Caerus: NIMBLE Task Scheduling for Serverless Analytics

Hong Zhang, Yupeng Tang, Anurag Khandelwal, Jingrong Chen, Ion Stoica

riselab UC Berkeley

Yale University

Serverless computing

AWS Lambda Google Cloud Functions Azure Functions IBM Cloud Functions

Fast Scaling

30 ~ 120 Seconds → < 1 Second

Fine-grained billing

Per second → Per millisecond

Serverless analytics

ExCamera (NSDI'17) Sprocket(SoCC'18)
Video processing

gg compiler (ATC'19)
Distributed compiling

Locus (NSDI'19)
Mapreduce & SQL-like queries

Web severing

IoT applications

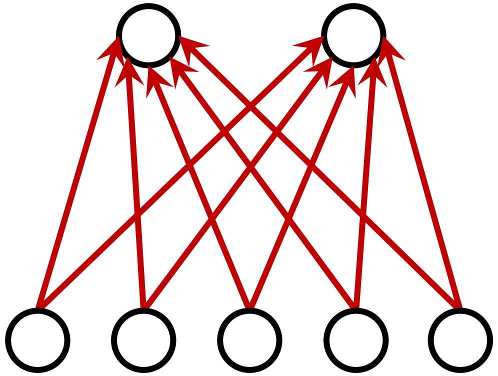
PyWren (SoCC'17)
Scaling python functions

NumPyWren
Matrix computation

Cloudburst (VLDB'20)
Stateful functions

Starling (Sigmod'20)
SQL query engine

2014 **Single-stage functions** 2017 **General data analytics** 2021

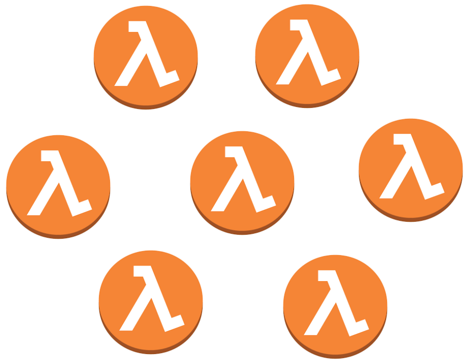


Job Execution plan

Job completion time (JCT)

Deploy

Cost (total duration of all workers)



A pool of (infinite) serverless workers

Serverless scheduling: a new problem

Server-centric users

Inter-job scheduling
Optimization Metrics: average JCT, cluster utilization, fairness *across jobs*

Now handled by the serverless platform

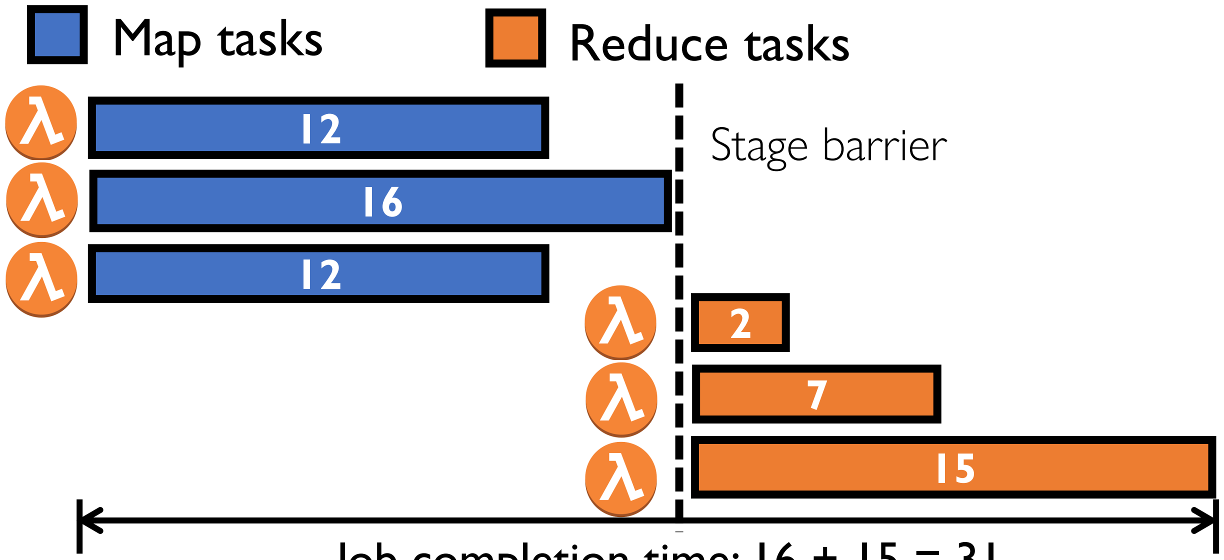
Serverless users

Intra-job scheduling across tasks
Optimization Metrics: *Both JCT and cost* for *each individual job*

Can existing server-centric intra-job scheduling policies optimize both JCT and cost in serverless settings?

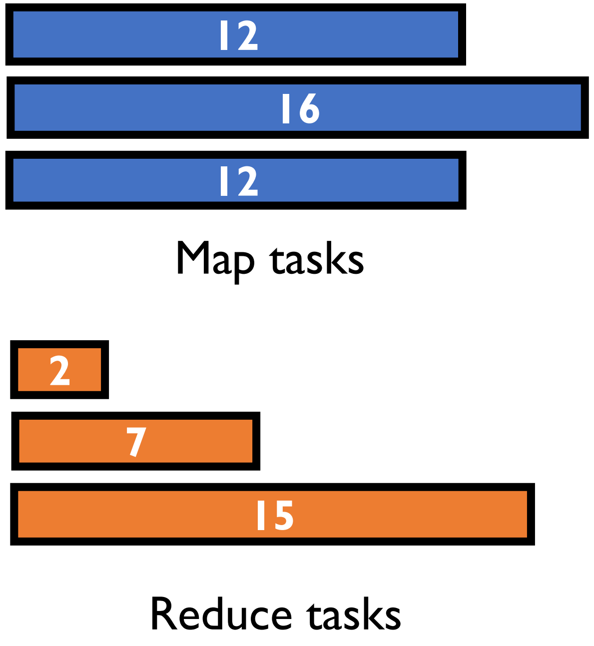
Trade-off: Lazy *vs.* Eager

Lazy: start a task after ALL tasks in its upstream stages have finished (Spark)



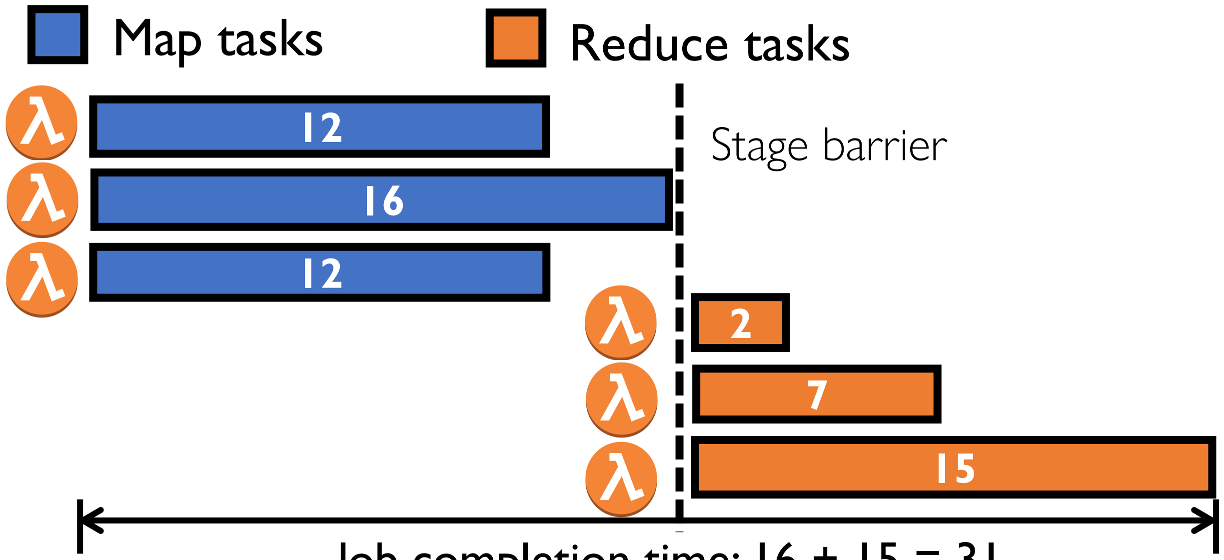
Job completion time: 16 + 15 = 31
Cost (total duration): (12*2+16)+(2+7+15) = 64

