CS 744: WELD

Shivaram Venkataraman

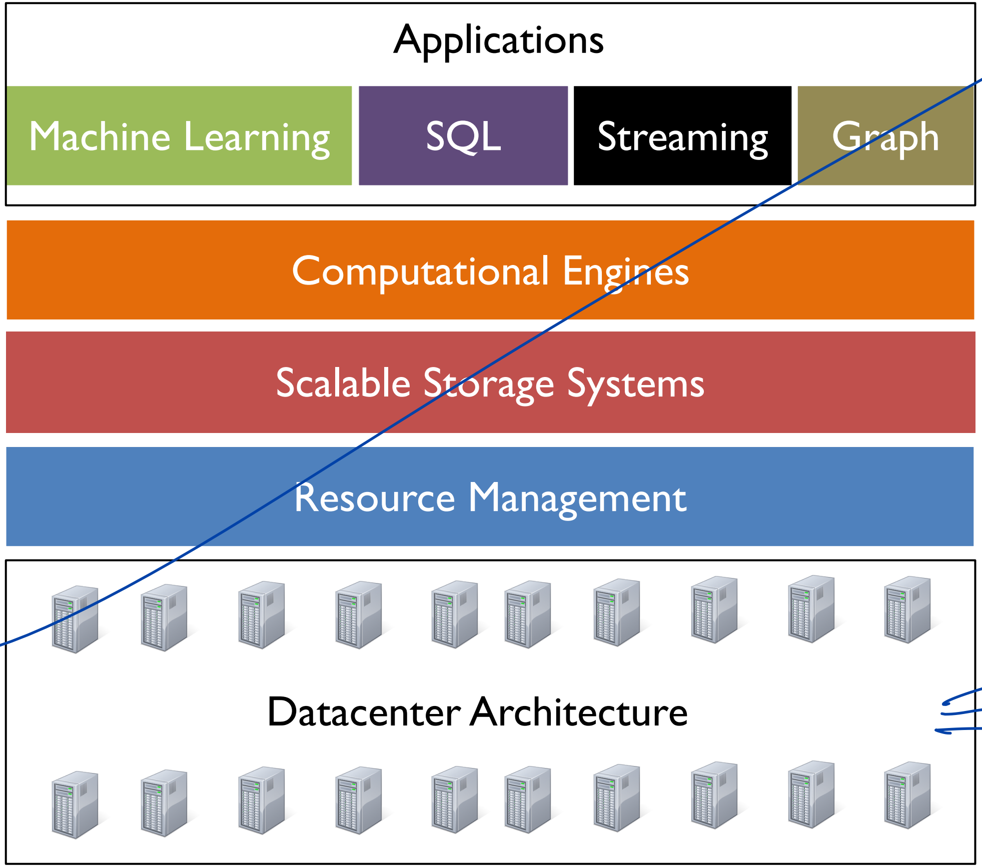
Fall 2019

ADMINISTRIVIA

Course Project: Check in meetings Thu, Mon

Preparation for the meeting

- what have you done so far
- a timeline for things you want to do next
- what are some specific things we can help you with



