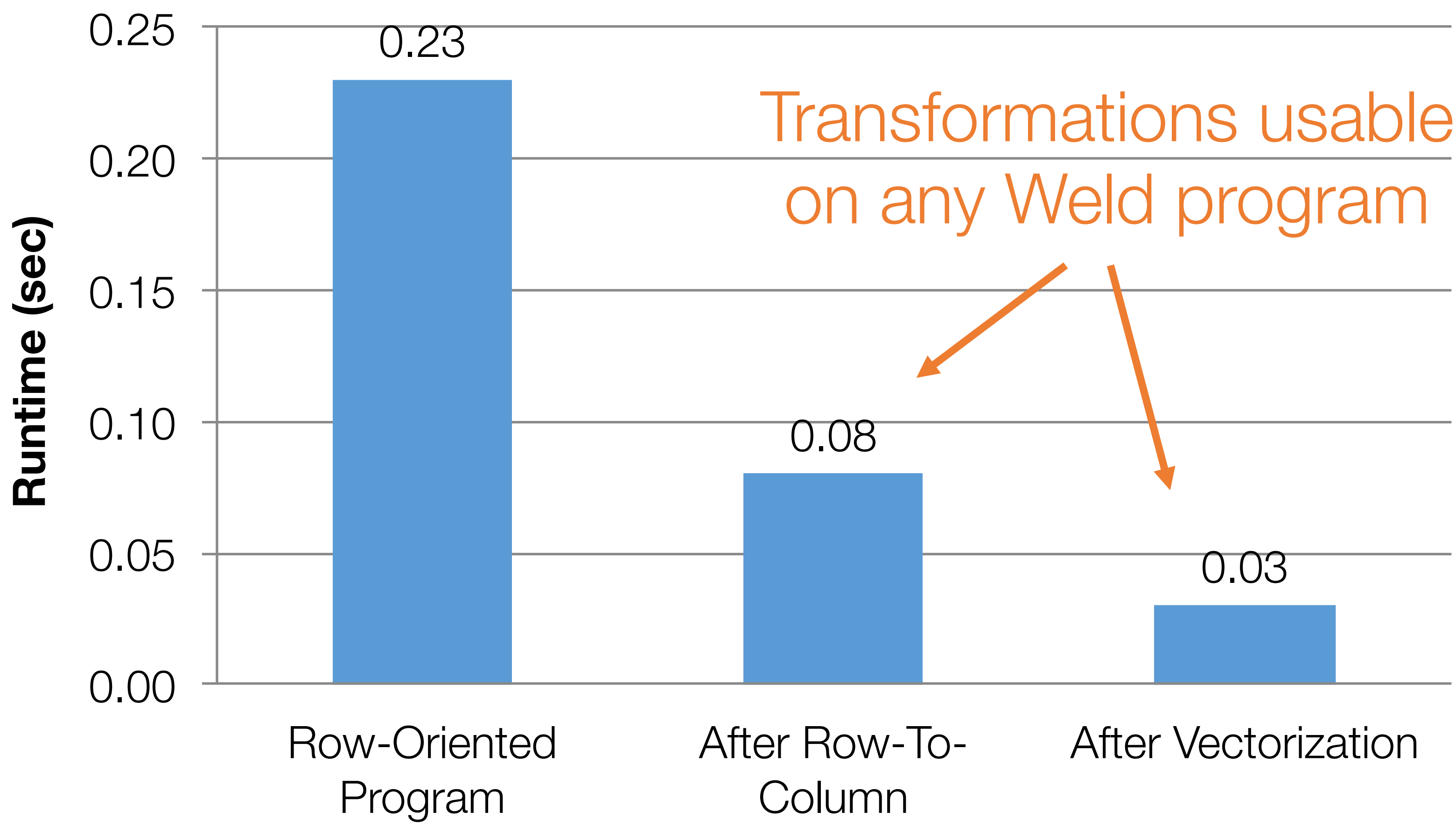
vectorization

```
  for i in 0..len(products) by 4:
    sum += price[i..i+4] * (dept[i..i+4] == [20,20,20,20])
```