A MapReduce job with 3 map tasks and 3 reduce tasks



Trade-off: Lazy *vs.* Eager

Lazy: start a task after ALL tasks in its upstream stages have finished (Spark)

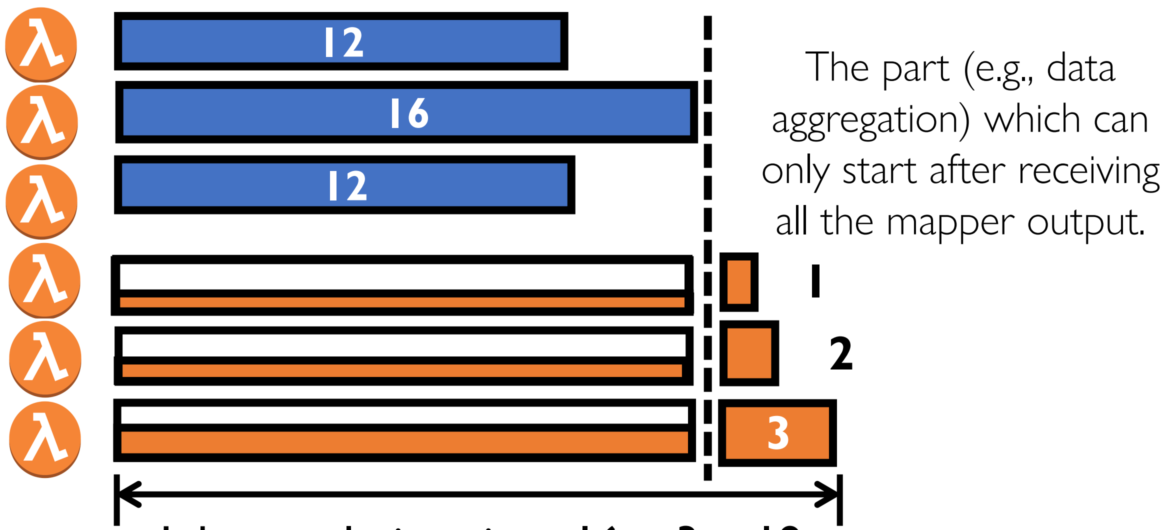


Job completion time: 16 + 15 = 31
Cost (total duration): (12*2+16)+(2+7+15) = 64

Minimizes cost (duration)
Much longer job completion time (1.63X) ☹

Trade-off

Eager: start a task when ANY output from its upstream stages is ready (Mapreduce Online)

