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

Sadjad Fouladi[1], Riad S. Wahby[1], Brennan Shacklett[1],
Karthikeyan Vasuki Balasubramaniam[2], William Zeng[1],
Rahul Bhalerao[2], Anirudh Sivaraman[3], George Porter[2], Keith Winstein[1]

[1]*Stanford University,* [2]*UC San Diego,* [3]*MIT*

https://ex.camera

Outline

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

What we currently have

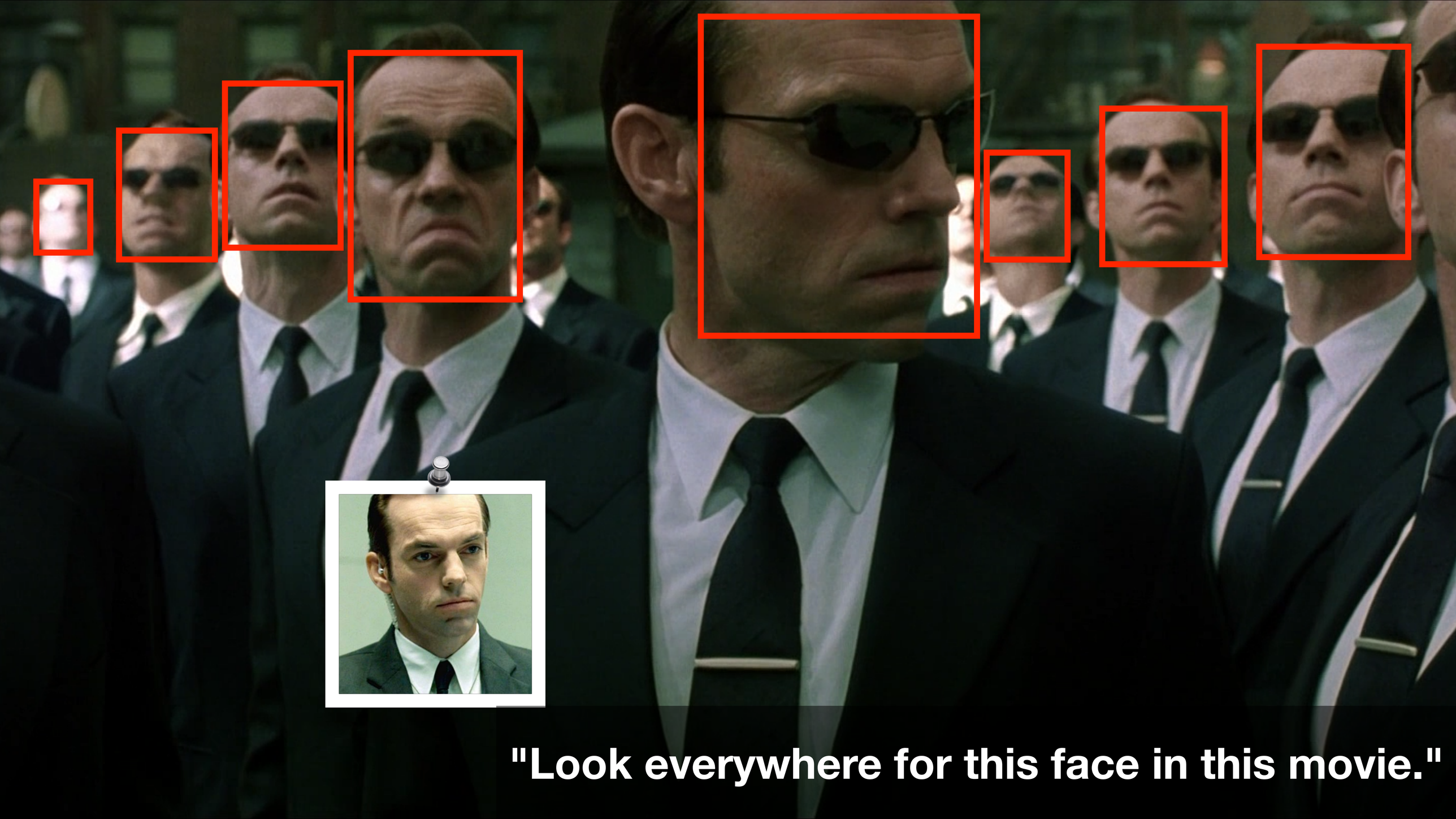
- People can make changes to a word-processing document
- The changes are instantly visible for the others

