Encoding, Fast and Slow:
Low-Latency Video Processing Using Thousands of Tiny Threads

Presenter: Wen-Fu Lee

Outline

- Vision & Goals
- mu: Supercomputing as a Service
- Fine-grained Parallel Video Encoding
- Evaluation
- Takeaways

Outline

- Vision & Goals
- mu: Supercomputing as a Service
- Fine-grained Parallel Video Encoding
- Evaluation
- Takeaways

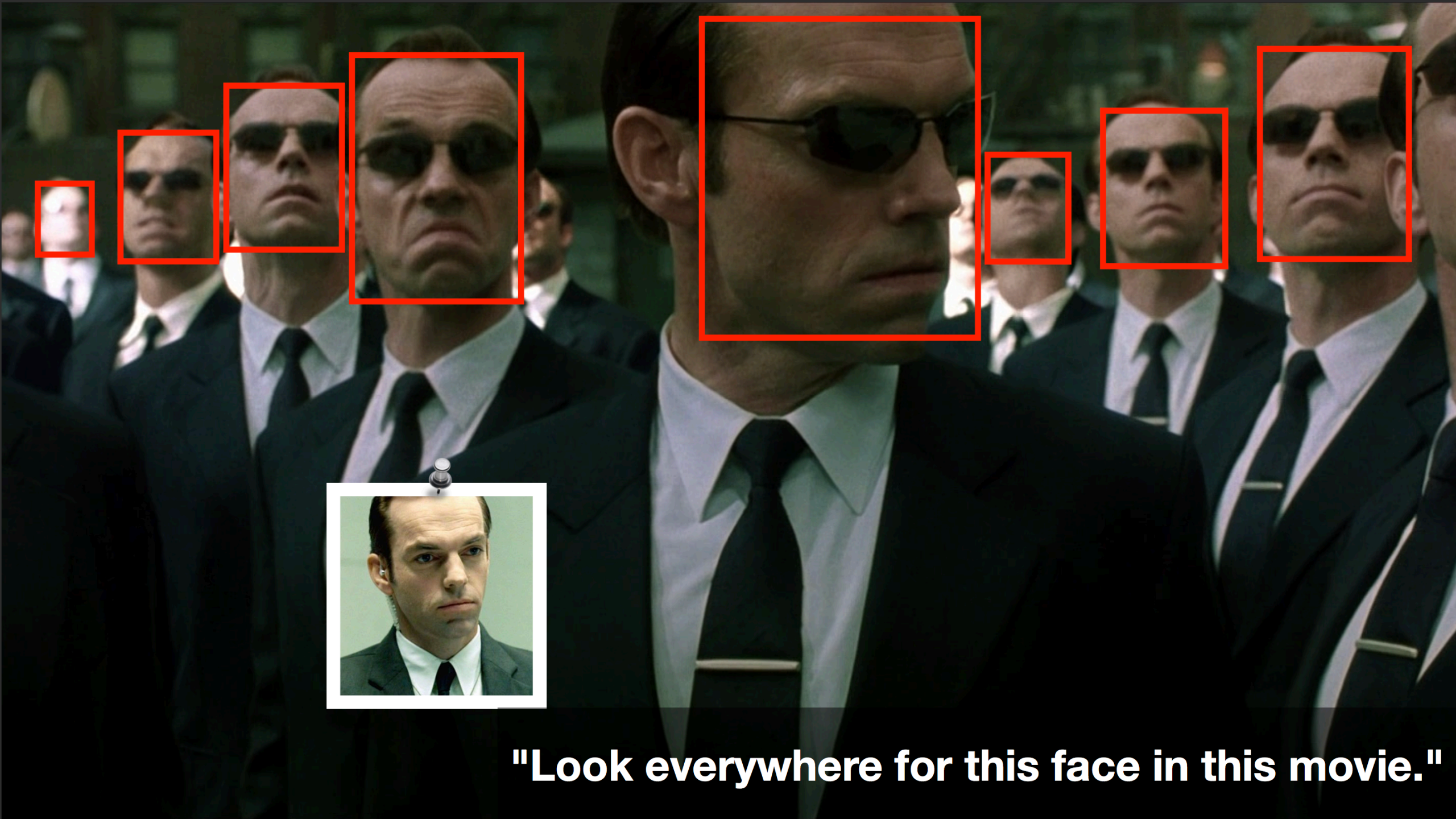
What we currently have

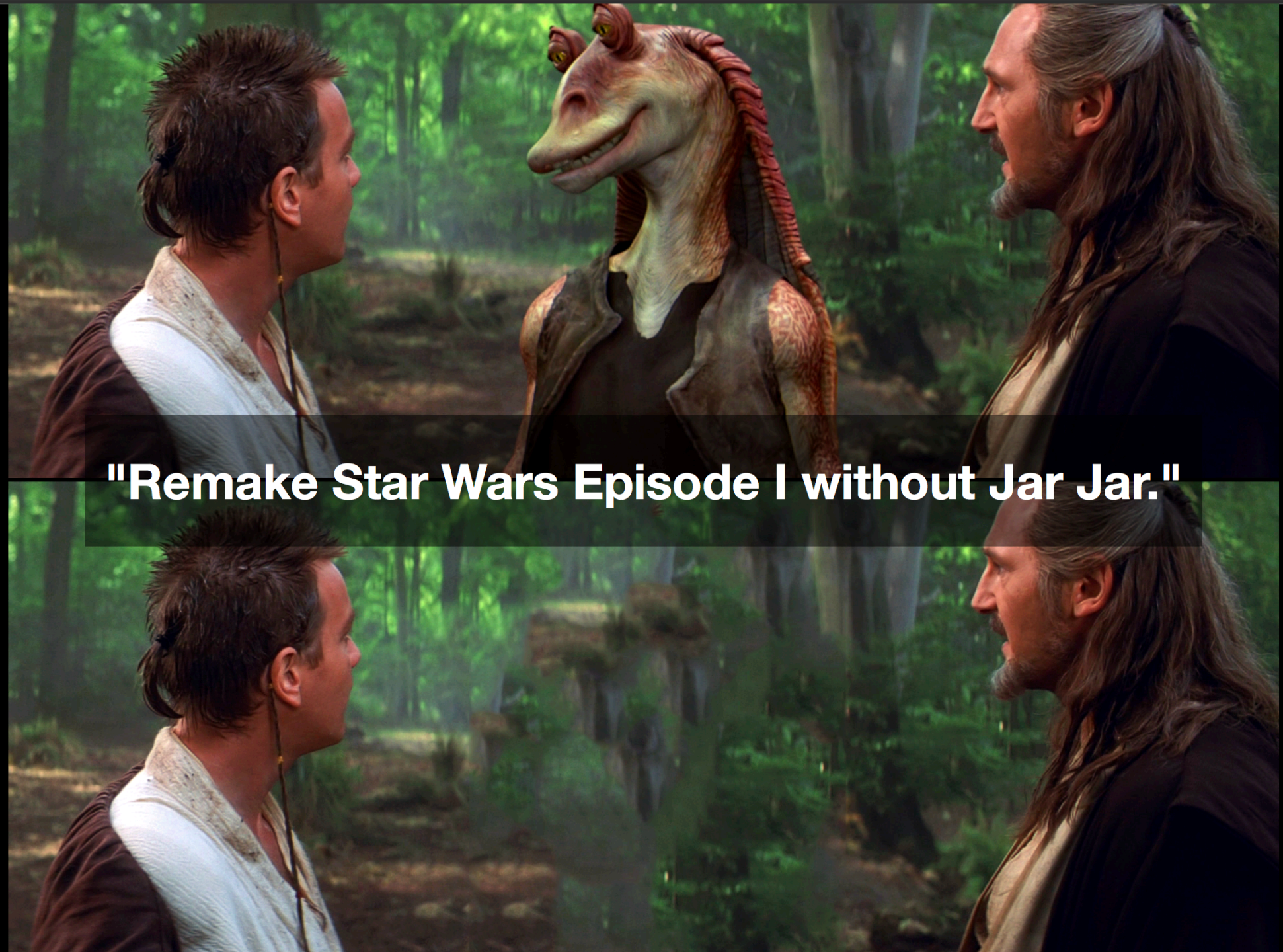
- People can make changes to a word-processing document
- The changes are instantly visible to the others

What we would like to have

Google Docs *for Video*?