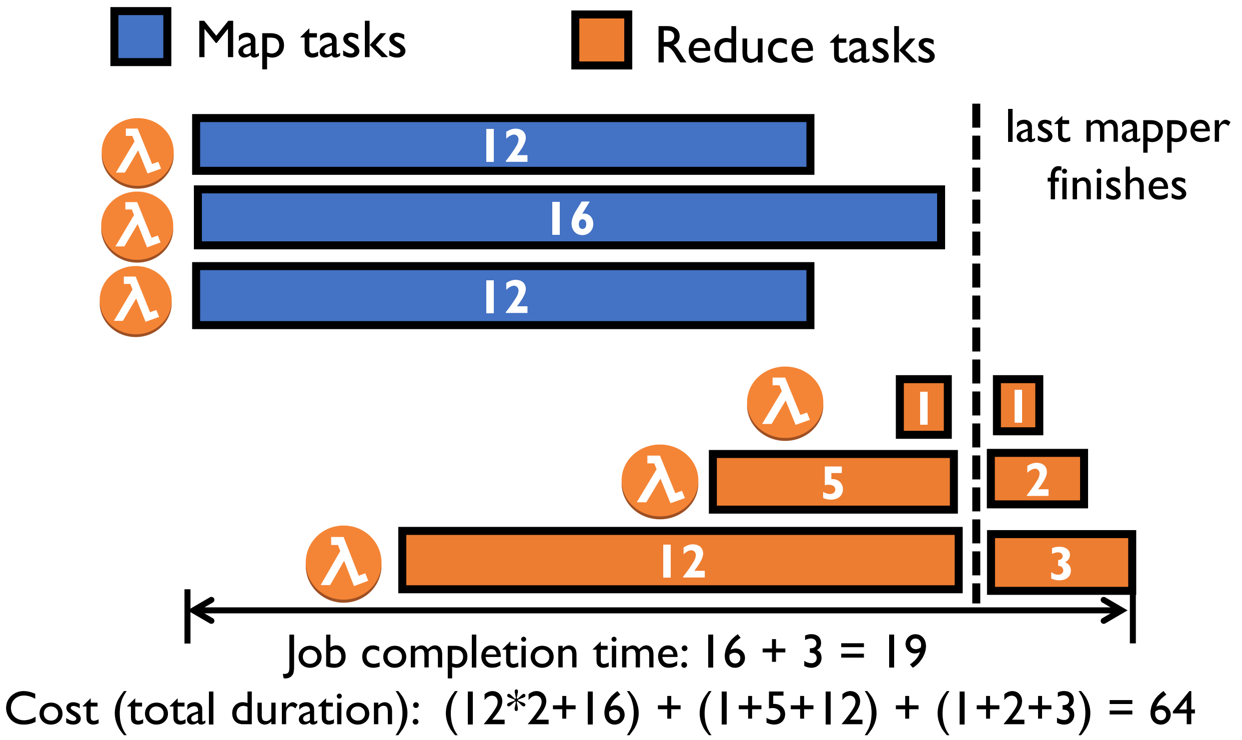


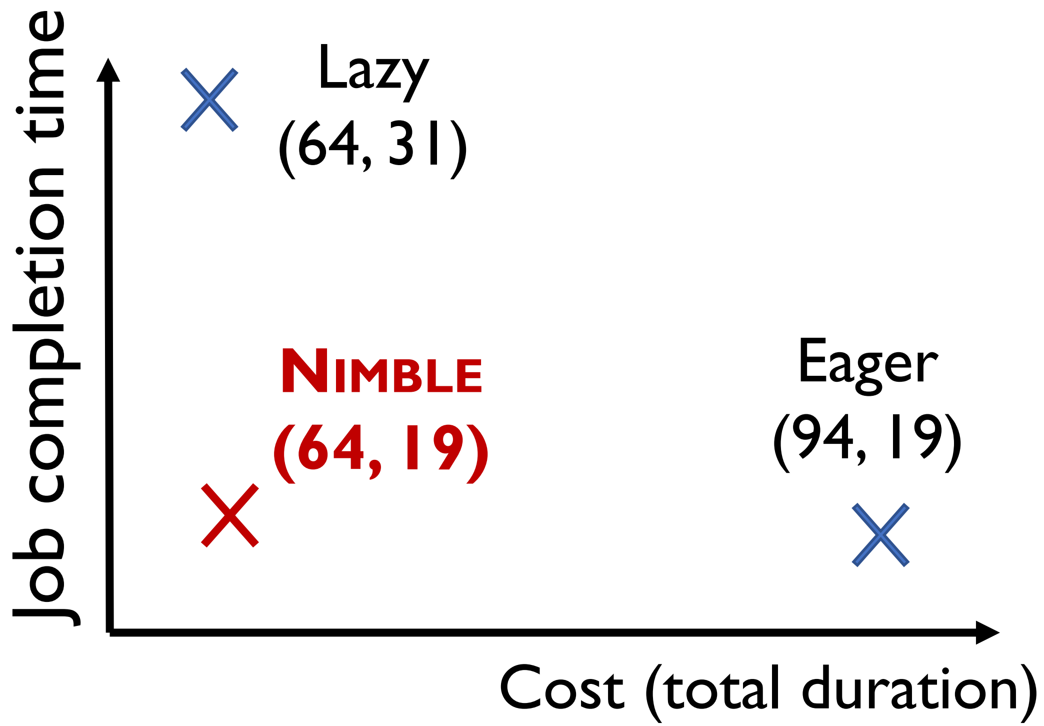
Job completion time: 16 + 3 = 19
Cost: (12*2+16)+ 16*3 + (1+2+3) = 94

Minimizes job completion time
Much higher cost (1.47X) ☹

NIMBLE scheduling: main idea

Main idea:
- Fully exploit the *flexible resource scaling* of serverless computing
- Calculate and enforce the *best launch time* for *each individual task*

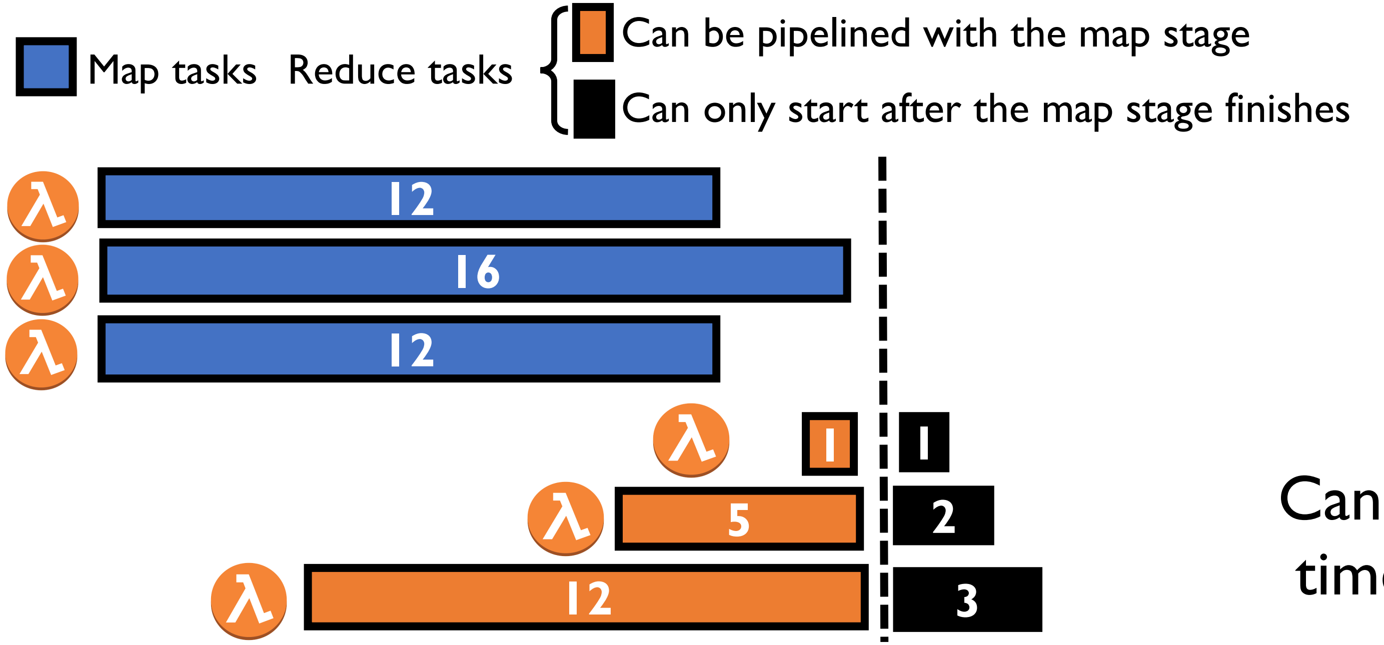


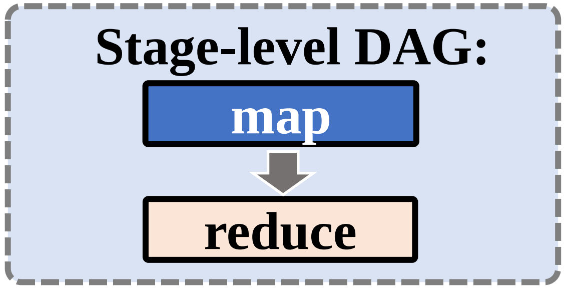


How to calculate the optimal launch time for each task?

Challenge 1: Describe pipelinablity

- NIMBLE scheduling requires a precise description of the pipelinablity across different job stages



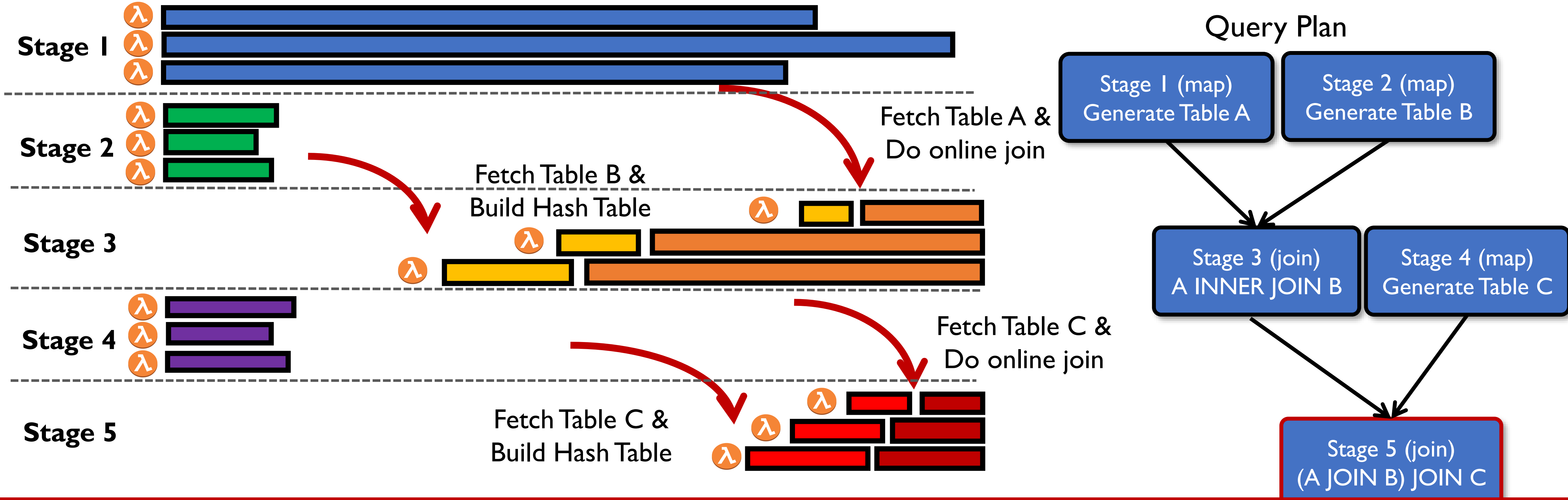


Cannot calculate the optimal task launch time without *sub-stage level* information

How to describe pipelinability at *sub-stage level*?