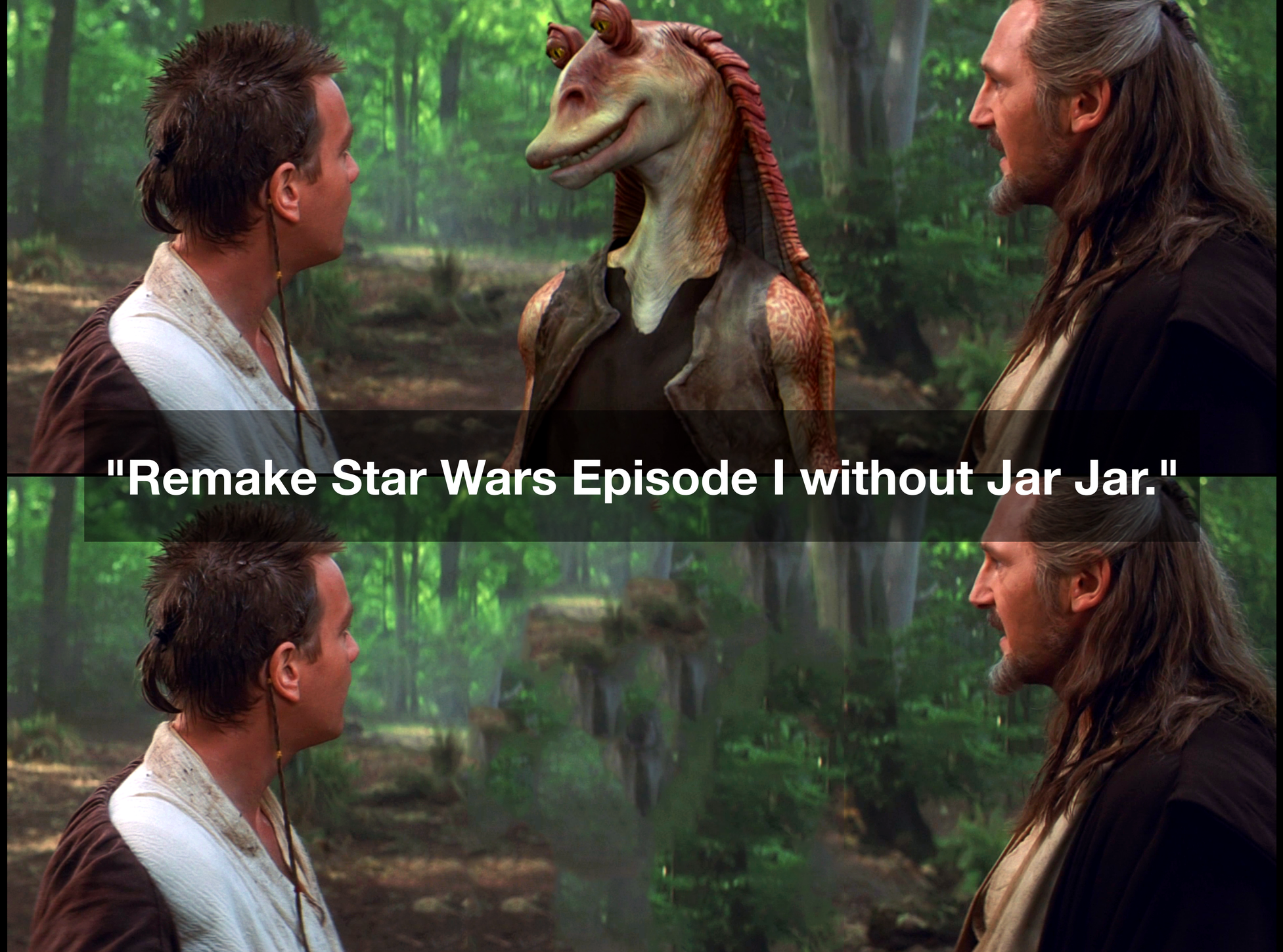
What we would like to have

Google Docs *for Video*?

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

"Apply this awesome filter to my video."


"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 compression efficiency.**

Enter *ExCamera*

- We made 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**.
- We call the whole system **ExCamera**.