Applications

Machine Learning | SQL | Streaming | Graph

Computational Engines

Scalable Storage Systems

Resource Management

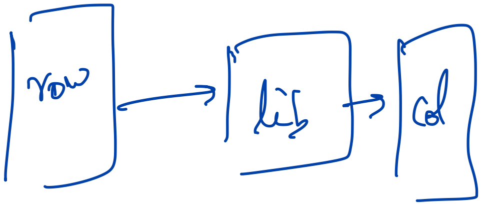
Datacenter Architecture

Emerging trends in Big Data

Computational resources

New hardware

SETTING



Data movement - is expensive

Multiple frameworks
↳ Address Spaces

- Process

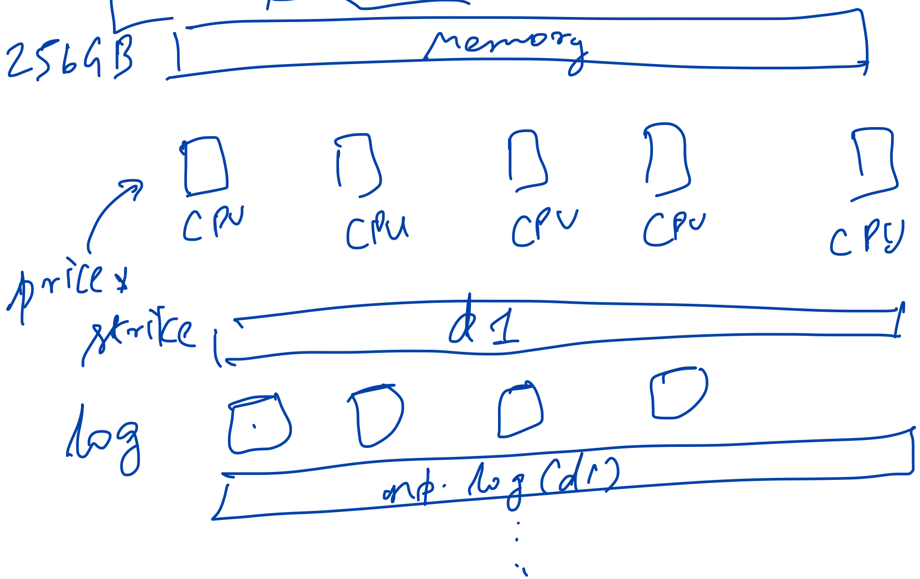
- Multi-core machines
- Multiple functions and libraries
- Data movement vs. compute

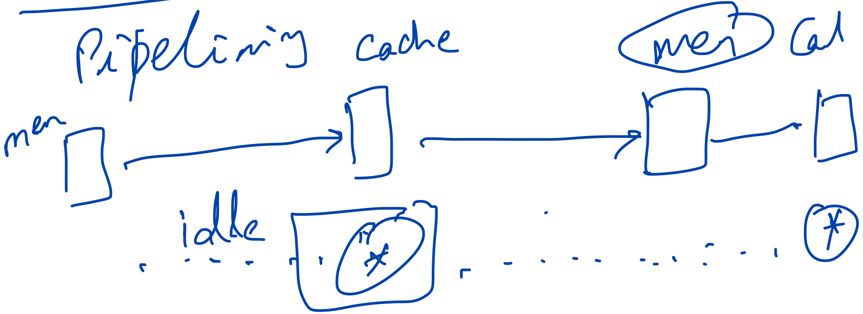
  CPU → Mem → CPU . . . . . .
- Alternate approaches?

```
// From Black Scholes
// all inputs are vectors
d1 = price * strike
d1 = np.log2(d1) + strike
```