Challenge 2: Arbitrary DAGs

- General analytics workloads can have complicated DAGs.
  - **Within a stage:** tasks can consume data from multiple upstream stages
  - **Across stages:** tasks can have cascading dependencies



How to calculate the optimal task launch time for arbitrary DAGs?

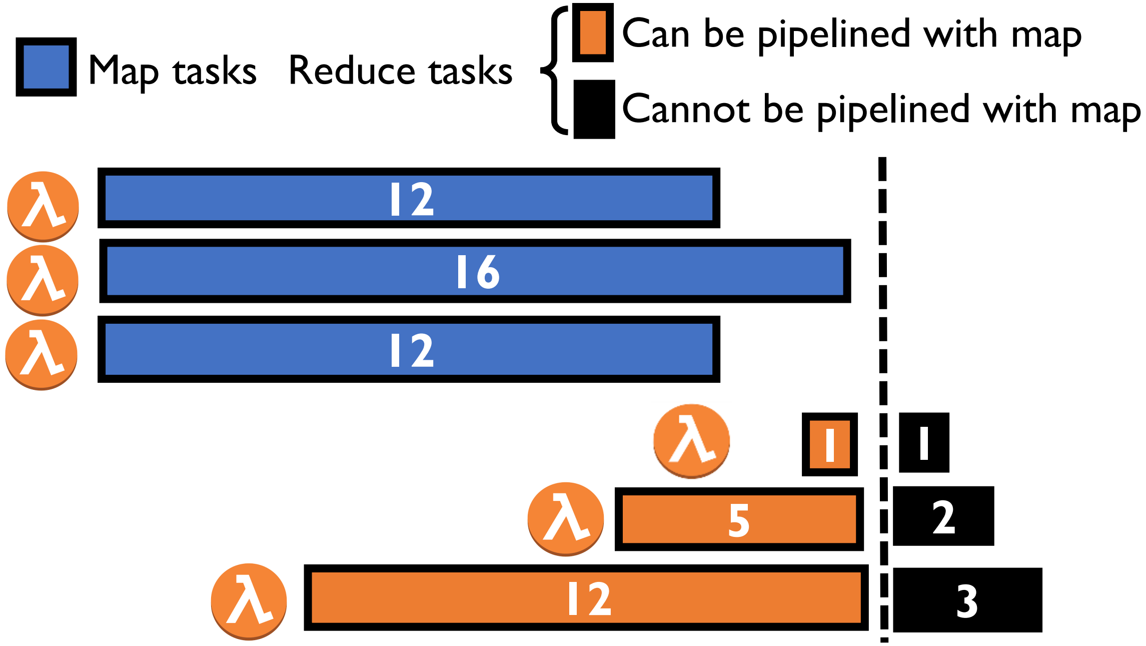
NIMBLE design outline

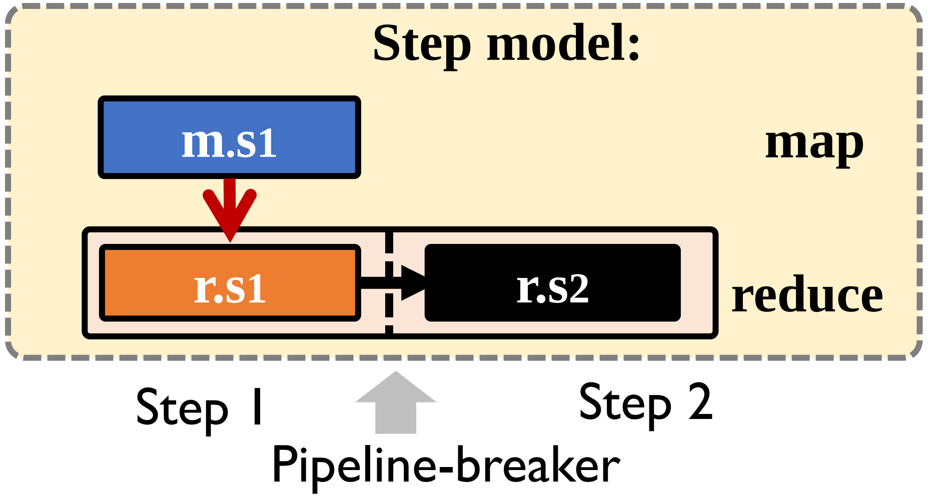
- **Challenge 1:** How to describe pipelinability at sub-stage level?
- Develop a ***step model*** to precisely capture the sub-stage level pipelinablity

- **Challenge 2:** How to calculate the optimal task launch time for arbitrary DAGs?
- Develop a scheduling algorithm which guarantees ***optimal cost*** while being ***Pareto-optimal*** between cost and JCT for arbitrary DAGs

Step model

- **Idea:** Break stages into steps
  - Step: largest pipeline-able component within a stage
  - Separated by pipeline breakers[1] (e.g., MIN, MAX, SUM)

