Sprocket: A Serverless Video Processing Framework

Lixiang Ao, Liz Izhikevich, Geoffrey M. Voelker, George Porter

UC San Diego

Video processing

```
$ ffmpeg -i input.mp4 -vf hue=s=0 greyscale.mp4
```

3 min clip vs. 120 min movie
4.5min vs. **190min** processing time

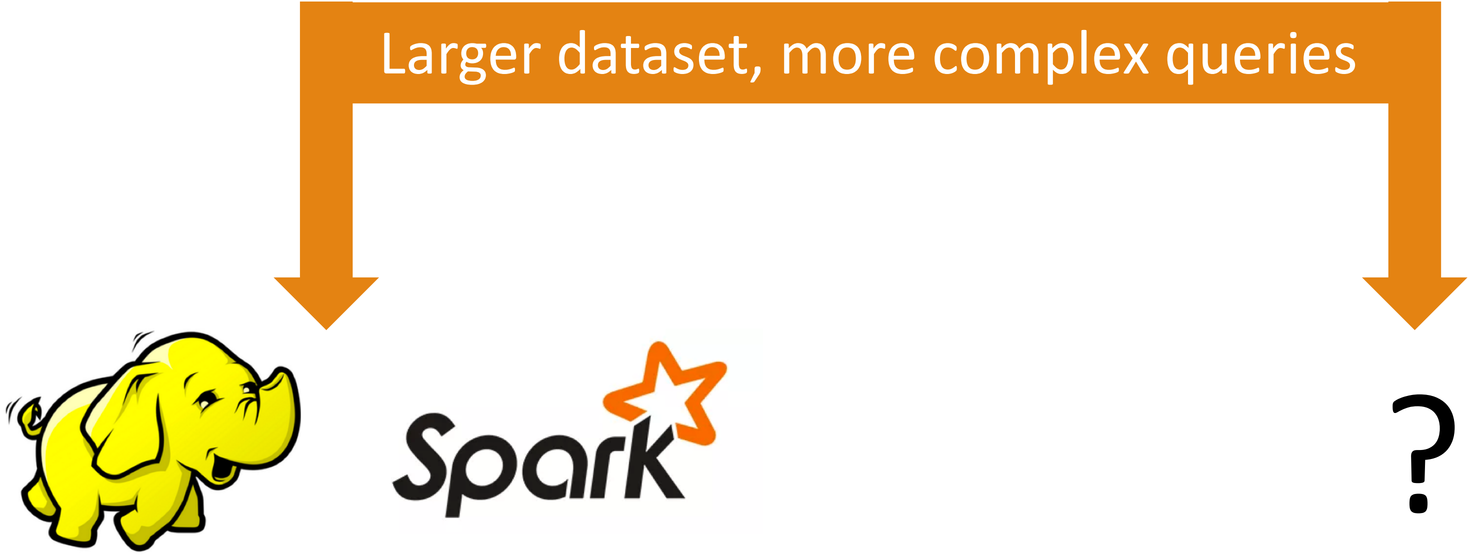
Low parallelism

"Show just the scenes in the movie in which Wonder Woman appears"

Complex queries not supported

```
$ tr ' ' '\n' < input | sort |
  uniq -c
```

```
$ ffmpeg -i input.mp4 -vf
  hue=s=0 greyscale.mp4
```



Larger dataset, more complex queries

```
$ tr ' ' '\n' < input | sort |
  uniq -c
```

```
$ ffmpeg -i input.mp4 -vf
  hue=s=0 greyscale.mp4
```

Larger dataset, more complex queries

A framework for
highly parallel,
complex video pipelines

Related work

ExCamera[NSDI '17]: Low latency video encoding w/ serverless, functional codec

Facebook SVE[SOSP '17]: Large scale video processing on dedicated cluster

Sprocket

Serverless video processing framework. (AWS Lambda)

Highly parallel, low-latency.

Low cost.

Build complex video pipelines with a simple domain-specific language.

Process an hour of 1080p video 1000-way parallelism in 10s seconds for < $3.