multiply

log





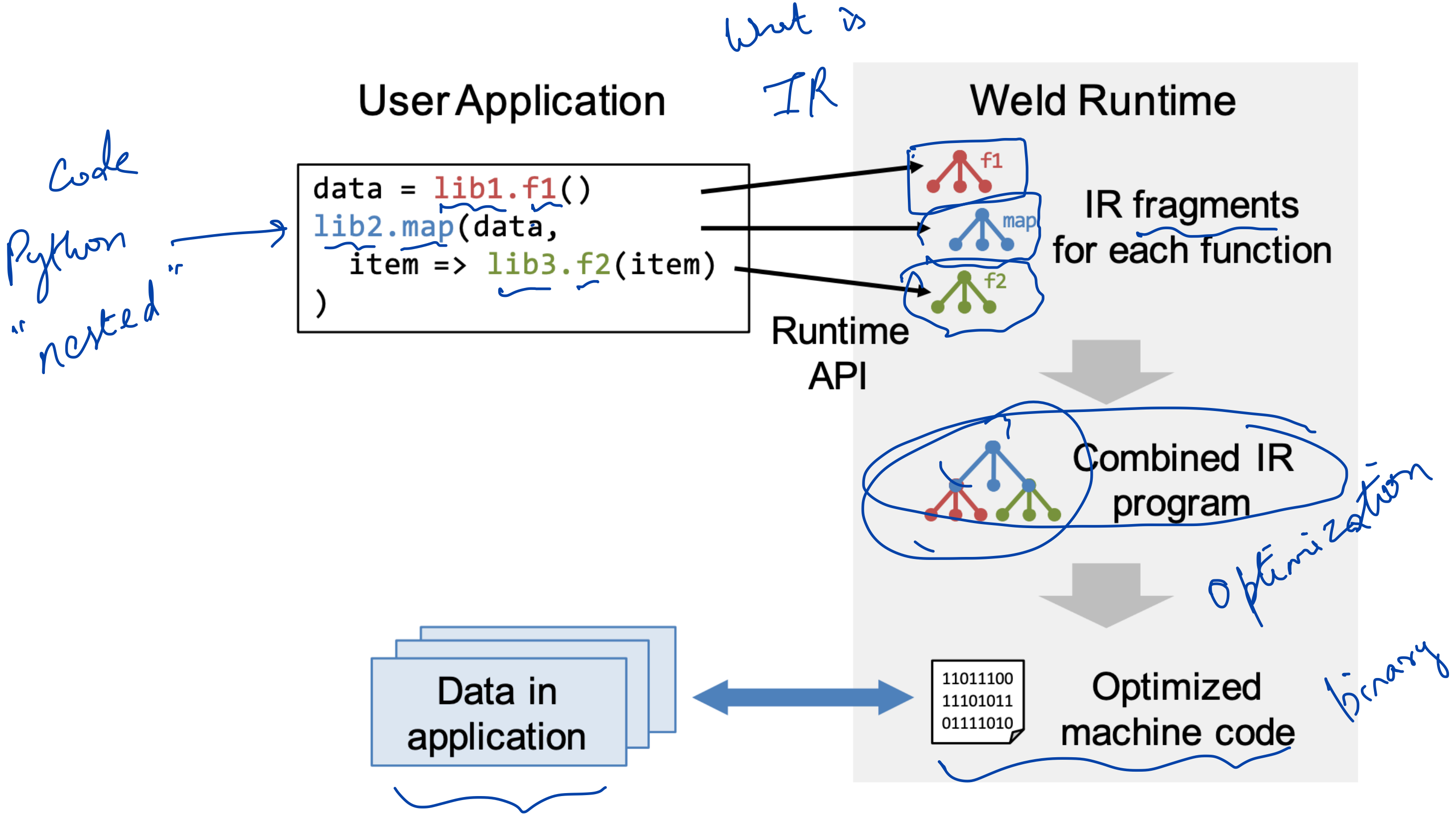
GOALS

Work with independently written libraries

Enable the most impactful cross-library optimizations

Integrate incrementally into existing systems

SYSTEM OVERVIEW



User Application

```
data = lib1.f1()
lib2.map(data,
  item => lib3.f2(item)
)
```

Weld Runtime

Runtime API

IR fragments for each function

Combined IR program

Optimized machine code

Data in application

WELD IR

Code → Binary

Code ← IR → Binary

x86

↳ Intermediate Representation

Data types

Scalars, structs, vectors, dictionaries

↳ Python

Parallel loops and builders

merge(builder, value)

↳ for(vector, builders, func)

result(builder)

↓

end

Explicit parallelism

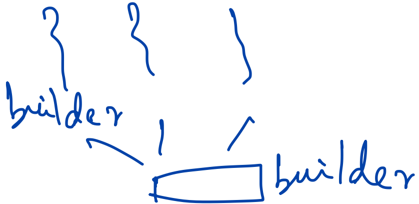
for (i = 1 to 10)