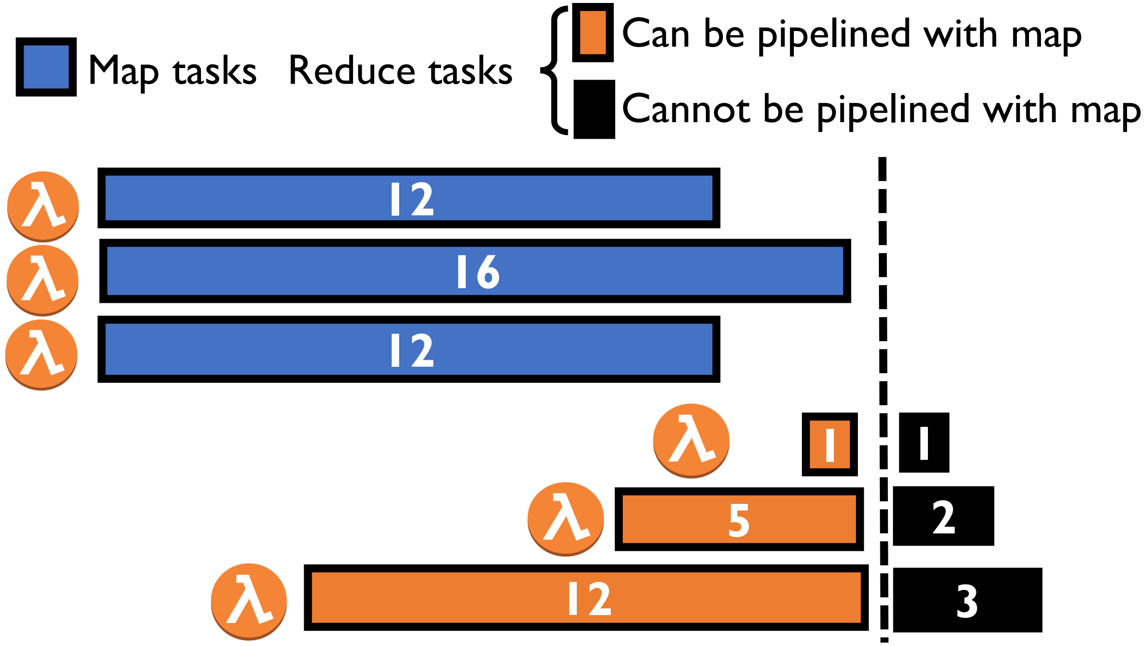


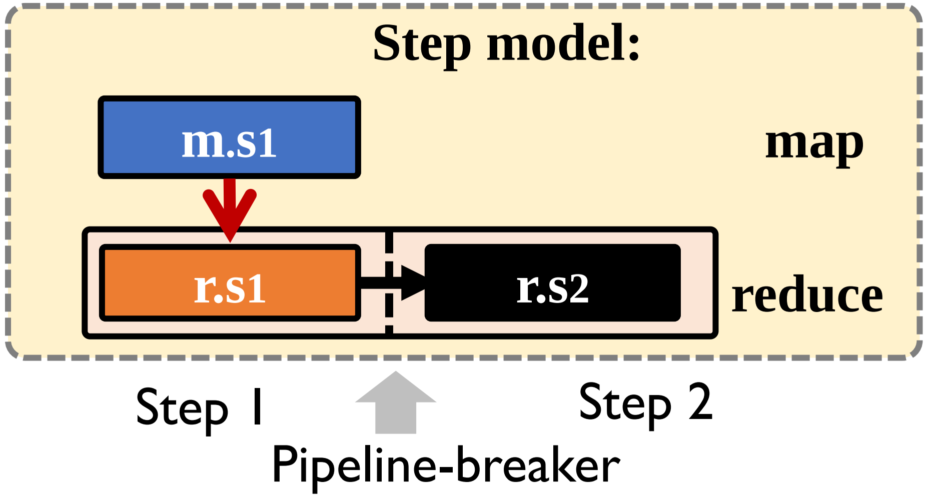


1. A pipeline breaker is an operator that produces the first output only after all its input has been processed

Step model

- **Idea:** Break stages into steps
  - Step: largest pipeline-able component within a stage
  - Separated by pipeline breakers[1] (e.g., MIN, MAX, SUM)





Data dependency within a stage:
- Must be executed *sequentially*

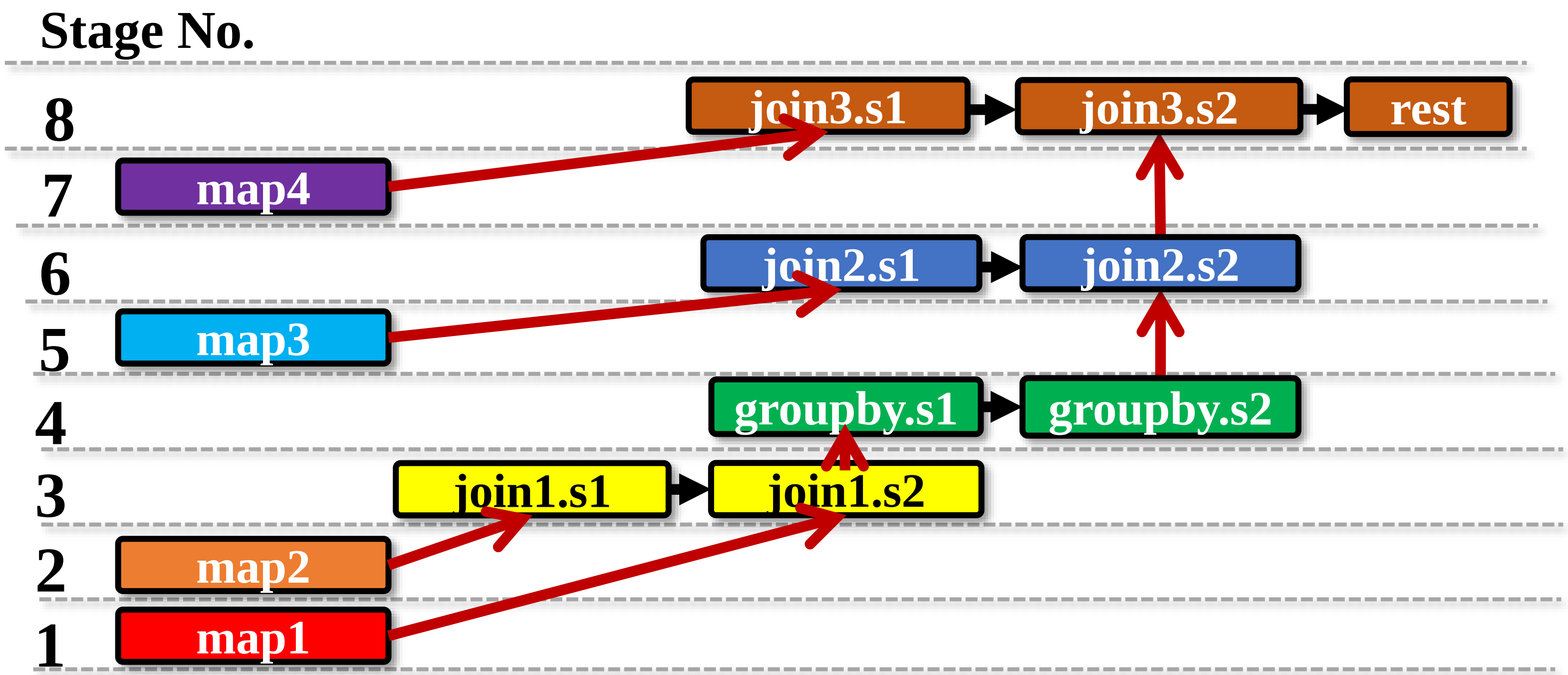
Data dependency across stages:
- Can be *pipelined*
- Referred as *parent-child step pairs*

1. A pipeline breaker is an operator that produces the first output only after all its input has been processed

Step model

- Example: the step model for a complicated SQL query in TPC-DS benchmark

```
WITH customer_total_return AS (
    SELECT
        sr_customer_sk AS ctr_customer_sk,
        sr_store_sk AS ctr_store_sk,
        sum(sr_return_amt) AS ctr_total_return
    FROM
        store_returns,
        date_dim
    WHERE
        sr_returned_date_sk = d_date_sk
        AND d_year = 2000
    GROUP BY
        sr_customer_sk,
        sr_store_sk
)
SELECT
    c_customer_id
FROM
    customer_total_return ctr1,
    store,
    customer
WHERE
    ctr1.ctr_total_return > (
        SELECT
            avg(ctr_total_return) * 1.2
        FROM
            customer_total_return ctr2
        WHERE
            ctr1.ctr_store_sk = ctr2.ctr_store_sk)
    AND s_store_sk = ctr1.ctr_store_sk
    AND s_state = 'TN'
    AND ctr1.ctr_customer_sk = c_customer_sk
ORDER BY
    c_customer_id
LIMIT 100;
```