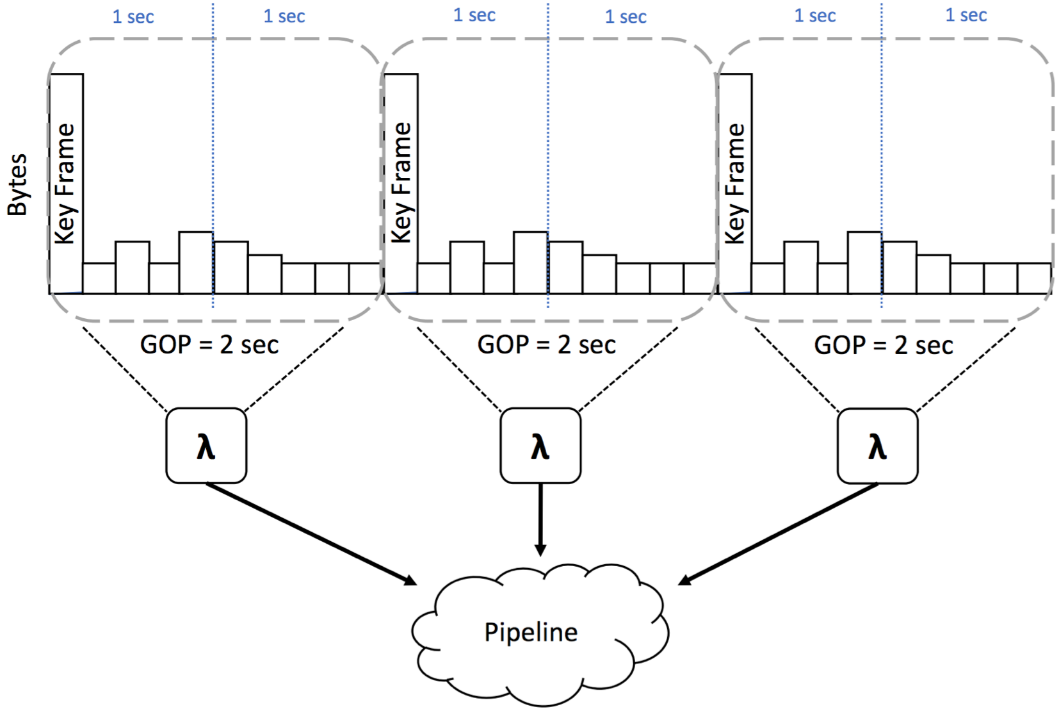
Intra-video parallelism

Video frames are interdependent within a Group of Pictures (GOP).

GOPs are independent of each other.

Each GOP is relative small in size.

Intra-video parallelism.



Why serverless?

Serverless: run user code in cloud without managing servers, e.g., AWS Lambda.