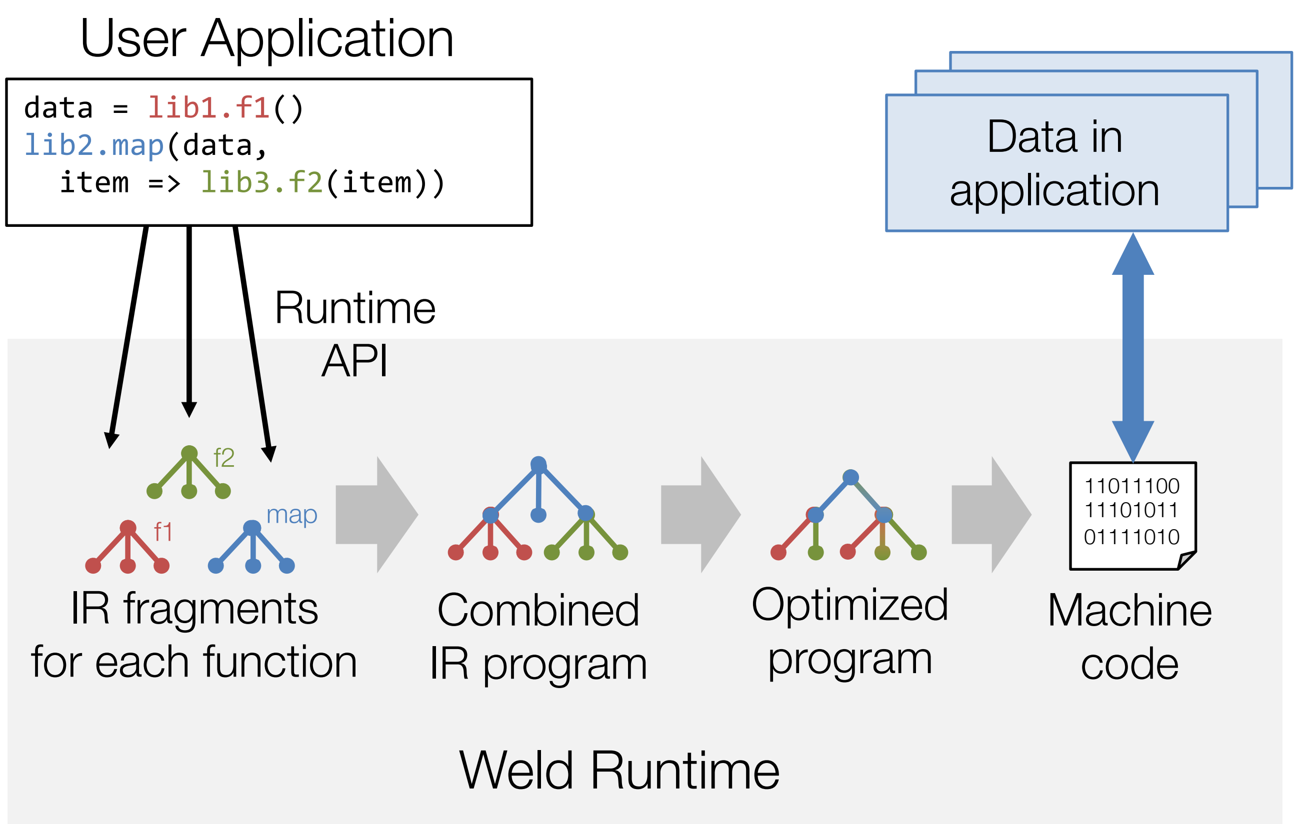
Weld Results: TPC-H Q6



Effect of Optimizations



How Weld Fits Into Applications



Example: Spark + NumPy

```
data = spark.sql("
  select user.features
  from users
  where age > 20")
```

Select some users using Spark SQL

```
scores = data.map(
  lambda vec: scoreMatrix * vec)
```