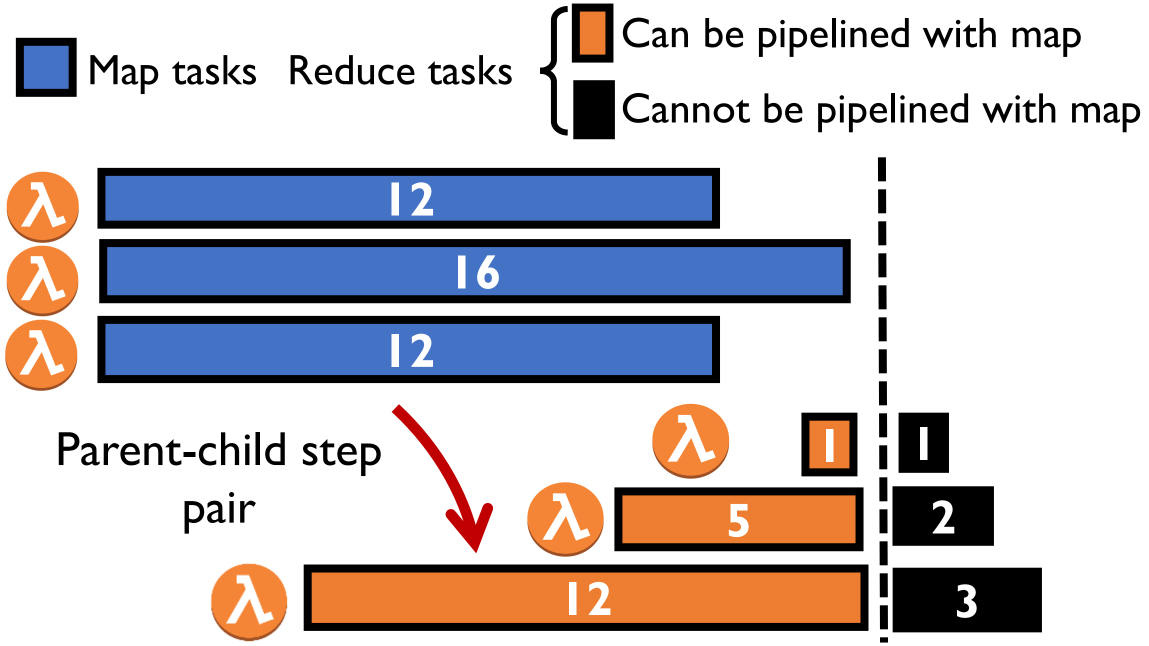


Step model can efficiently describe pipelinability across a wide range of applications

Basic algorithm for 2-stage map-reduce

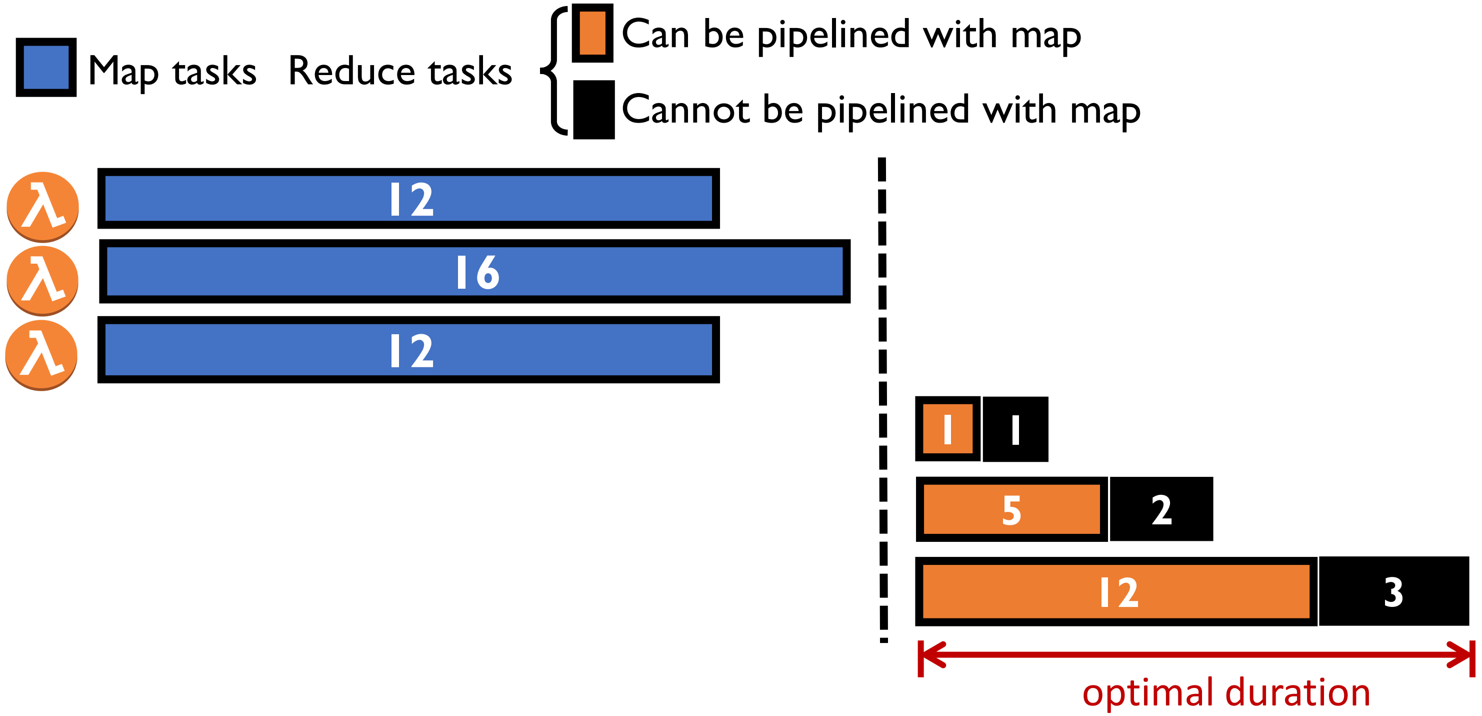
- Intuition to calculate the launch time:
  - Optimally overlap the *parent-child step pair* based on the *data produce and data consume rate*

Historical
+ online job information



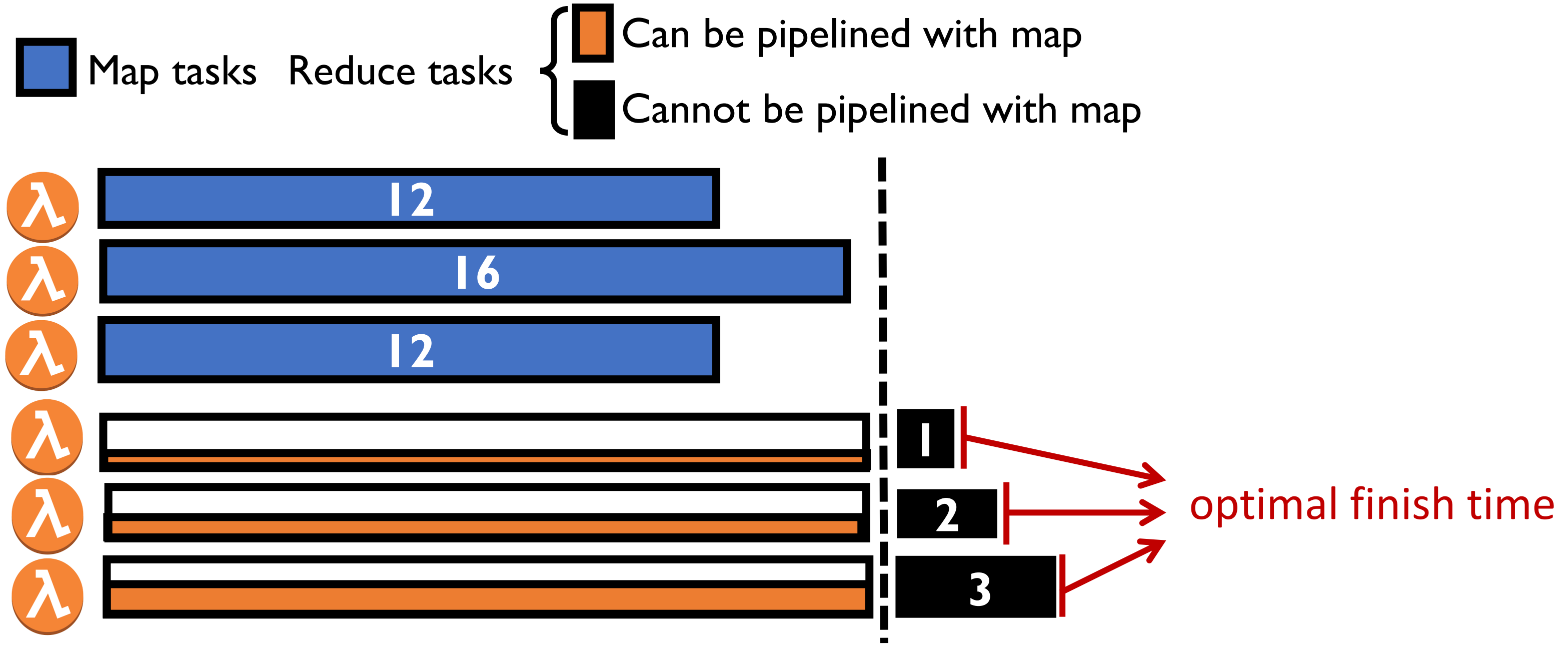
Basic algorithm for 2-stage map-reduce

- **Optimal launch time in three simple steps**
  - Step 1: Calculate *optimal task duration* based on *Lazy* solution