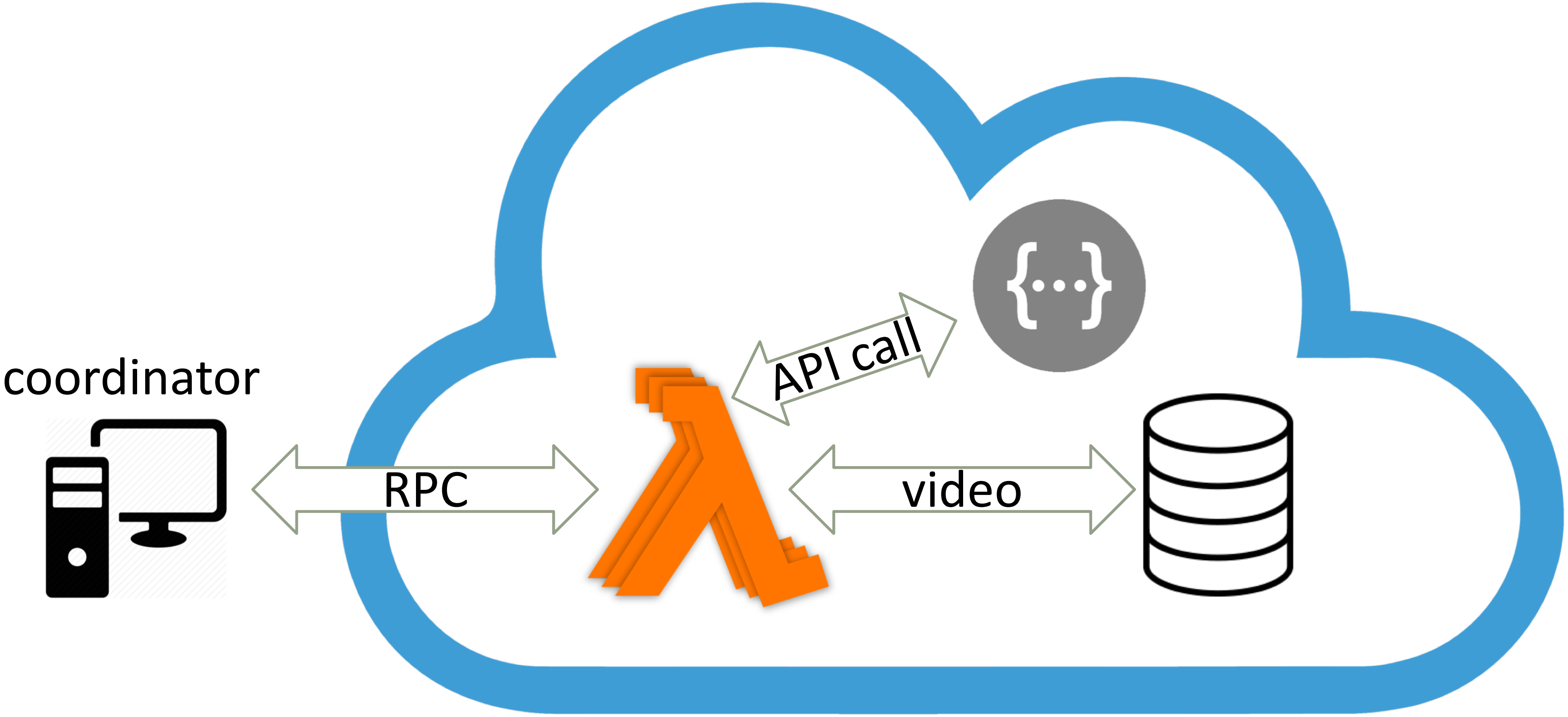
Each instance naturally matches GOP’s size.

Burst-parallelism – thousand of instances in sub-second on demand.

Only pay actual running time.

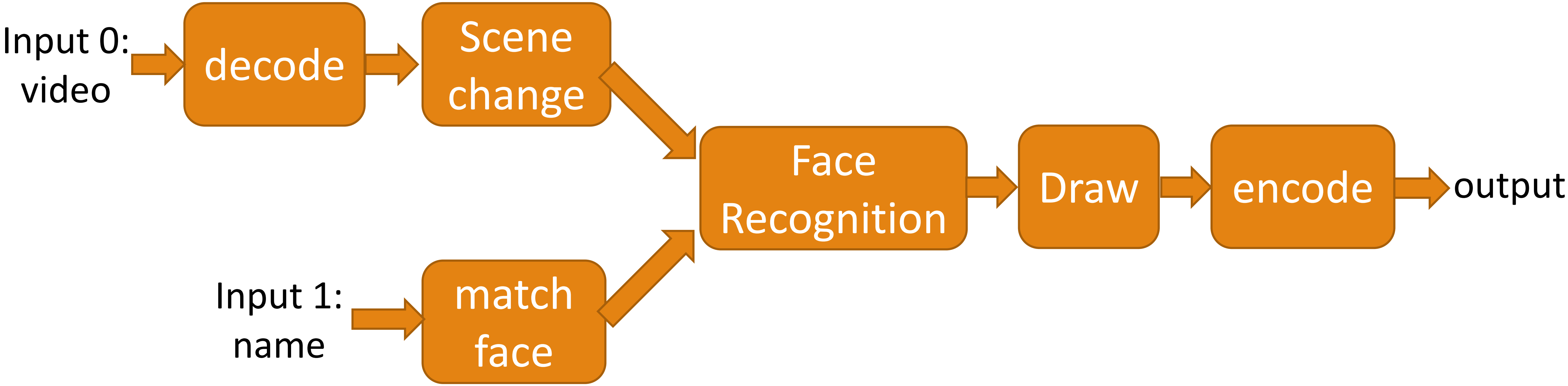
Cloud computer vision services, e.g., AWS Rekognition and Google Vision.

System Overview



How do we program Sprocket applications?