- People can interactively edit and transform a video
- The changes are instantly visible to the others


"Look everywhere for this face in this movie."


"Remake Star Wars Episode I without Jar Jar."

The Problem

Currently, running such pipelines on videos takes hours and hours, even for a short video.

The Question

Can we achieve interactive collaborative video editing by using massive parallelism?

The challenges

- Low-latency video processing would need **thousands of threads**, **running in parallel**, with **instant startup.**

- However, **the finer-grained the parallelism, the worse the video compression efficiency.**

ExCamera

- Two contributions
  - Framework to run **5,000-way parallel jobs** with IPC* on a commercial "cloud function" service.
  - Purely functional video codec for **massive fine-grained parallelism**.

*Inter-process communication (IPC)

Outline

- Vision & Goals
- mu: Supercomputing as a Service
- Fine-grained Parallel Video Encoding
- Evaluation
- Takeaways

Where to find thousands of threads?

	Virtual machine
Cloud Service Providers	Amazon: EC2 Microsoft: Azure Google: GCE
Think about it as	Base layer Unit = VM
Pros & cons	**[+]** Thousands of threads **[+]** Arbitrary Linux executables **[-]** Minute-scale startup time - OS has to boot up, ... **[-]** High minimum cost - 60 mins EC2, 10 mins GCE
Running 3,600 threads for 1 sec	> $20

Where to find thousands of threads?

	Virtual machine	Cloud function
Cloud Service Providers	Amazon: EC2 Microsoft: Azure Google: GCE	AWS Lambda Google Cloud Functions
Think about it as	Base layer Unit = VM	Event-driven compute (microservice) Unit = function
Pros & cons	**[+]** Thousands of threads **[+]** Arbitrary Linux executables **[-]** Minute-scale startup time - OS has to boot up, ... **[-]** High minimum cost - 60 mins EC2, 10 mins GCE	**[+]** Thousands of threads **[+]** Arbitrary Linux executables **[+]** Sub-second startup **[+]** Sub-second billing
Running 3,600 threads for 1 sec	> $20	10 cents

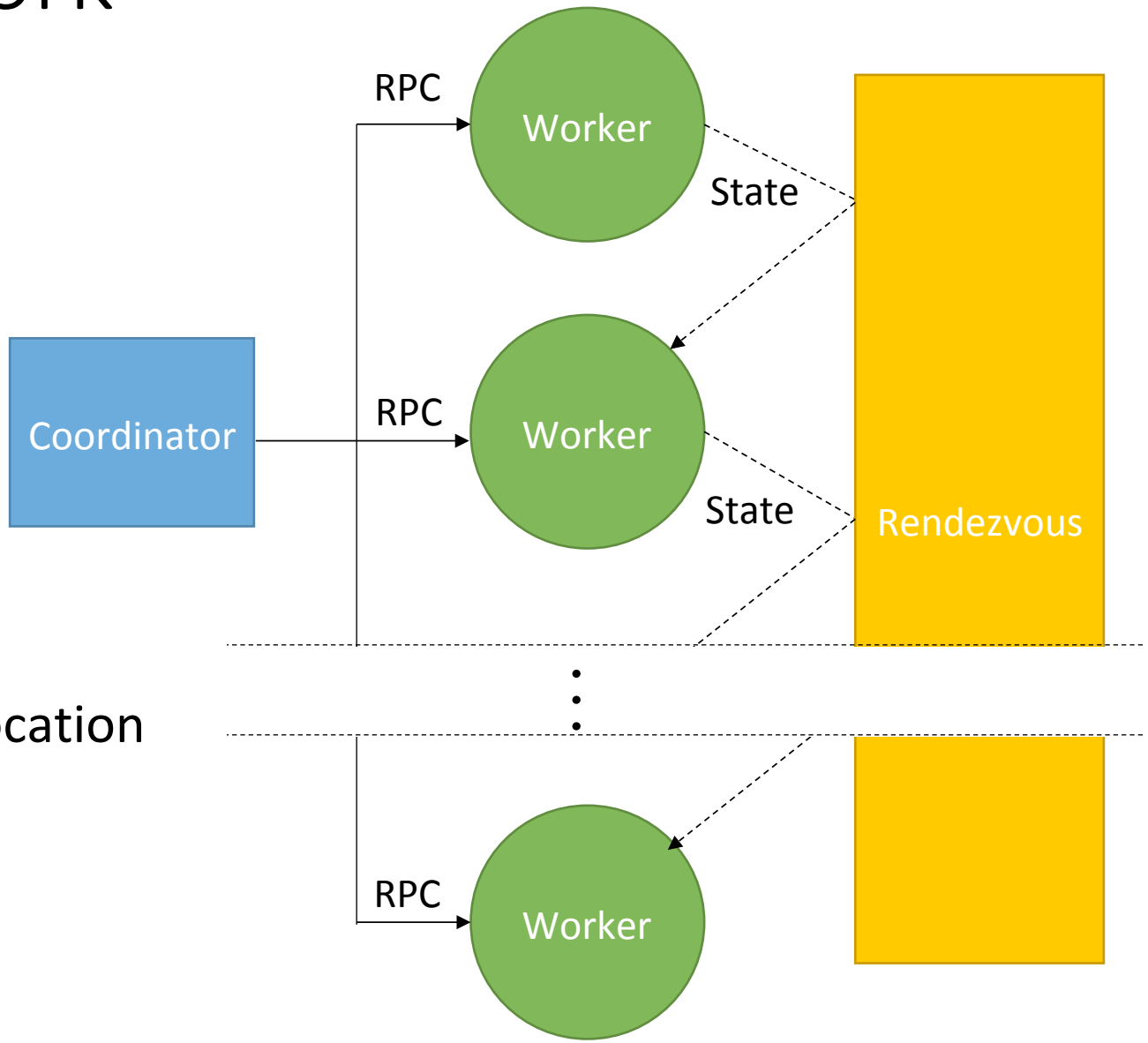
mu, supercomputing as a service

- ***mu***, a library for designing and deploying general-purpose parallel computations on **AWS Lambda**.

- The system starts up thousands of threads in seconds and manages inter-thread communication.

mu software framework

- Coordinator
  - Long-lived server
  - Dependency-aware scheduling
- Rendezvous
  - Long-lived server
  - Inter-thread communication
- Workers
  - Short-lived Lambda function invocation



Outline

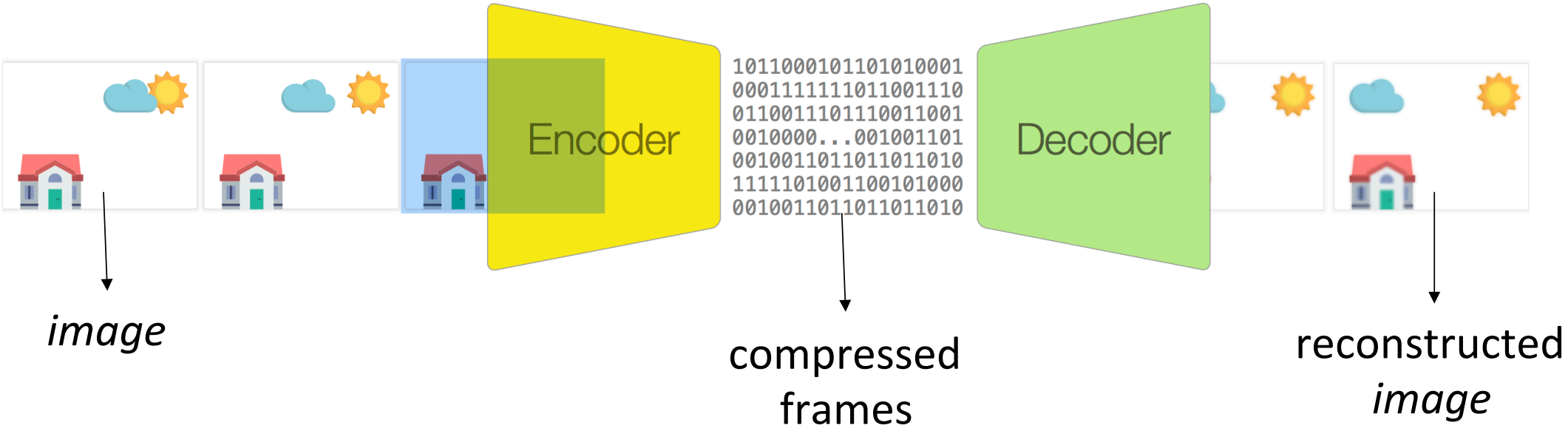
- Vision & Goals
- mu: Supercomputing as a Service
- Fine-grained Parallel Video Encoding
- Evaluation
- Takeaways