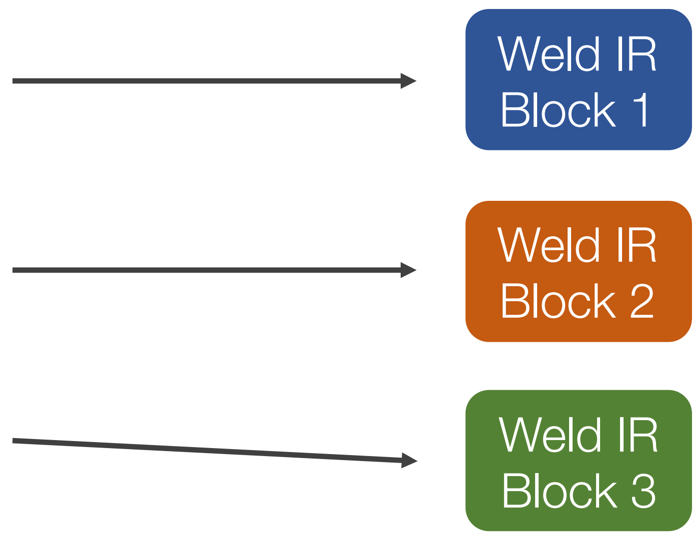
Score them using NumPy

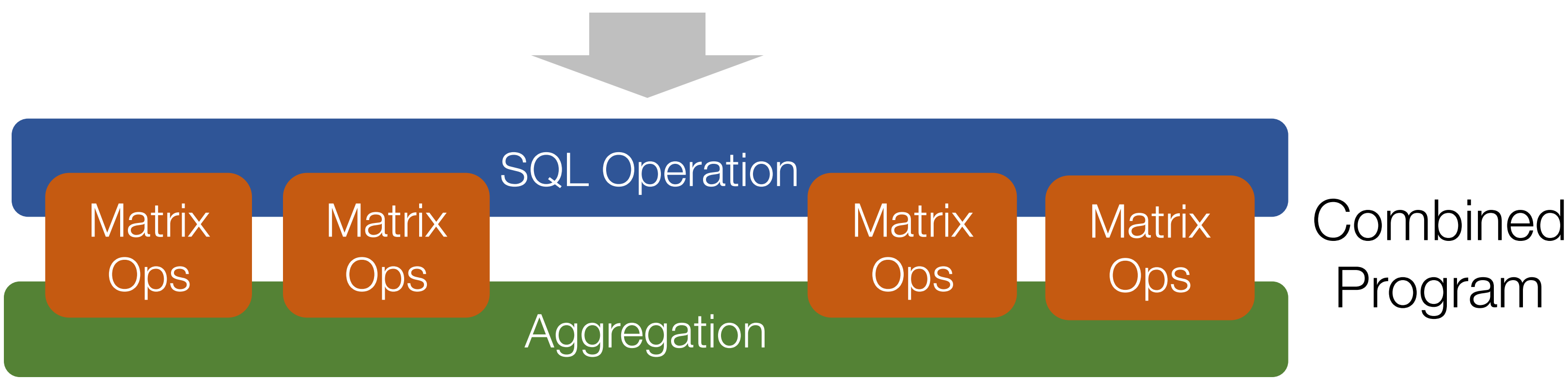
```
average = scores.mean()
```

Compute average using Spark

Example: Spark + NumPy

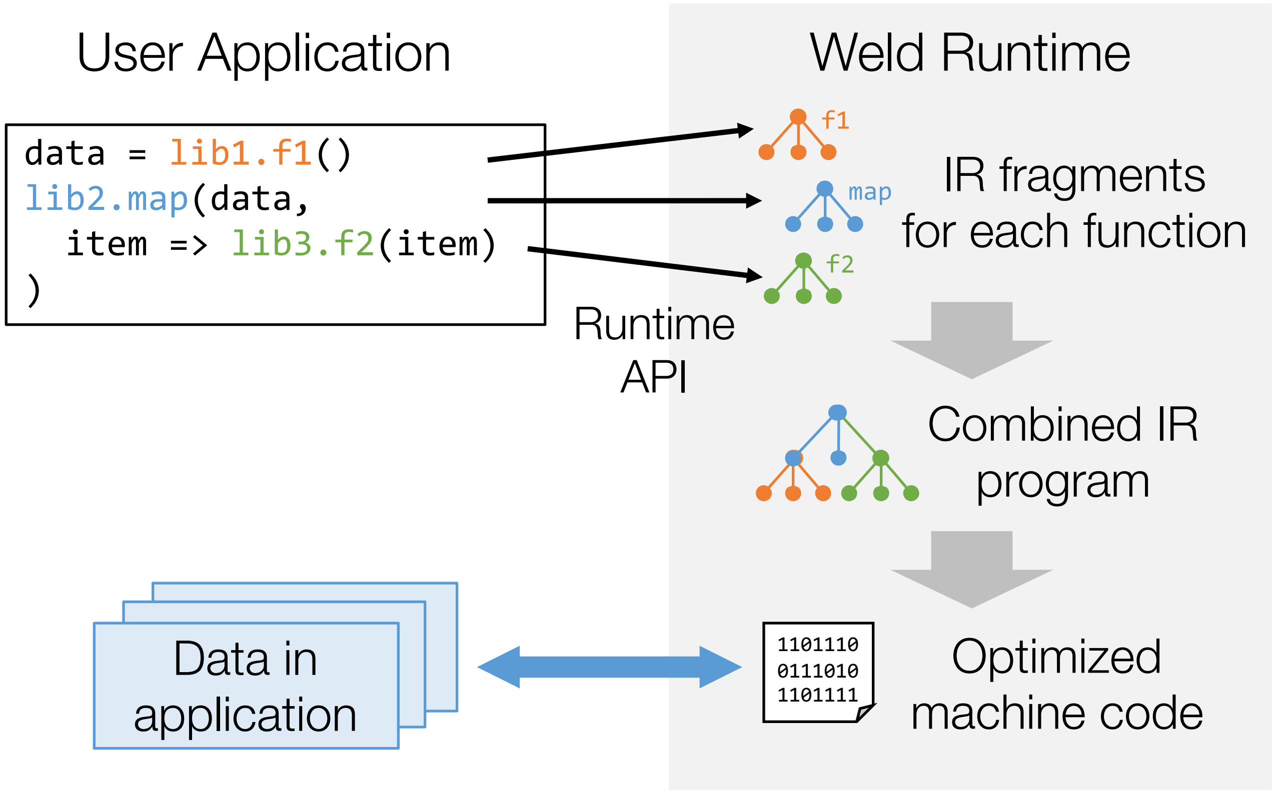
```
data = spark.sql("
  select user.features
  from users
  where age > 20")

scores = data.map(
  lambda vec: scoreMatrix * vec)

average = scores.mean()
```





Runtime API

Uses lazy evaluation to collect work across libraries



Supported Optimizations

Loop Fusion

Row-to-Column

Common Subexpressions

Inlining

Insert free() Calls

Loop Tiling

Constant Folding

Branch Predication

Vectorization

…