Basic algorithm for 2-stage map-reduce

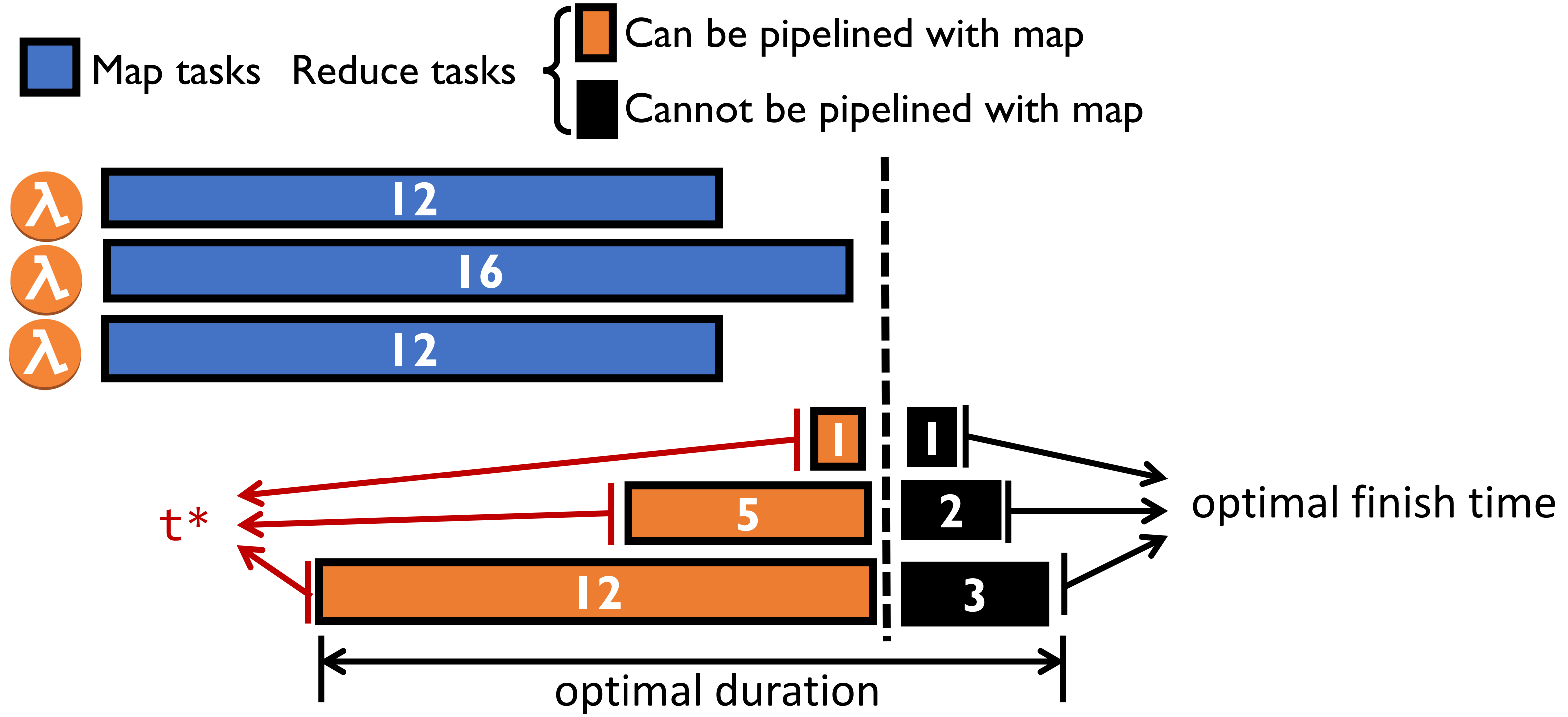
- **Optimal launch time in three simple steps**
  - Step 2: Calculate *optimal task finish time* based on *Eager* solution



Basic algorithm for 2-stage map-reduce

- **Optimal launch time in three simple steps**
  - Step 3: Calculate the task launch time ***t**** as:

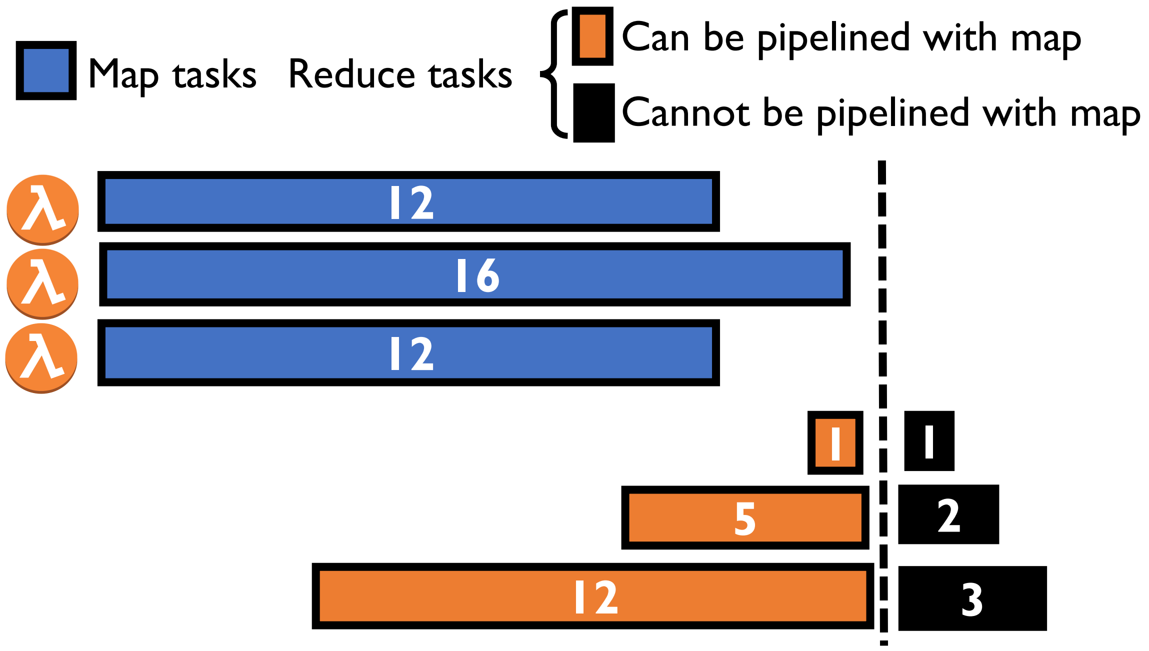
    optimal task finish time - optimal task duration



Basic algorithm for 2-stage map-reduce

- **Optimal launch time in three simple steps**

Theorem 1: ***t* ensures optimal cost and finish time*** for each reduce task.