Now we have the threads, but…

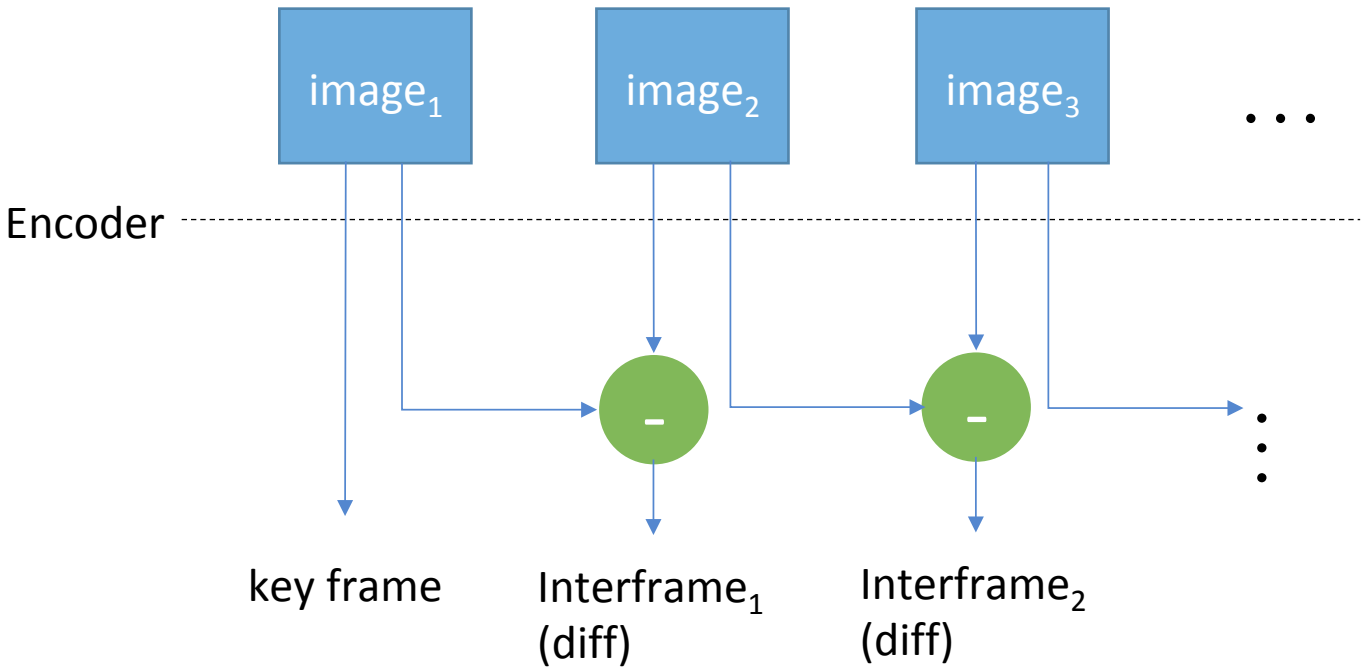
- With the existing encoders, the finer-grained the parallelism, the worse the compression efficiency.

Video Codec

- A piece of software or hardware that compresses and decompresses a digital video.



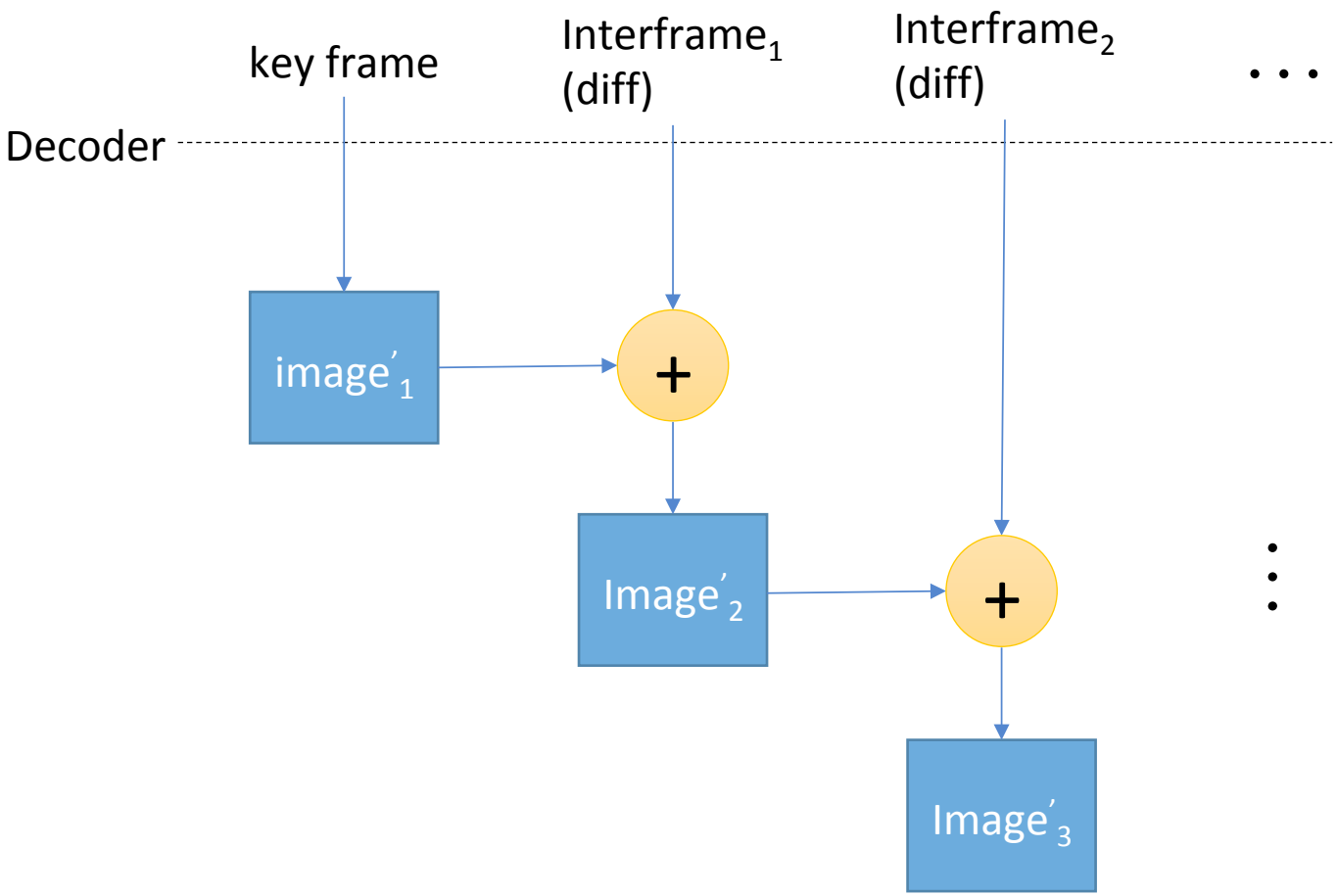
Encoder



`encode([🌄,🌄,...,🌄])` → `keyframe + interframe[2:n]`

In a 4K video @15Mbps, a key frame is **~1 MB**, but an interframe is **~25 KB**.