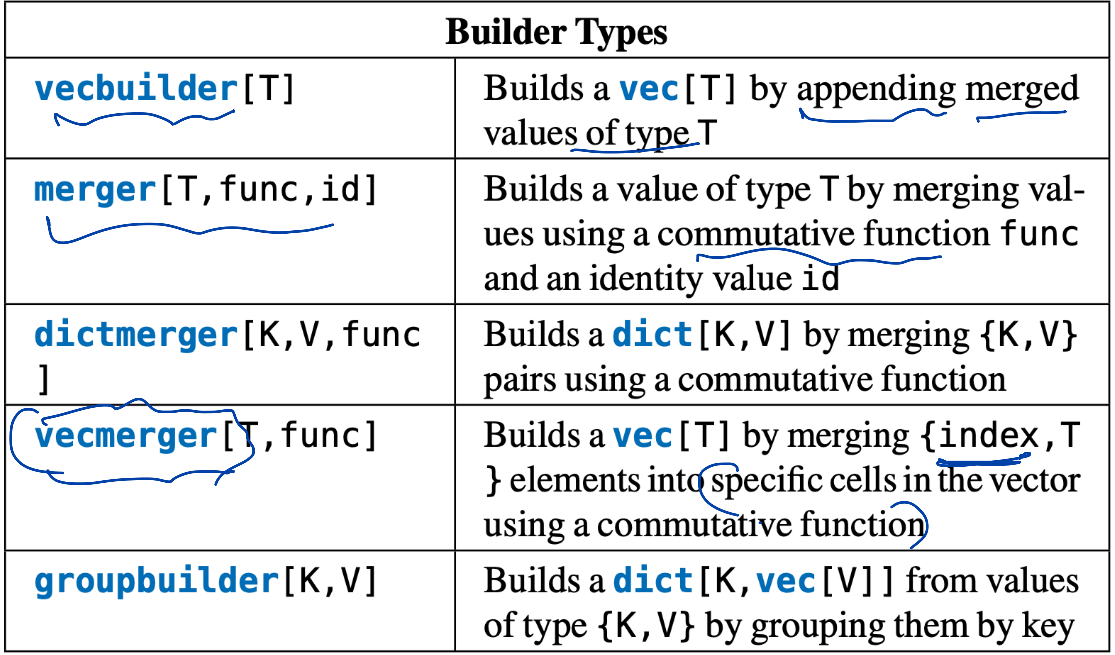
< : > sum += i

Merge results

builder

BUILDER TYPES





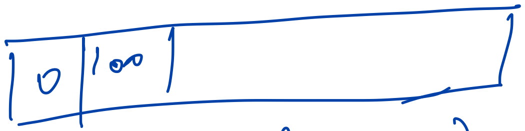
Builder Types	
vecbuilder[T]	Builds a **vec**[T] by appending merged values of type T
merger[T,func,id]	Builds a value of type T by merging values using a commutative function func and an identity value id
dictmerger[K,V,func]	Builds a **dict**[K,V] by merging {K,V} pairs using a commutative function
vecmerger[T,func]	Builds a **vec**[T] by merging {index,T } elements into specific cells in the vector using a commutative function
groupbuilder[K,V]	Builds a **dict**[K,**vec**[V]] from values of type {K,V} by grouping them by key

Accumulators

↳ Spark

↳ Power Graph

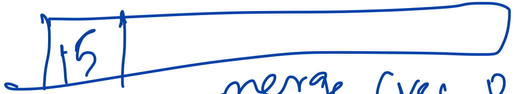
vector



merge(vec, 0)

merge(vec, 100)

vec merger(+)



merge(vec, 0, 5)

merge(vec, 0, 10)