Outline

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

Where to find thousands of threads?

- IaaS services provide virtual machines (e.g. EC2, Azure, GCE):
  - Thousands of threads
  - Arbitrary Linux executables
  - 👎 Minute-scale startup time (OS has to boot up, ...)
  - 👎 High minimum cost
    (60 mins EC2, 10 mins GCE)

3,600 threads on EC2 for one second → >$20

Cloud function services have (as yet) unrealized power

- AWS Lambda, Google Cloud Functions
- Intended for event handlers and Web microservices, *but...*
- Features:
  - ✔ Thousands of threads
  - ✔ Arbitrary Linux executables
  - ✔ Sub-second startup
  - ✔ Sub-second billing

3,600 threads for one second → 10¢

mu, supercomputing as a service

- We built *mu*, a library for designing and deploying general-purpose parallel computations on a commercial “cloud function” service.
- The system starts up thousands of threads in seconds and manages inter-thread communication.
- *mu* is open-source software: *https://github.com/excamera/mu*

HAVE YOU SEEN THIS MAN?

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.

Outline

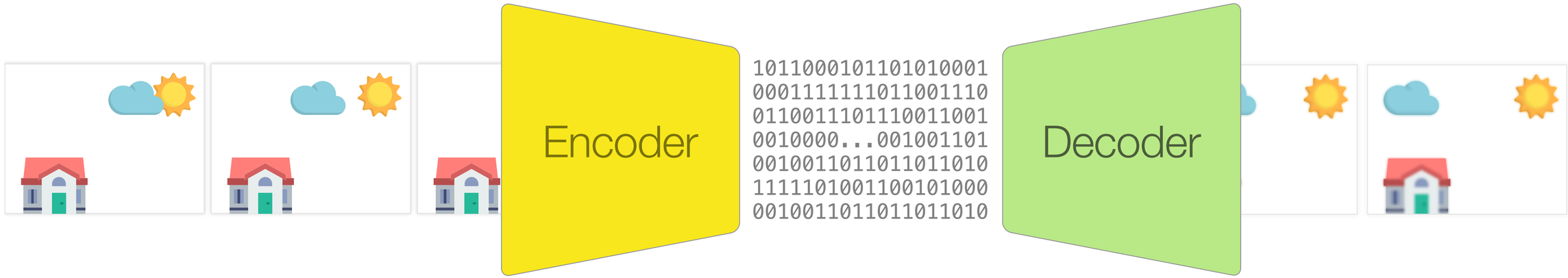
- Vision & Goals
- mu: Supercomputing as a Service
- Fine-grained Parallel Video Encoding
- Evaluation
- Conclusion & Future Work

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 digital video.



How video compression works

- Exploit the temporal redundancy in adjacent images.
- Store the first image on its entirety: a **key frame**.
- For other images, only store a "diff" with the previous images: an **interframe**.