Decoder




```
decode(keyframe + interframe[2:n]) → [🌅,🌅,...,🌅]
```

Traditional parallel video encoding is limited

serial ↓

```
encode(i[1:200])    → keyframe₁ + interframe[2:200]
```

Traditional parallel video encoding is limited

serial ↓

```
encode(i[1:200])    → keyframe₁ + interframe[2:200]
```

parallel ↓

```
[thread 01] encode(i[1:10])     → kf₁ + if[2:10]
[thread 02] encode(i[11:20])    → kf₁₁ + if[12:20]
[thread 03] encode(i[21:30])    → kf₂₁ + if[22:30]
          ⋮
[thread 20] encode(i[191:200])  → kf₁₉₁ + if[192:200]
```

+1 MB

+1 MB

+1 MB

Traditional parallel video encoding is limited

serial ↓

```
encode(i[1:200])    → keyframe₁ + interframe[2:200]
```

parallel ↓

```
[thread 01] encode(i[1:10])     → kf₁ + if[2:10]
[thread 02] encode(i[11:20])    → kf₁₁ + if[12:20]
[thread 03] encode(i[21:30])    → kf₂₁ + if[22:30]
          ⋮
[thread 20] encode(i[191:200])  → kf₁₉₁ + if[192:200]
```

+1 MB

+1 MB

+1 MB

finer-grained parallelism ⇒ more key frames ⇒ worse compression efficiency

What we built: a video codec in an explicit state-passing style

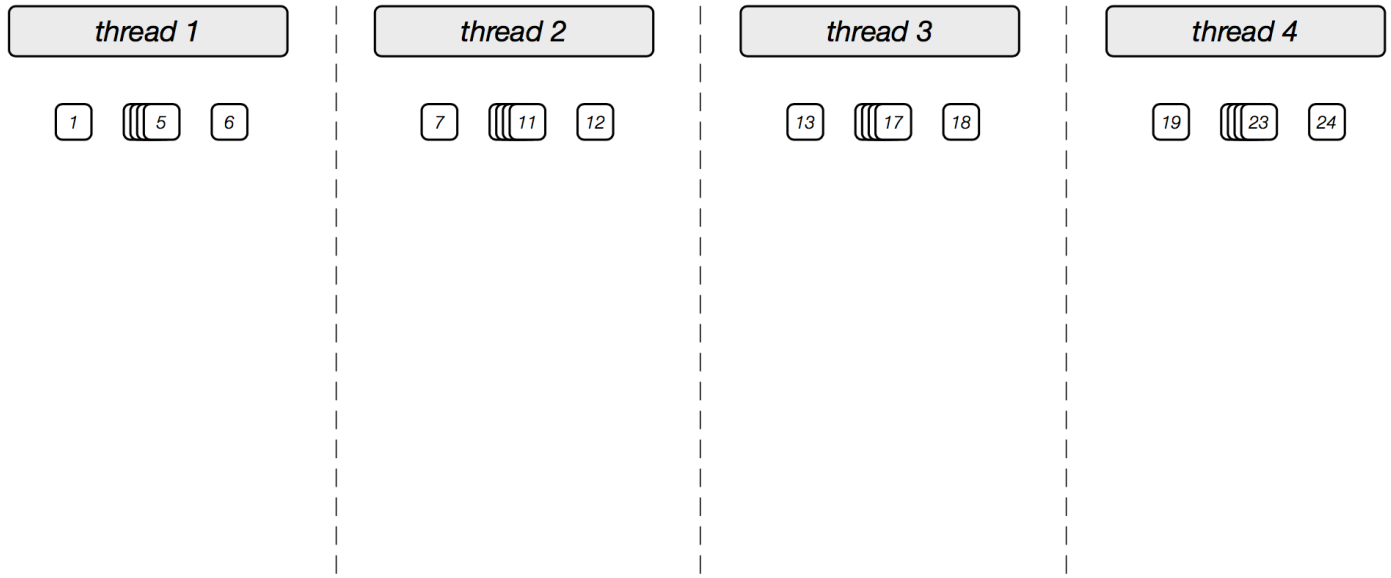
- VP8 decoder with no inner state:
  - **decode**(state, frame) → (state’, image)

- VP8 encoder: resume from specified state
  - **encode**(state, image) → interframe

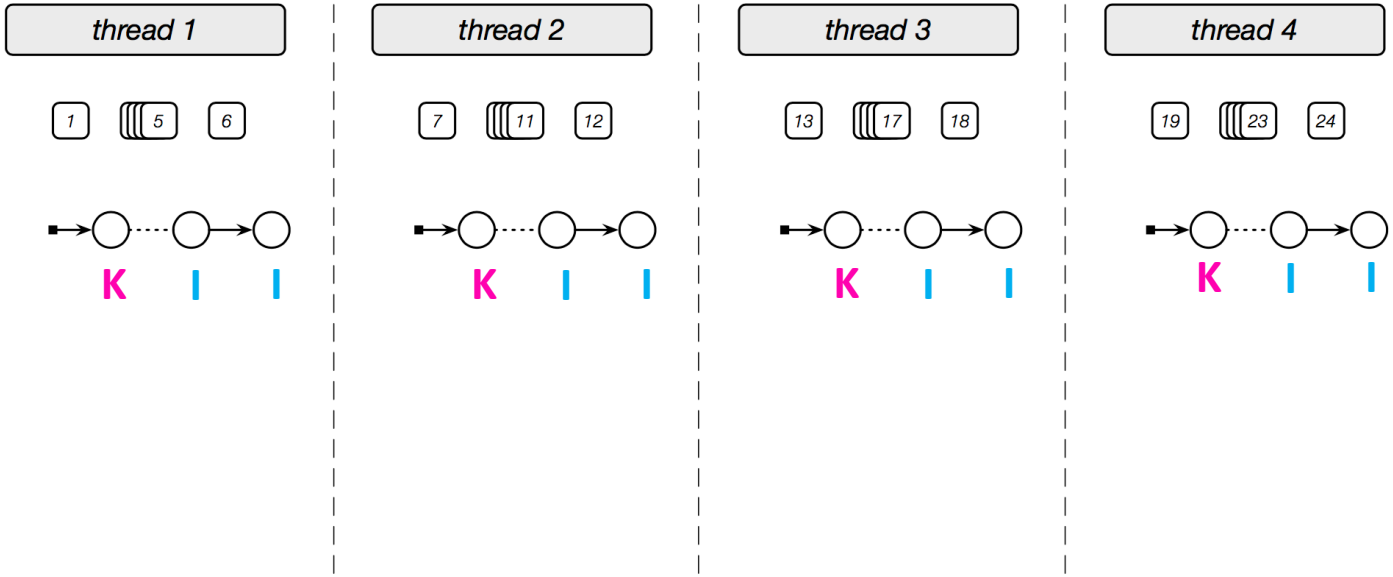
- Adapt a frame to a different source state
  - **rebase**(state, image, interframe) → interframe’