Logical DAG (Directed Acyclic Graph):



Domain-specific language: pipespec:

```
{
  "nodes":[
    {
      "name": "matchFace",
      "stage": "matchFace",
      "config": {
      }
    },
    {
      "name": "decode",
      "stage": "stealwork_decode",
      "config": {
        "stealwork": true,
        "transform": "-f image2 -c:v png"
      }
    },
    {
      "name": "face_rek",
      "stage": "rek",
      "delivery_function": "serialized_scene",
      "config": {
      }
    },
…
```

logical DAG node

control logic encoded in stages

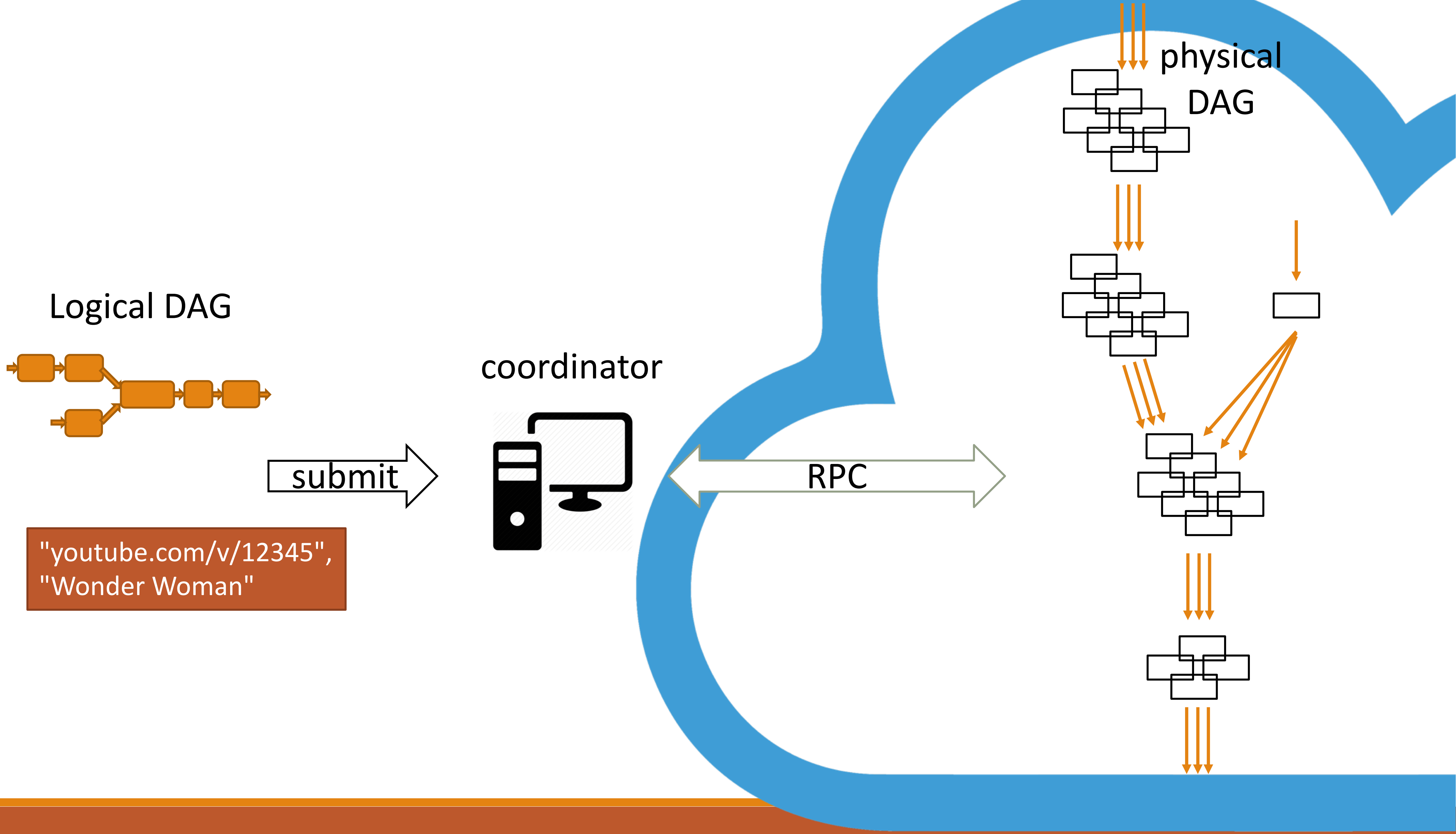
stage configs

dependency definition

```
"streams":
  [
    {
      "src": "input_0:chunks",
      "dst": "decode:chunks"
    },
    {
      "src": "input_1:person",
      "dst": "matchFace:person"
    },
    {
      "src": "decode:frames",
      "dst": "scenechange:frames"
    },
    {
      "src": "scenechange:scene_list",
      "dst": "face_rek:scene_list"
    },
    {
      "src": "face_rek:frame",
      "dst": "draw:frame"
    },
…
```

logical DAG edge

node:edgeID



Logical DAG

submit

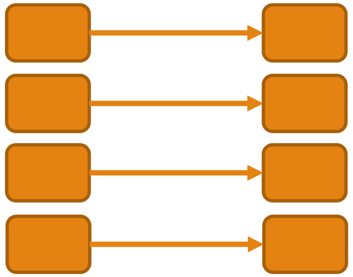
"youtube.com/v/12345",
"Wonder Woman"

coordinator

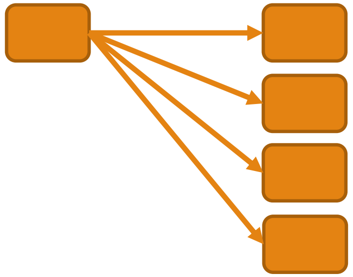
RPC