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

Existing video codecs only expose a simple interface

compressed video

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

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

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

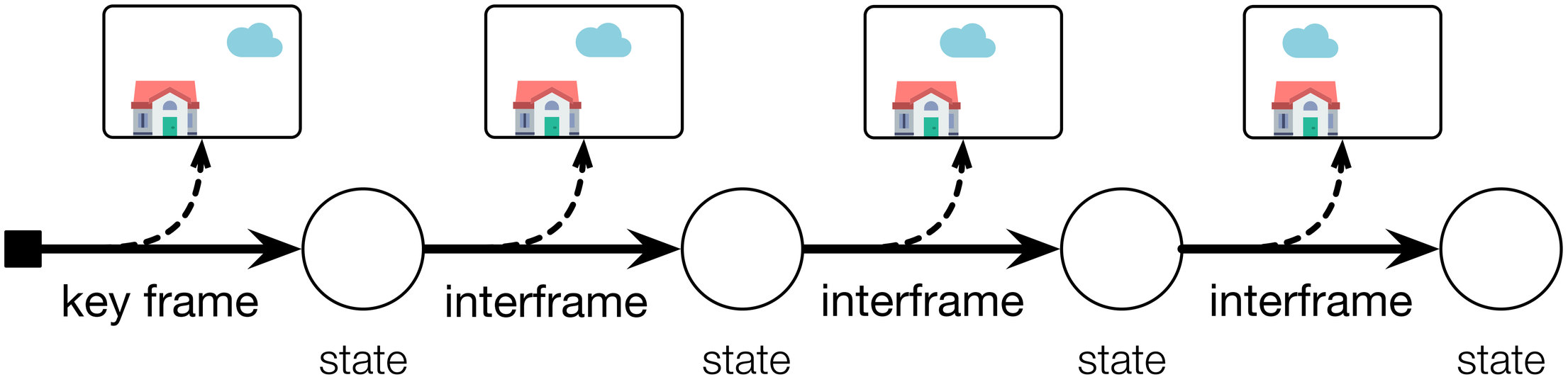
+1 MB

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

We need a way to start encoding mid-stream

- Start encoding mid-stream needs access to intermediate computations.
- Traditional video codecs *do not* expose this information.
- We formulated this internal information and we made it explicit: the **“state”**.

The decoder is an automaton



What we built: a video codec in 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`

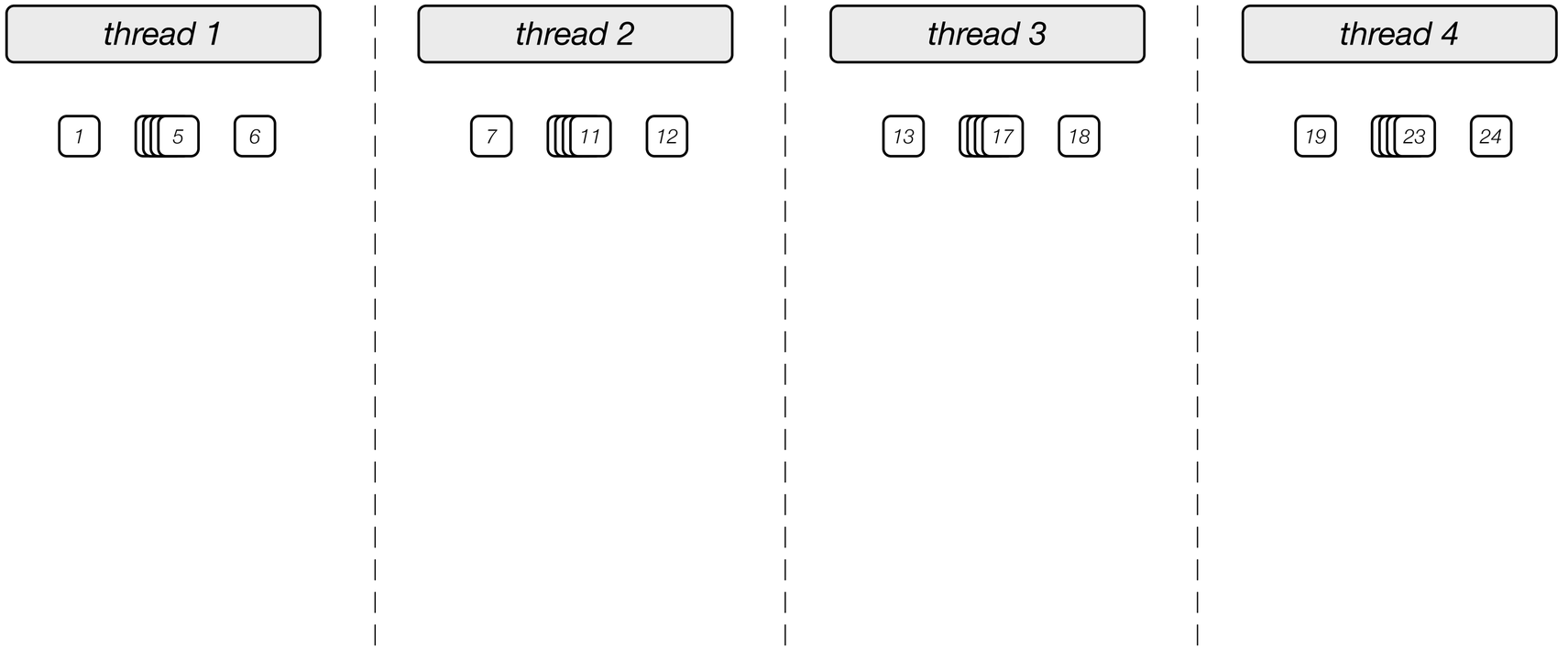
- Adapt a frame to a different source state