ExCamera
Encoder’s Algorithm

1. [Parallel]
Download a tiny chunk of raw video



2. [Parallel]
Google’s VP8 encoder

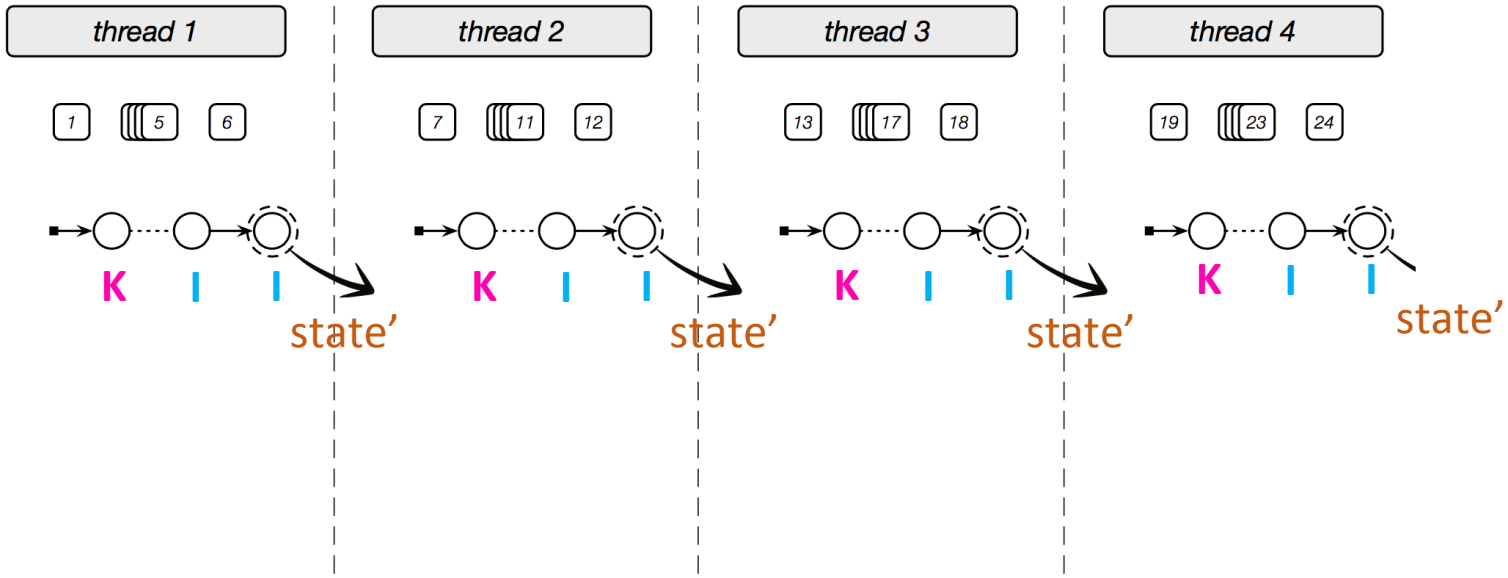


Google's VP8 encoder

```
encode(img[1:n]) → keyframe + interframe[2:n]
```

3. [Parallel]
decode(state, frame)

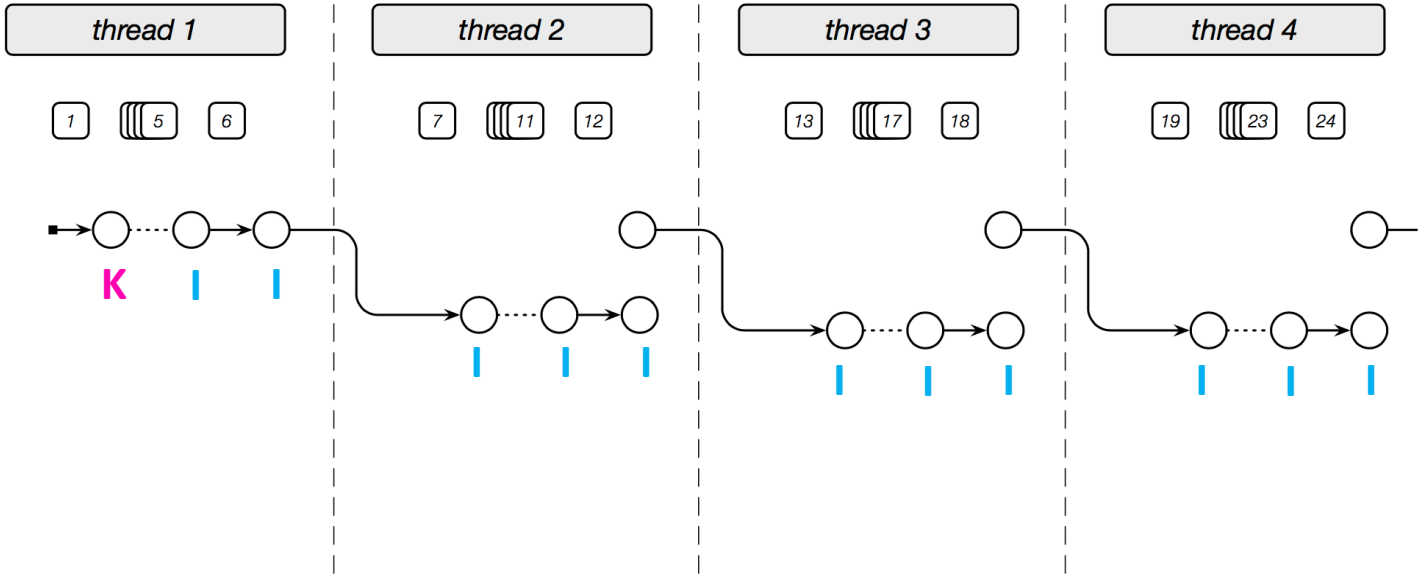
state:=(images'[3])



Our explicit-state style decoder

```
decode(state, frame) → (state′, image)
```

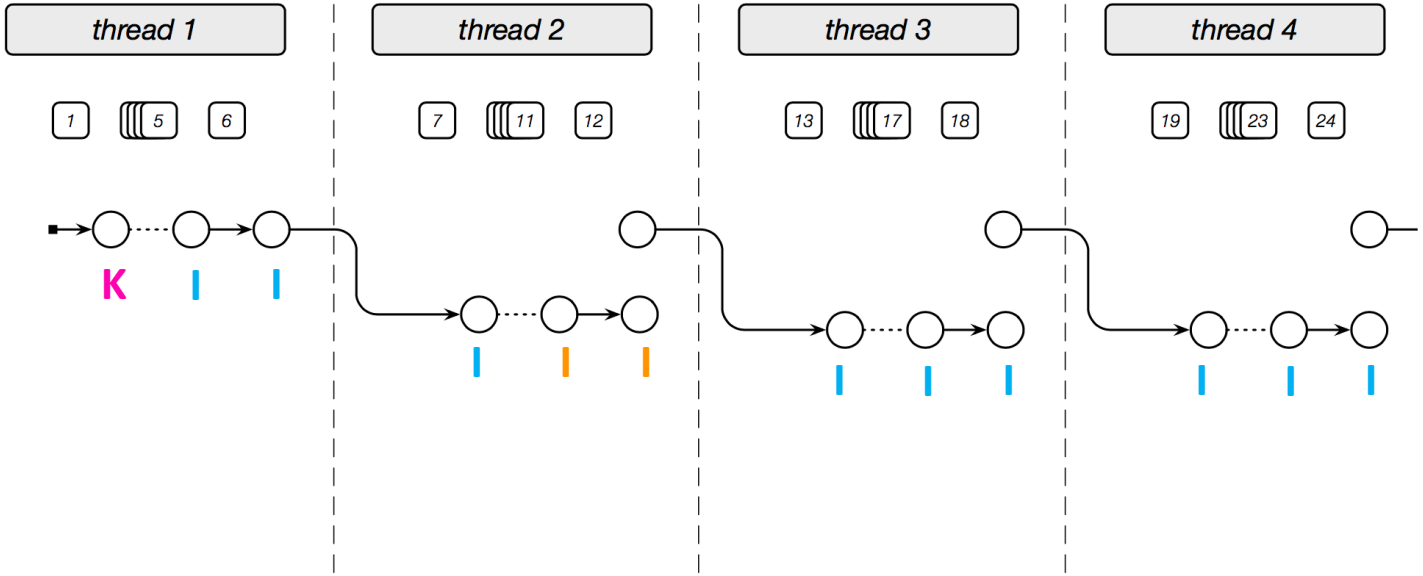
4. [Parallel]
encode(state, image)