EXAMPLES OF BUILDERS

```
b1 := vecbuilder[int];
b2 := merge(b1, 5);
b3 := merge(b2, 6);
result(b3)
```

immutable

IR

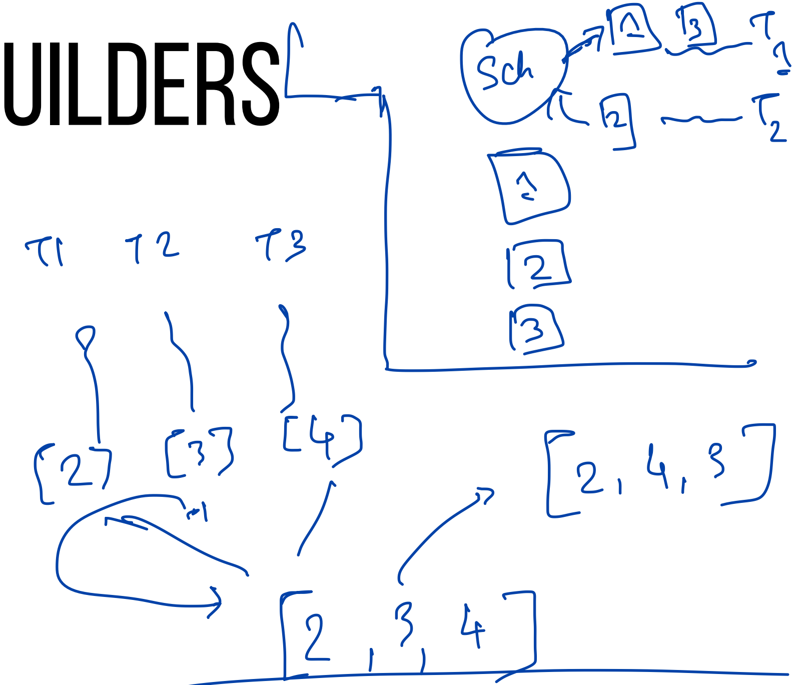
[5, 6]

```
b1 := vecbuilder[int];
b2 := for([1,2,3], b1, (b, x) => merge(b, x+1));
result(b2)
```

data

builders within loop

for runs on every element



Work stealing

T2 finishes fast

T1 still running

Cilk

MULTIPLE BUILDER

Code

```
data := [1,2,3];
r1 := map(data, x => x+1);
r2 := reduce(data, 0, (x, y) => x+y);
```

[2,3,4]

6

map

reduce

```
data := [1,2,3];
result(
    for(data, {vecbuilder[int], merger[+]},
        (bs, x) =>
            {merge(bs.0, x+1), merge(bs.1, x)}
))
```

Scan the data once and produce two results

RUNTIME API

numpy.log()

API to express IR fragments in libraries

IR fragment()

Capture dependencies across functions/libraries.

multiply( )

IR fragment()

Lazy Evaluation

IR (vector)

```
def square(self, arg):
    # Programatically construct an IR expression.
    expr = weld.Multiply(arg, arg)
    return NewWeldObject([arg], expr)
```

for, builder

IR dep

print num

eval(IR)

RUNTIME API

```
def large_cities_population(data):
  # data is a Pandas DataFrame object.
  filtered = data[data["population"] > 500000]
  sum = numpy.sum(filtered)
  print sum
```

→ filter

```
# Dataframe col > f, Input Weld expr: v0: vec[int], c0: int
filter(v0, x => x > c0)
```

= for , builder

```
# Numpy.sum Input Weld expr: v0: vec[int]
reduce(v0, 0, (x, y) => x+y)
```

merger

= for, builder

RUNTIME API

Modified numpy
Pandas

```
reduce(
    filter(v0,
           (x) => x>500000),
    0,
    (x,y) => x+y)
```