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

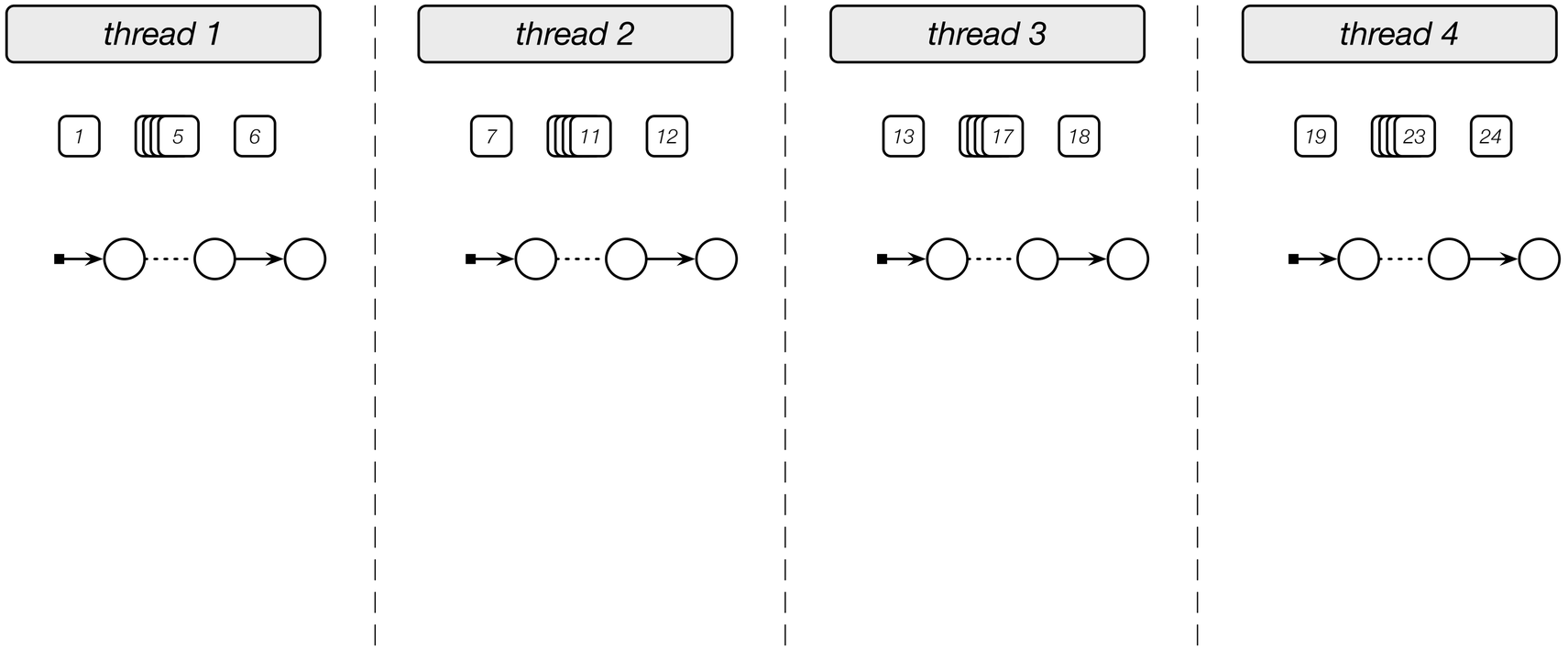
Putting it all together: ExCamera

- Divide the video into tiny chunks:
  - **[Parallel]** **`encode`** tiny independent chunks.
  - **[Serial]** **`rebase`** the chunks together and remove extra keyframes.