Our explicit-state style encoder

```
encode(state, image) → interframe
```

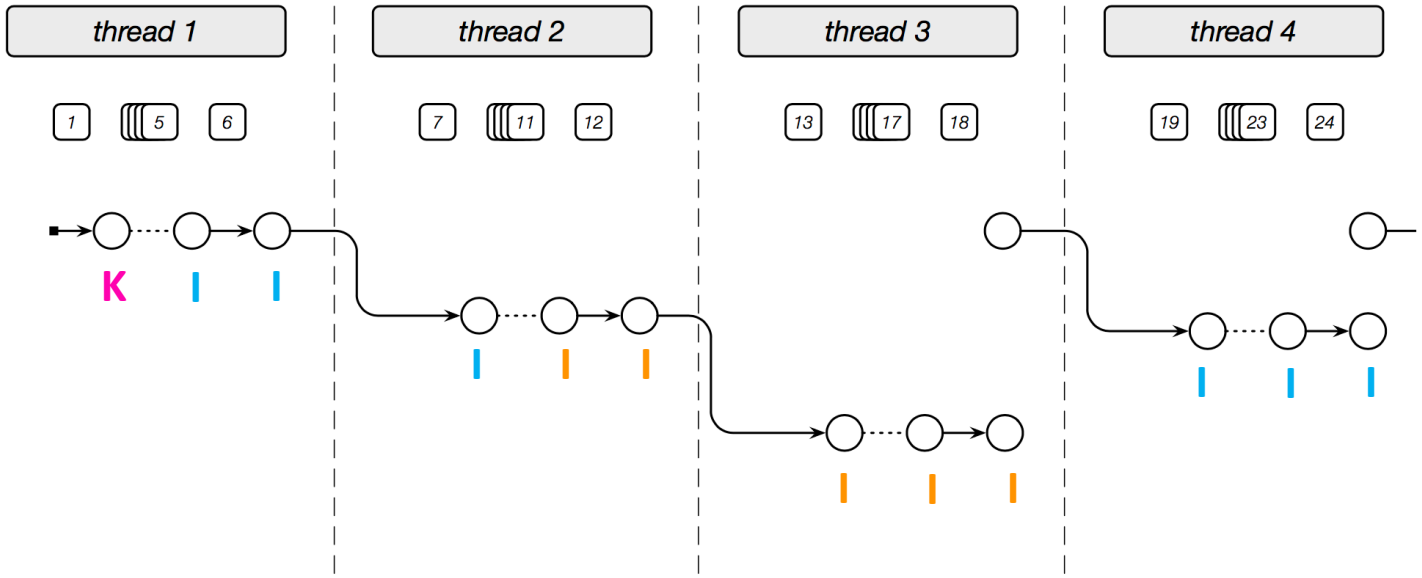
5. [Serial]
rebase(state, image, interframe)



Adapt a frame to a different source state

```
rebase(state, image, interframe) → interframe′
```

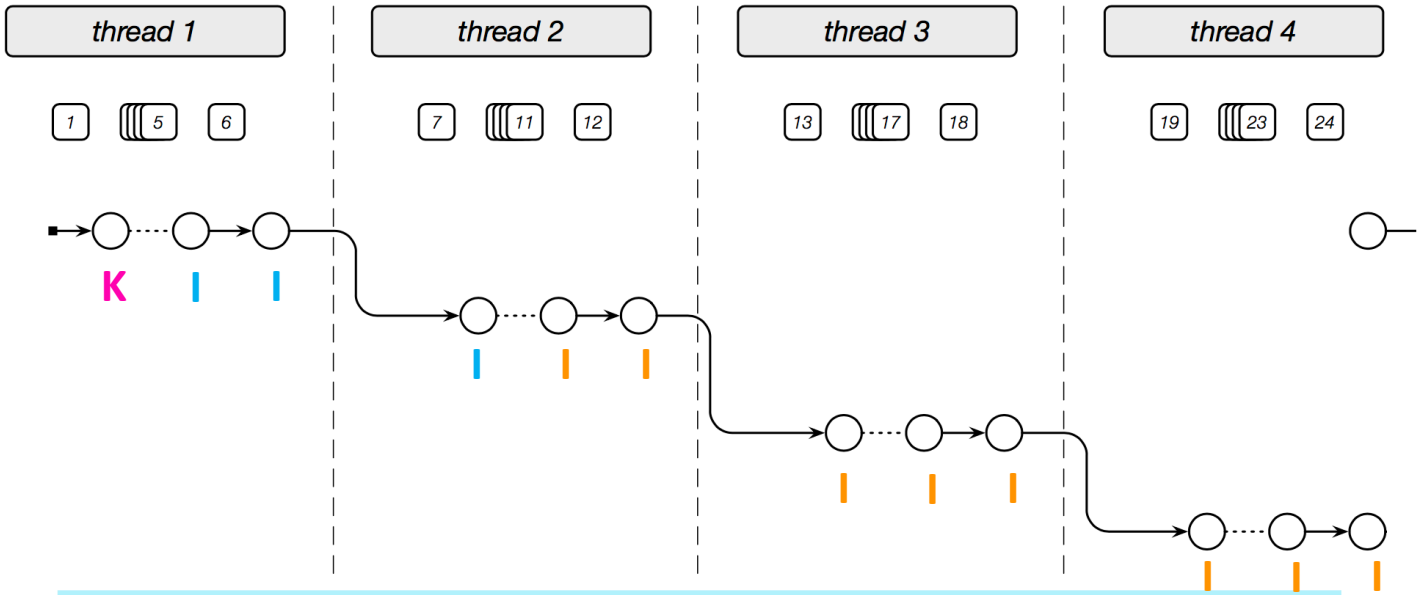
5. [Serial]
rebase(state, image, interframe)



Adapt a frame to a different source state

```
rebase(state, image, interframe) → interframe′
```

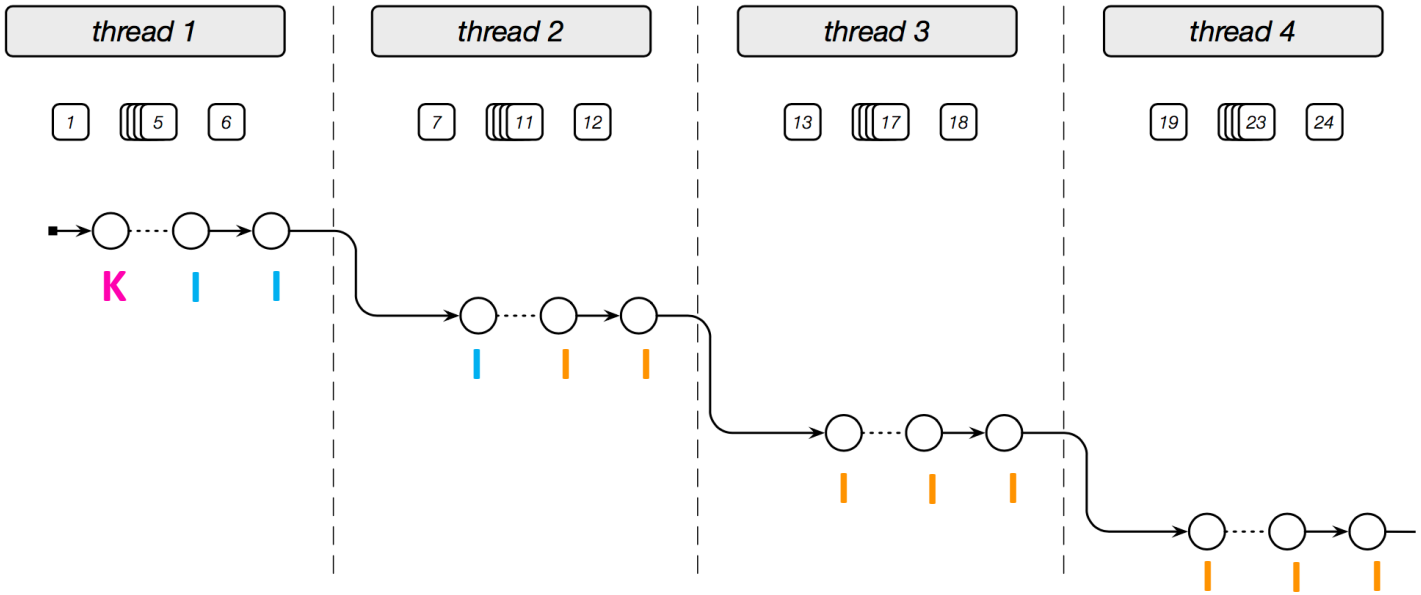
5. [Serial]
rebase(state, image, interframe)