physical DAG

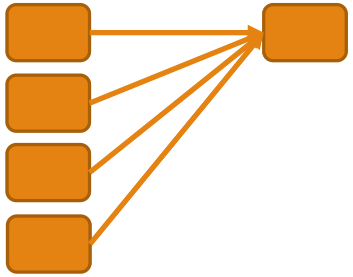
Data dependencies



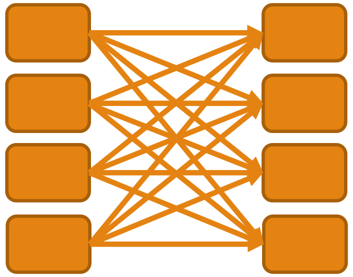
Chain of filters



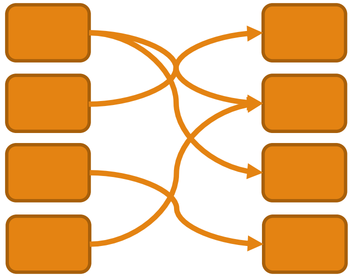
Decode to frames



Encode from frames

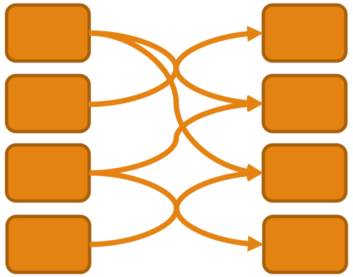


Full shuffling



User defined?

delivery function



$$f : (\mathrm{I}, \mathrm{global\ states}) \rightarrow (I \ \rightarrow O)$$

- user-defined dependency between upstream & downstream
- produces a mapping from inputs to outputs using inputs and/or global states
- dynamically updates physical DAG

Scheduling

Manages limited resources, e.g., concurrent Lambda workers

Simplified by serverless platform

Implements fine-grained (task-level) priority control

Priority is defined with an API

Streaming scheduler

Straggler mitigation

Stragglers seen in:

- Lambda Invocation
- Intermediate data I/O
- Worker task

Solved by:

- Worker late binding + over-provision
- Speculative I/O
- Work-stealing by exploiting the GOP structure

Evaluations

Questions we want to answer:

- Can Sprocket utilize burst-parallelism provided by serverless platforms?
- Can Sprocket schedule pipeline efficiently?
- Is Sprocket cost-efficient?
- Can Sprocket mitigate stragglers? (see paper)