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



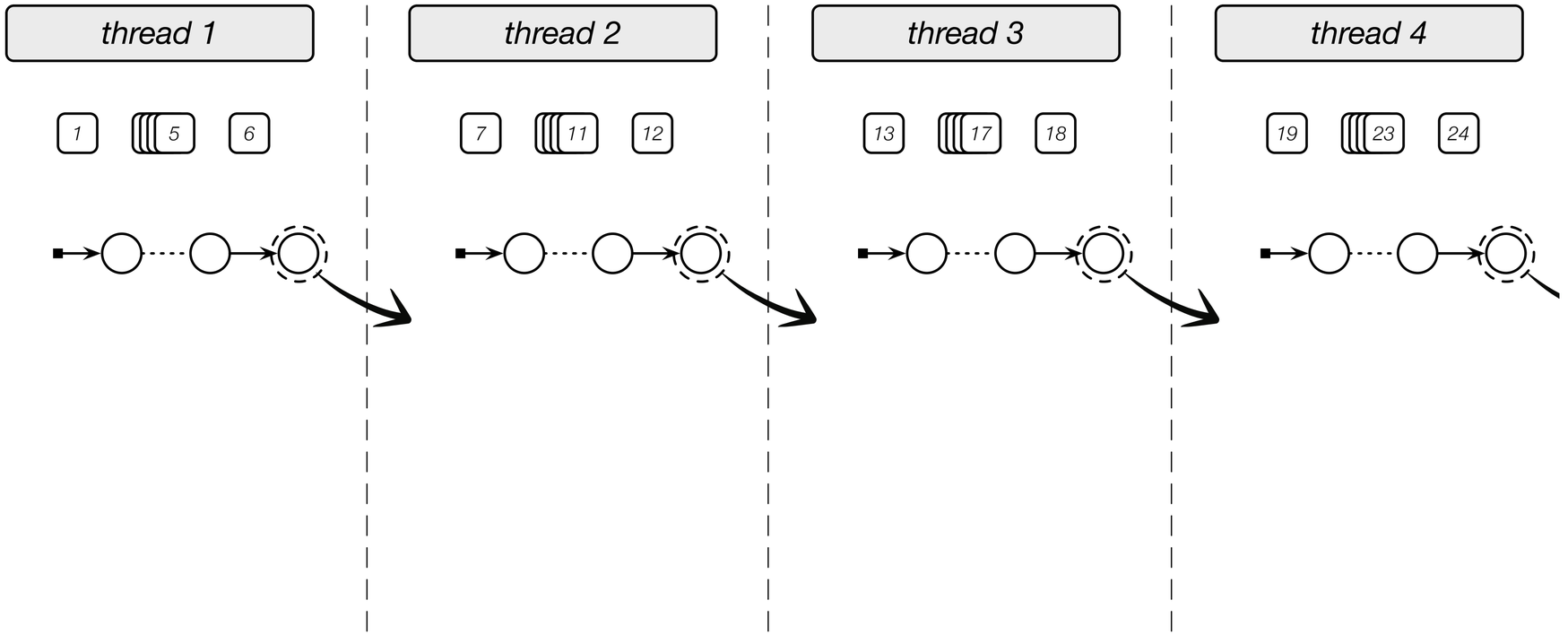
2. [Parallel] `vpxenc` → keyframe, interframe[2:n]