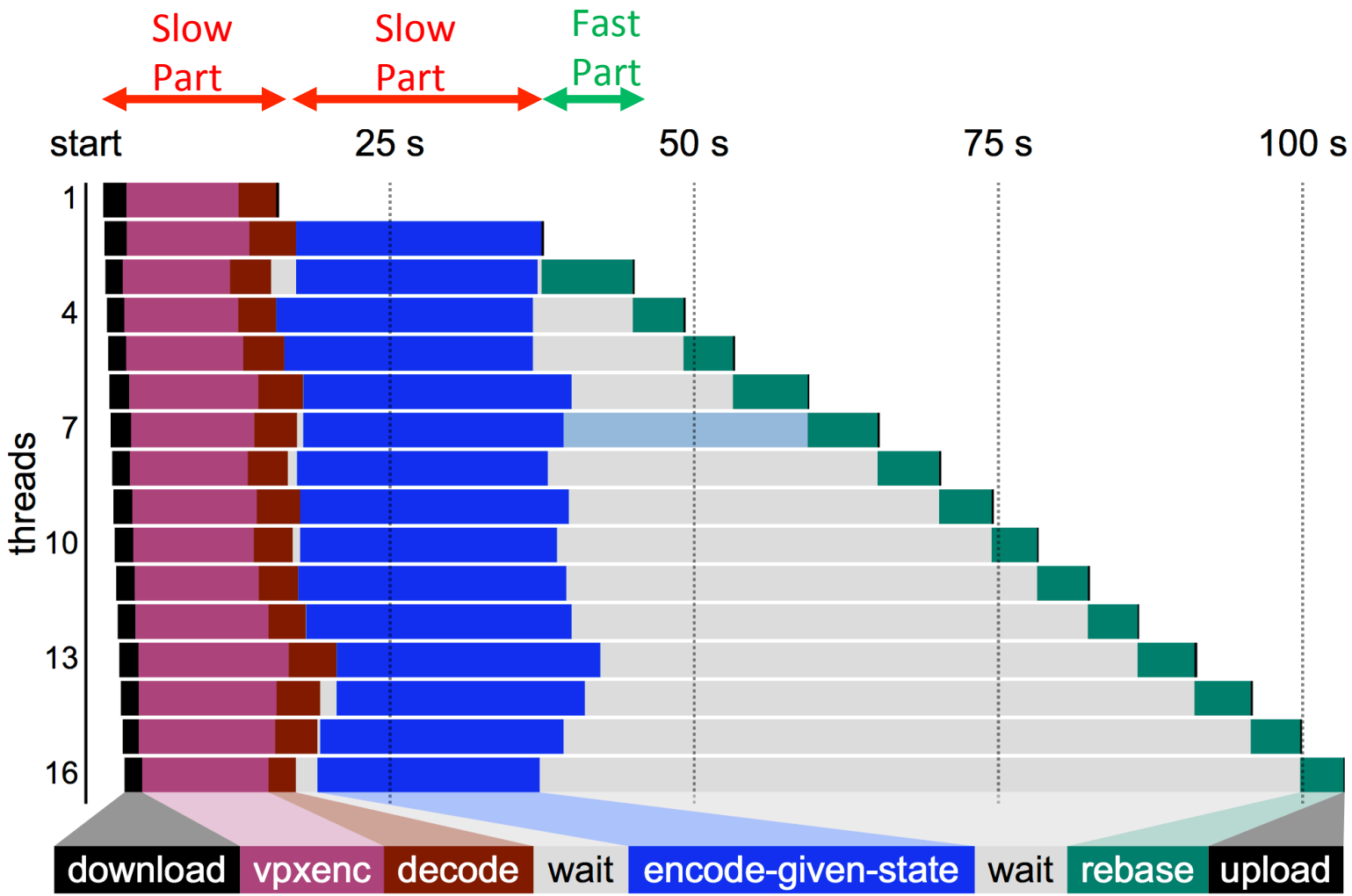
Adapt a frame to a different source state

```
rebase(state, image, interframe) → interframe′
```

6. [Parallel]
Upload finished video



Time Distribution



Wide range of different configurations

ExCamera[n, x]

number of frames in each chunk

Wide range of different configurations

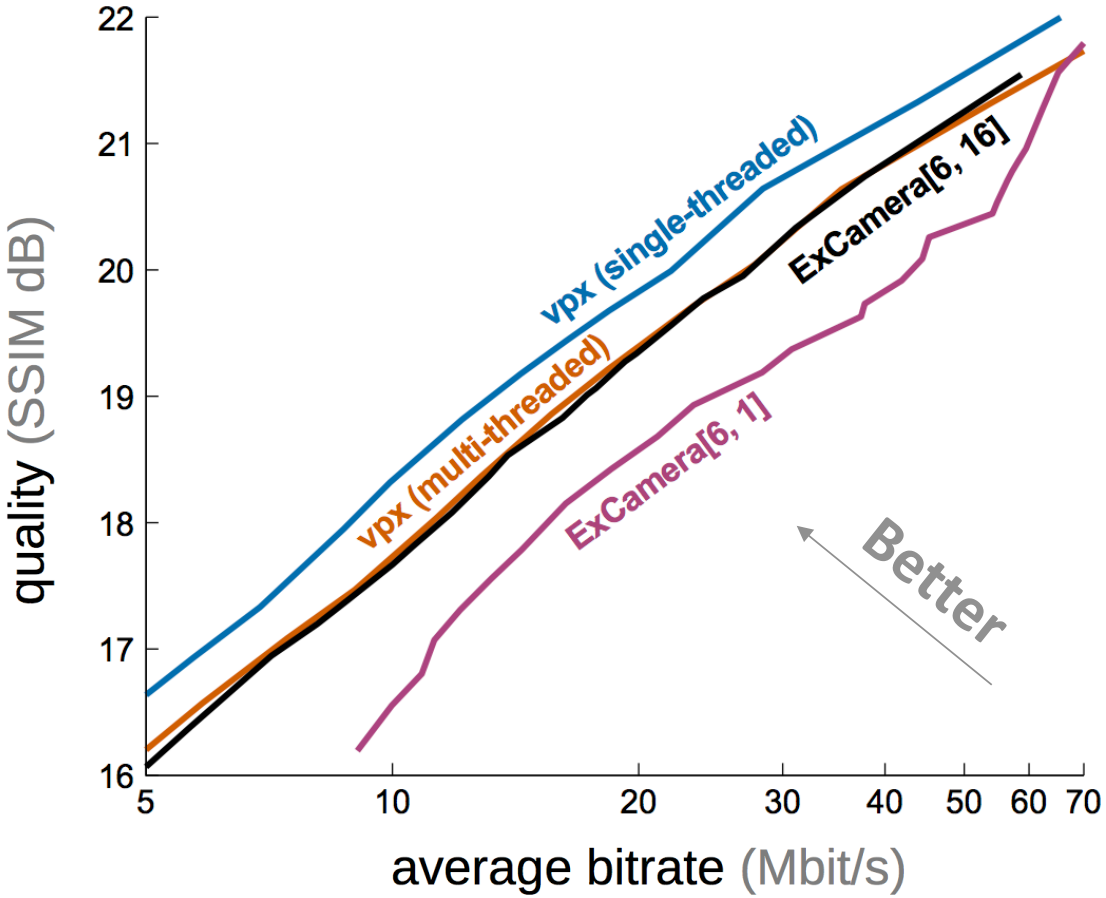
ExCamera[n, x]