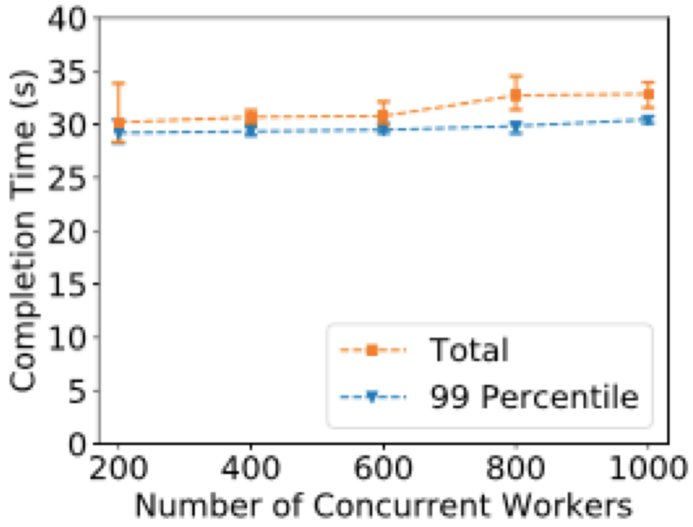
Parallelism tests

Three-stage greyscale pipeline

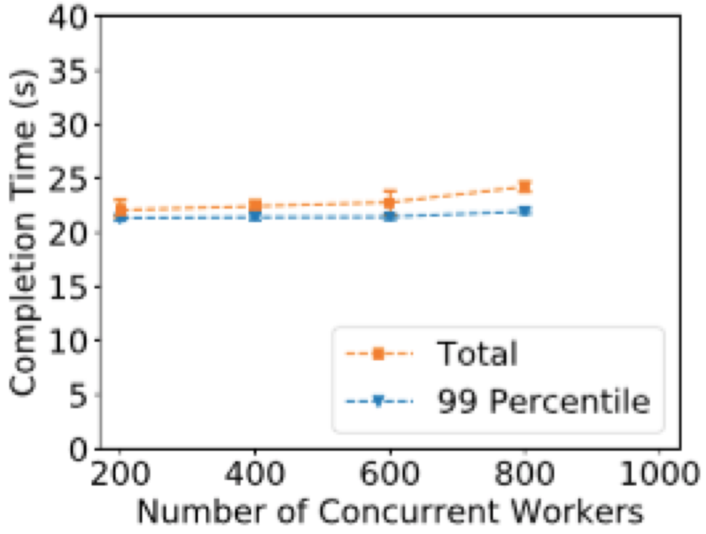
Each Lambda worker handles a GOP.