Lazy

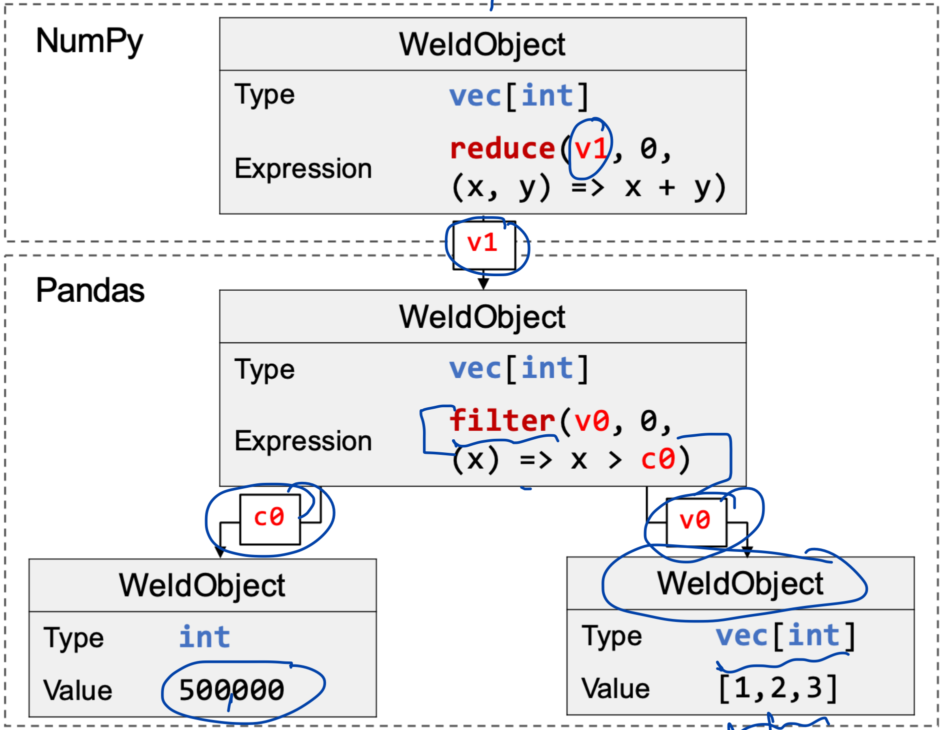
↓ optimizer

```
result(
    for(v0, merger[+,0],
      (b, x) =>
        if (x > 500000)
          merge(b, x)
        else
          b
))
```

Fused

merge when cond is true

print (sum)



NumPy

WeldObject

Type vec[int]

Expression reduce(v1, 0, (x, y) => x + y)

v1

Pandas

WeldObject

Type vec[int]

Expression filter(v0, 0, (x) => x > c0)

c0

v0

WeldObject

Type int

Value 500000

WeldObject

Type vec[int]

Value [1,2,3]

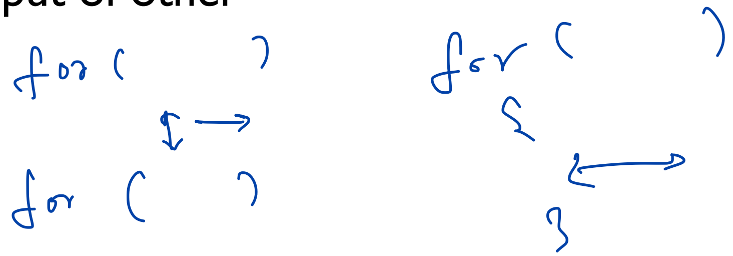
encoder

OPTIMIZATIONS → Extensively studied

Loop Fusion

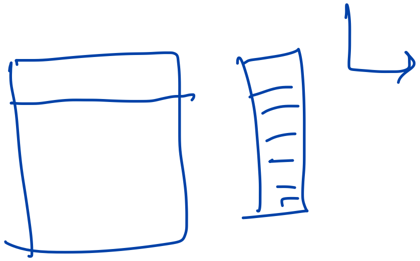
Fuse adjacent loops when output of one loop is input of other