number of chunks "rebased" together

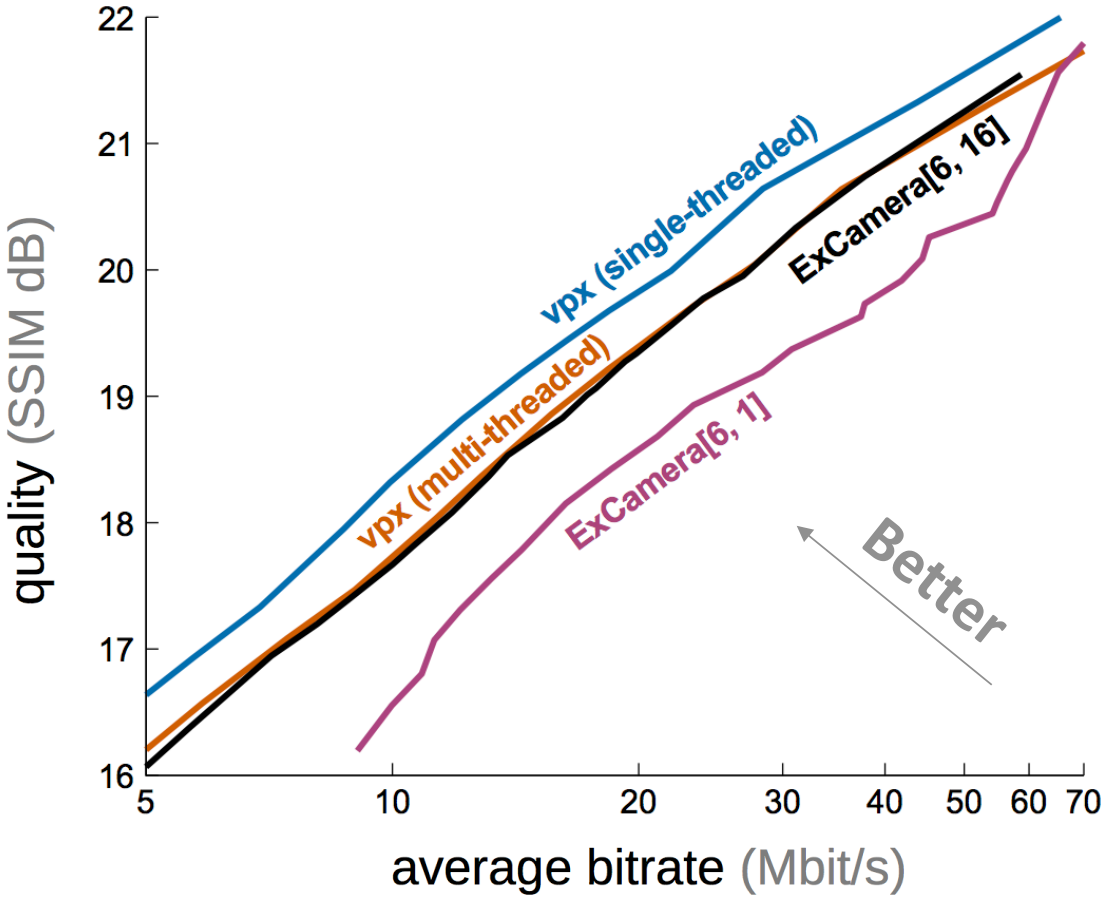
Outline

- Vision & Goals
- mu: Supercomputing as a Service
- Fine-grained Parallel Video Encoding
- Evaluation
- Takeaways

How well does it compress?



How well does it compress?



14.8-minute **4K** Video *@20dB*	
vpxenc Single-Threaded	**453 mins**
vpxenc Multi-Threaded	**149 mins**
ExCamera[6, 16]	**2.6 mins**

Encoding Speed

Outline

- Vision & Goals
- mu: Supercomputing as a Service
- Fine-grained Parallel Video Encoding
- Evaluation
- Takeaways

ExCamera vs. PyWren

	PyWren	ExCamera
Same	Using AWS Lambda	
Different	No	Inter-thread communication
	Serverless	Coordinator & rendezvous

Takeaways

- Target: Low-latency video processing

- Two major contributions
  - Framework to run **5,000-way parallel jobs** with IPC on AWS Lambda.
  - Purely functional video codec for **massive fine-grained parallelism**.

- 56× faster than existing encoder, for <$6.
  - Lots of speedup from fine-grained parallelism -> need to restructure the application to get maximum benefits out of it.

Reference

- http://pages.cs.wisc.edu/~shivaram/cs744-readings/excamera.pdf
- https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi
- https://doublehorn.com/comparing-the-big-3-aws/
- https://en.wikipedia.org/wiki/VP8

Thanks for your attention.

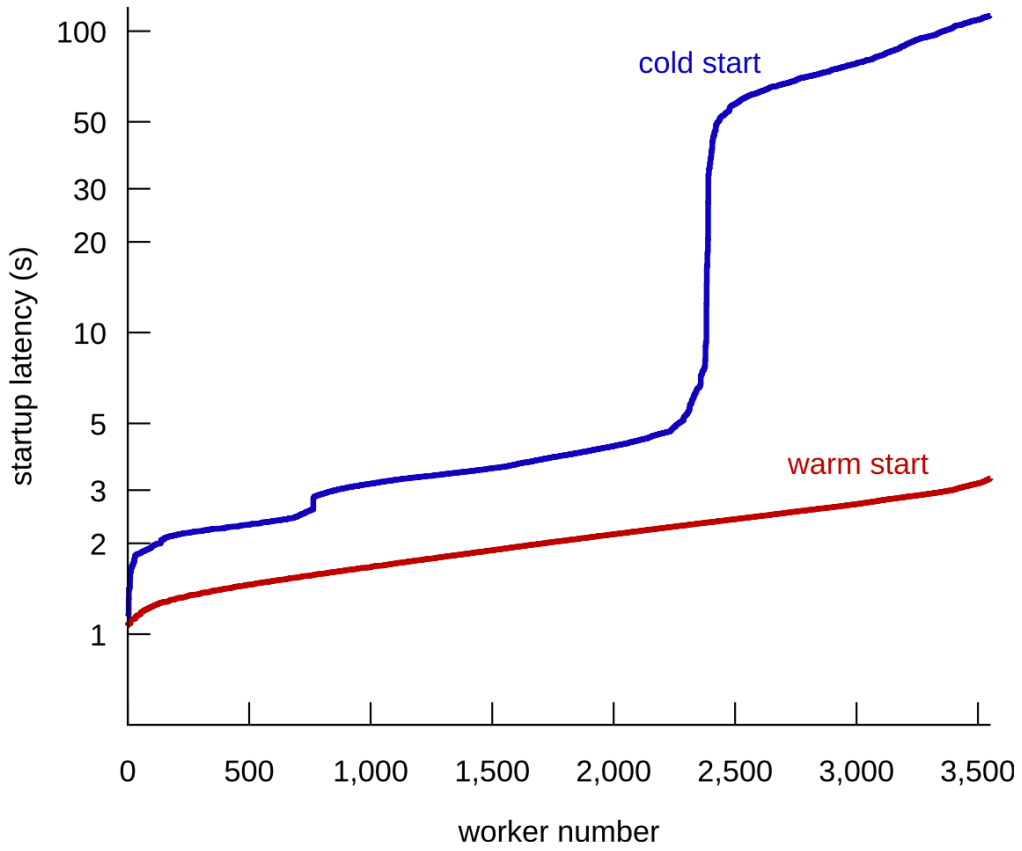
Q&A

Backup

Functions

- $\texttt{state} := (\texttt{prob\_model}, \texttt{references}[3])$
- **decode**$(\texttt{state}, \texttt{compressed\_frame}) \rightarrow (\texttt{state}', \texttt{image})$
  - **decode**$(\textit{any state}, \texttt{key\_frame}) \rightarrow (\texttt{state}', \texttt{image})$
- $\texttt{interframe} :=$
  $(\texttt{prediction\_modes}, \texttt{motion\_vectors}, \texttt{residue})$
- **encode-given-state**$(\texttt{state}, \texttt{image}, \textit{quality}) \rightarrow \texttt{interframe}$
- **rebase**$(\texttt{state}, \texttt{image}, \texttt{interframe}) \rightarrow \texttt{interframe}'$

Cold start vs. Warm start



Demo: Massively parallel face recognition on AWS Lambda

- ~**6 hours** of video taken on the first day of NSDI.
  - 1.4TB of uncompressed video uploaded to S3.
- Adapted *OpenFace* to run on AWS Lambda.
  - OpenFace: face recognition with deep neural networks.
- Running 2,000 *Lambda*s, looking for a face in the video.

The future is granular, interactive and massively parallel

- Parallel/distributed make
- Interactive Machine Learning
  - e.g. PyWren (Jonas et al.)
- Data Visualization
- Searching Large Datasets
- Optimization