Pipeline completion time

Burst parallelism of serverless supports highly parallel video processing



(a) Synthetic



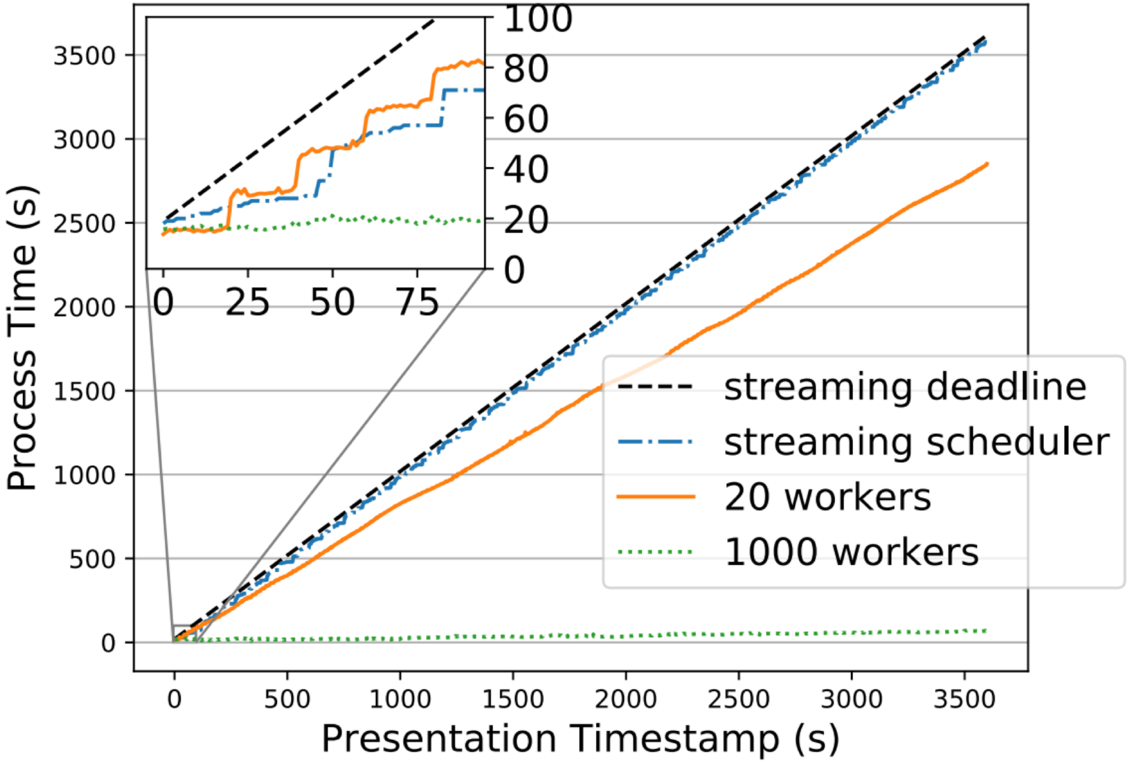
(c) Sintel

Streaming scheduler

Users consume output while video processed.

Meet streaming deadline while minimizing resource consumption.

Adjust number of workers according to progress and deadline.



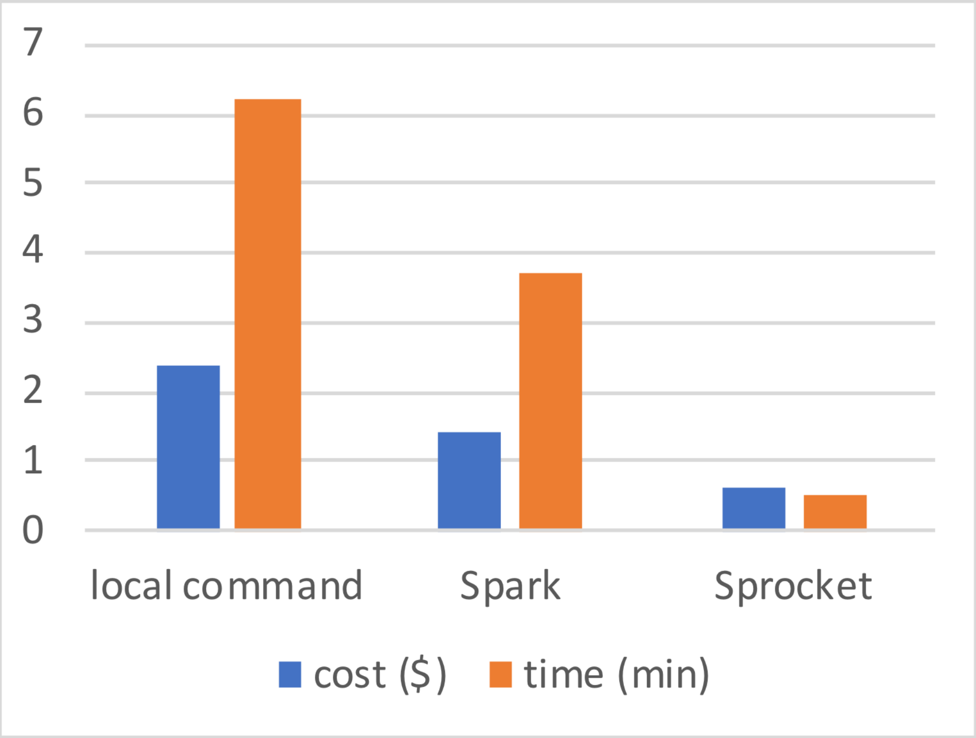
Monetary cost

FFmpeg greyscale filter on a 30-min 1080p video.

Local command: a m4.16xlarge instance w/64 cores, 256G RAM.

Spark: 18-node cluster m4.2xlarge instance w/8 cores, 32G RAM.

Sprocket: 900 concurrent 3G RAM Lambdas.



Conclusion

A framework for highly parallel, complex video processing is needed.

Serverless is an ideal platform for such a framework.

Sprocket introduces low-latency complex video processing with low cost.

Thank you!

Q & A