Google's VP8 encoder

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

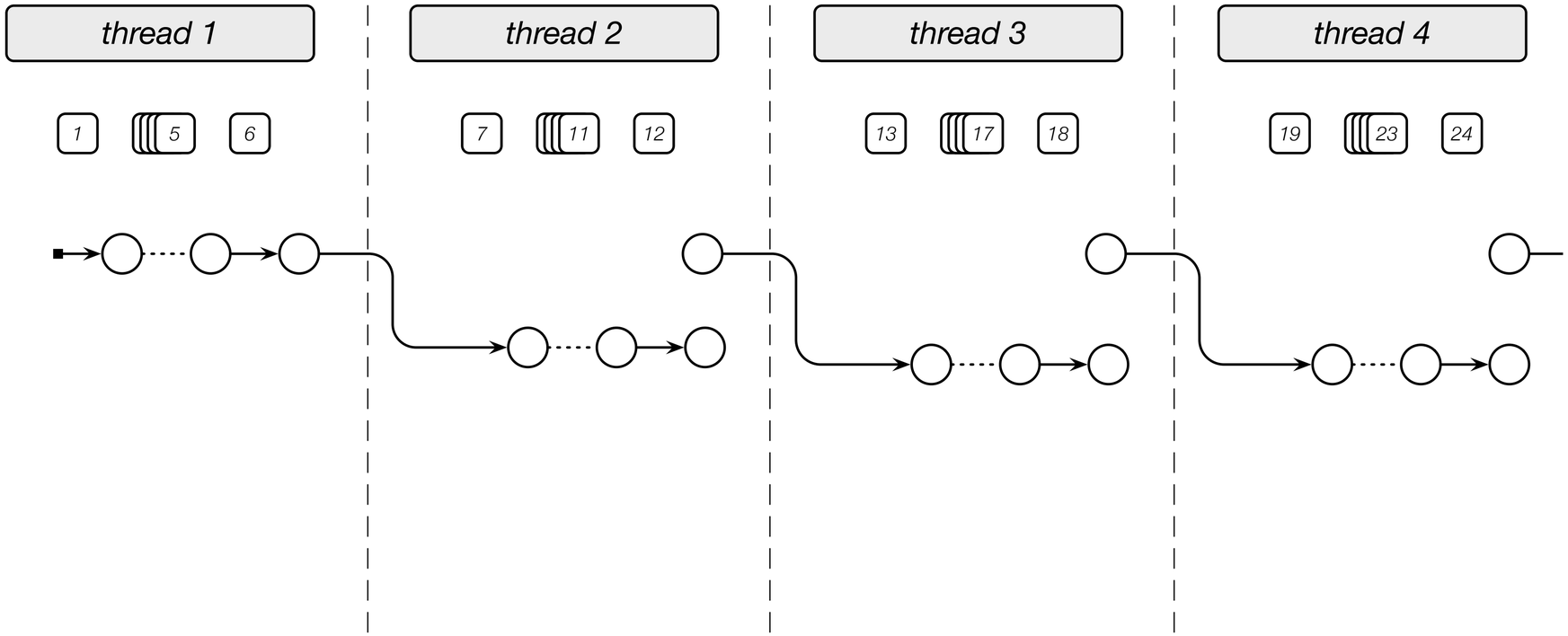
3. [Parallel] `decode` → state ⇝ *next thread*



Our explicit-state style decoder

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

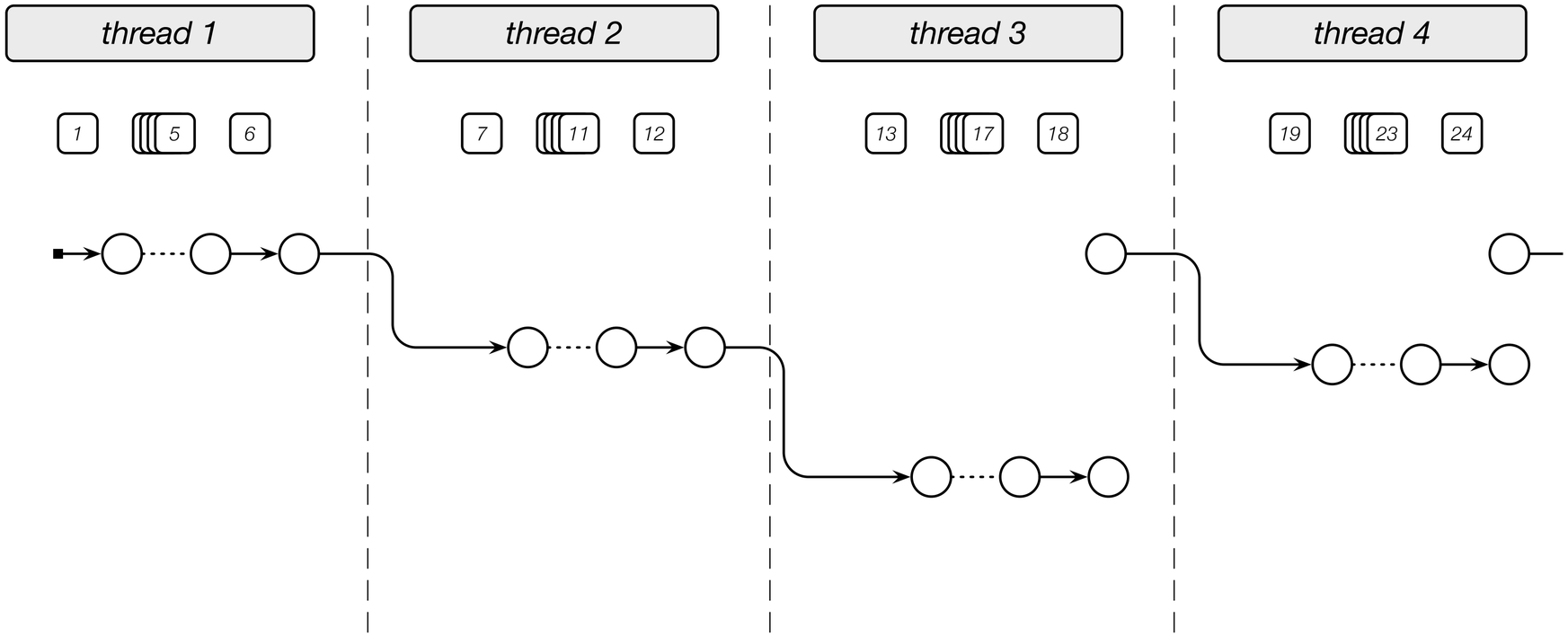
4. [Parallel] *last thread's state* ⇝ encode