Fuse multiple passes over the same vector

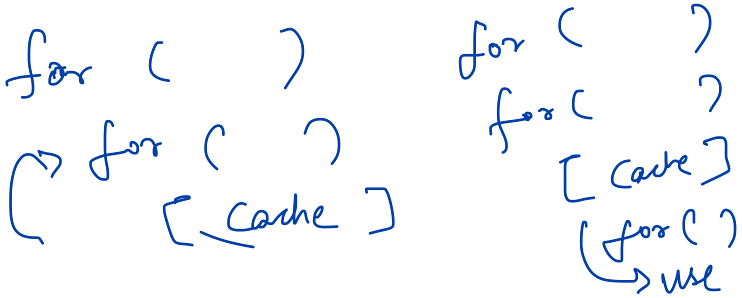


Loop Tiling

Break nested loops into blocks



Linear Algebra



OPTIMIZATIONS

Vectorization

Transform loops to use vector instructions

AVX instructions

for(i= 1 to 10, i+=1)
inst

for (i = 1 to 10, i+=4)
AVX Instr
< 4 >

Common subexpression elimination

Transforms to not run the same computation multiple times

$a = (b * c) + g$

$e = b * c * f$

$tmp = b * c$

$a = tmp + g$

$e = tmp * f$

Multiple passes

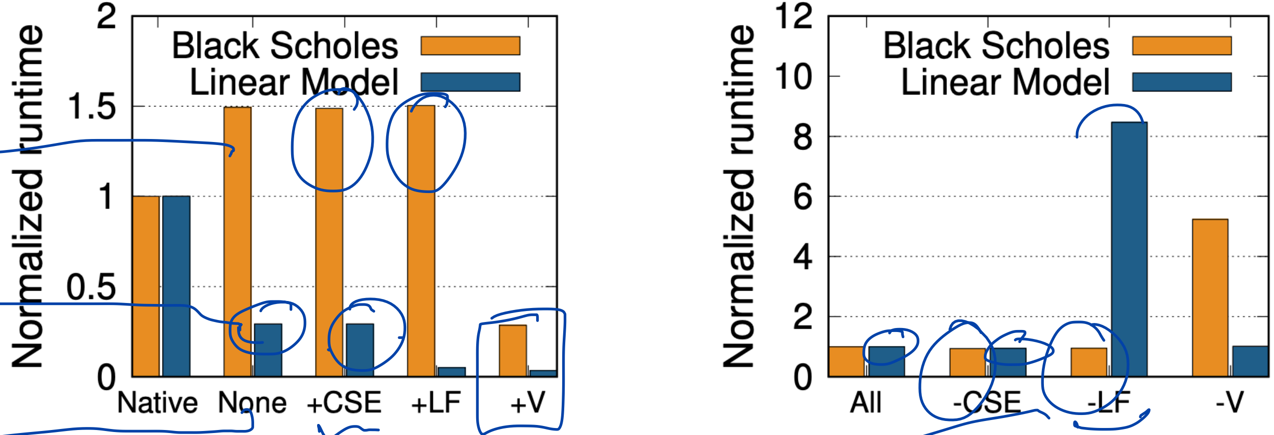
DISCUSSION

https://forms.gle/DxHfcmuS2juK1tuE7

Sync

IR overhead?

Why?



linear model workload

None?

(a) Adding Optimizations

(b) Removing Optimizations

All optimizations might not help

Parallelism?

Vectorization = Hardware Computation

Linear model is greatly affected by removing LF.

Data movement

What are some possible limitations of Weld as described in the paper?

↳ Scale to more libraries!

↳ Each library has to be modified to get perf win

Backend for each arch

→ Debug

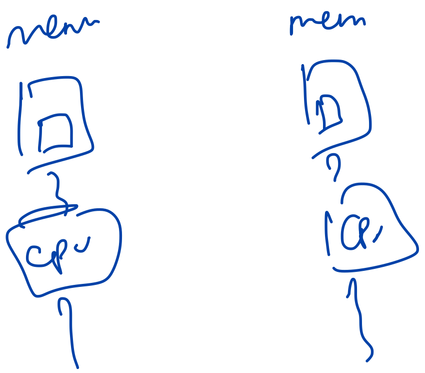
↳ Full deterministic

↳ Async SGD

↳ Placement NUMA

↳ Fault tolerance } Restricted to Single machine

Data doesnt fit in memory?



What does the Weld paper tell us about the using scale-up vs. scale-out?

NEXT STEPS

Next class: PyWren

Project check-in meetings