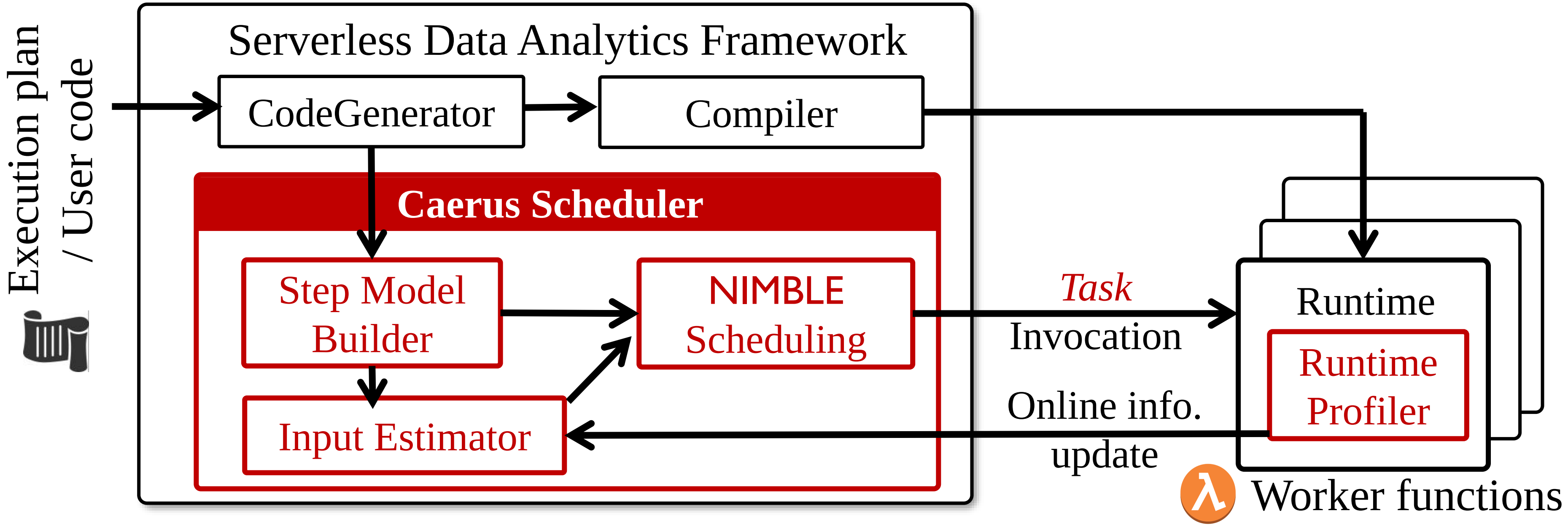
From map-reduce to arbitrary DAGs

- **Challenges for arbitrary DAGs :**
  - Within a stage: tasks can consume data from multiple upstream stages
  - Across stages: tasks can have cascading dependencies

- **Takeaways:**
  - **Bad news**: ***Impossible*** to design an algorithm that can achieve optimal cost and JCT ***simultaneously*** for arbitrary DAGs
  - **Good news:** Extend the basic algorithm to guarantee ***optimal cost*** while being ***Pareto-optimal*** between cost and JCT

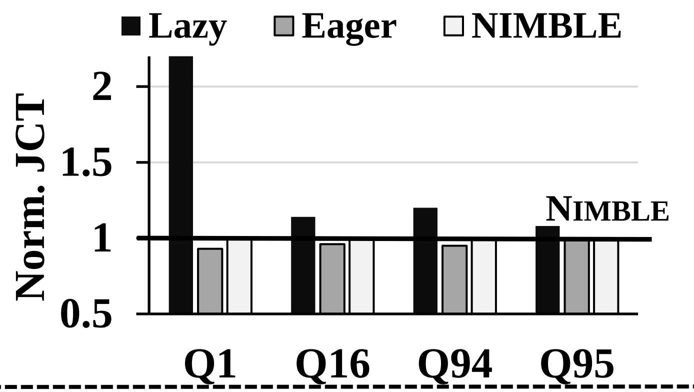
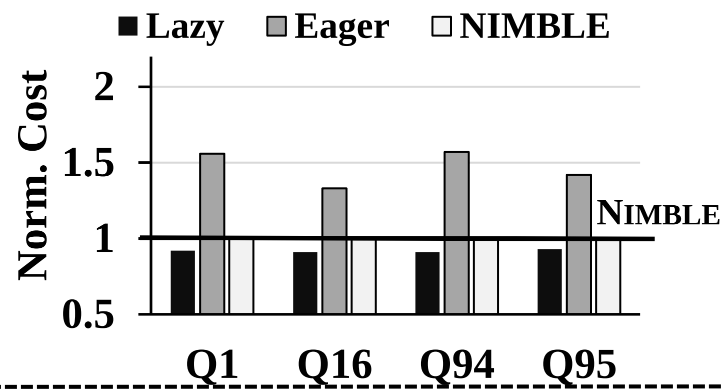
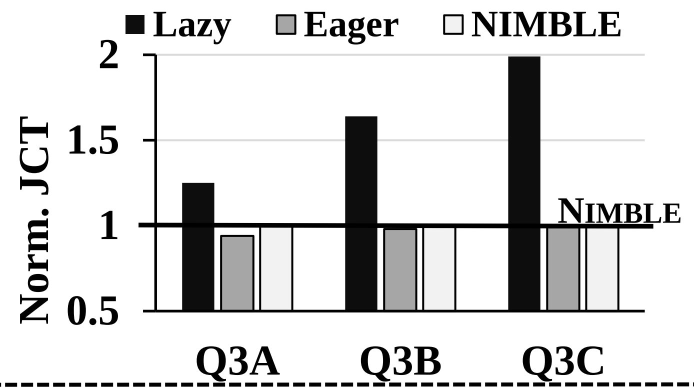
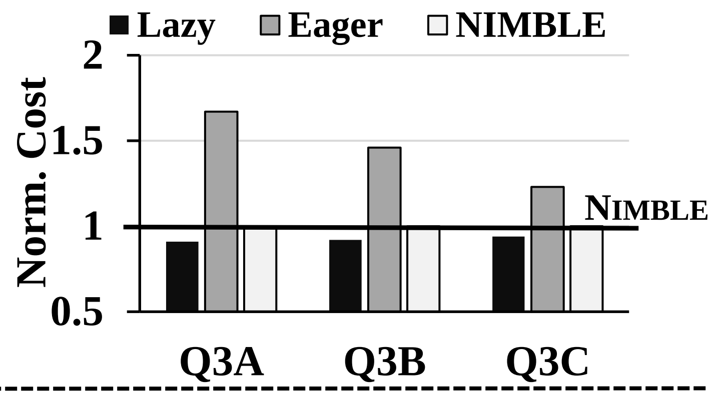
Caerus System

- Caerus: a task-level scheduler for serverless analytics which enables NIMBLE scheduling



Data Analytics Framework with **Caerus**

Evaluation results on AWS

	JCT (1.1-2.2X improvement)	Cost (1.2-1.6X improvement)
TPC-DS (4 queries)		
BigData (3 queries)		

NIMBLE scheduling can effectively optimize both JCT and cost across all these workloads

Takeaways

- **Serverless analytics introduces a new intra-job scheduling problem to optimize both JCT and cost**
  - Existing solutions expose a hard tradeoff between these two metrics
- **NIMBLE scheduling with a simple idea: to launch each task at *its right time***
  - Step model to capture sub-stage level pipelinablity and data dependencies
  - Achieves cost optimality while being Pareto-optimal between cost and JCT
- **Caerus: a task-level scheduler for serverless analytics which enables NIMBLE scheduling in practice**

Thank you!

Contact email:
hongzhangblaze@gmail.com