Our explicit-state style encoder

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

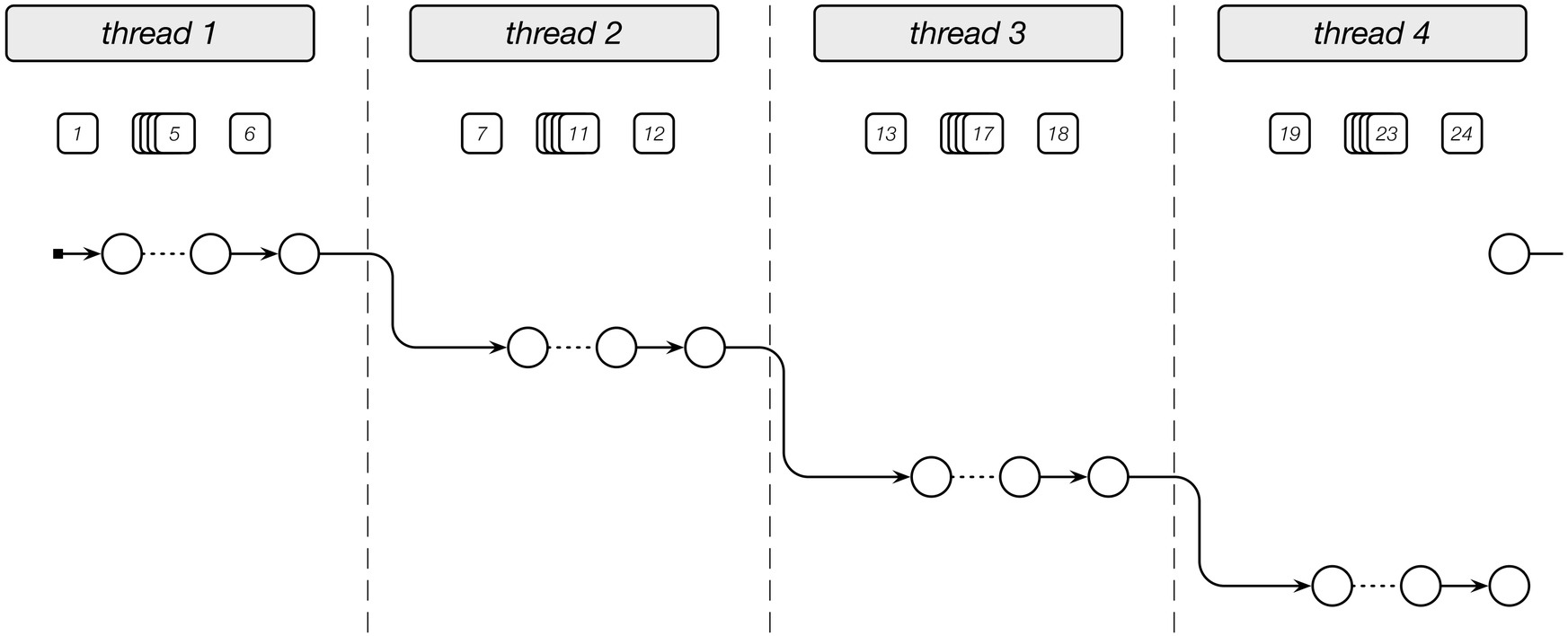
5. [Serial] *last thread's state* ⇝ `rebase` → state ⇝ *next thread*



Adapt a frame to a different source state

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

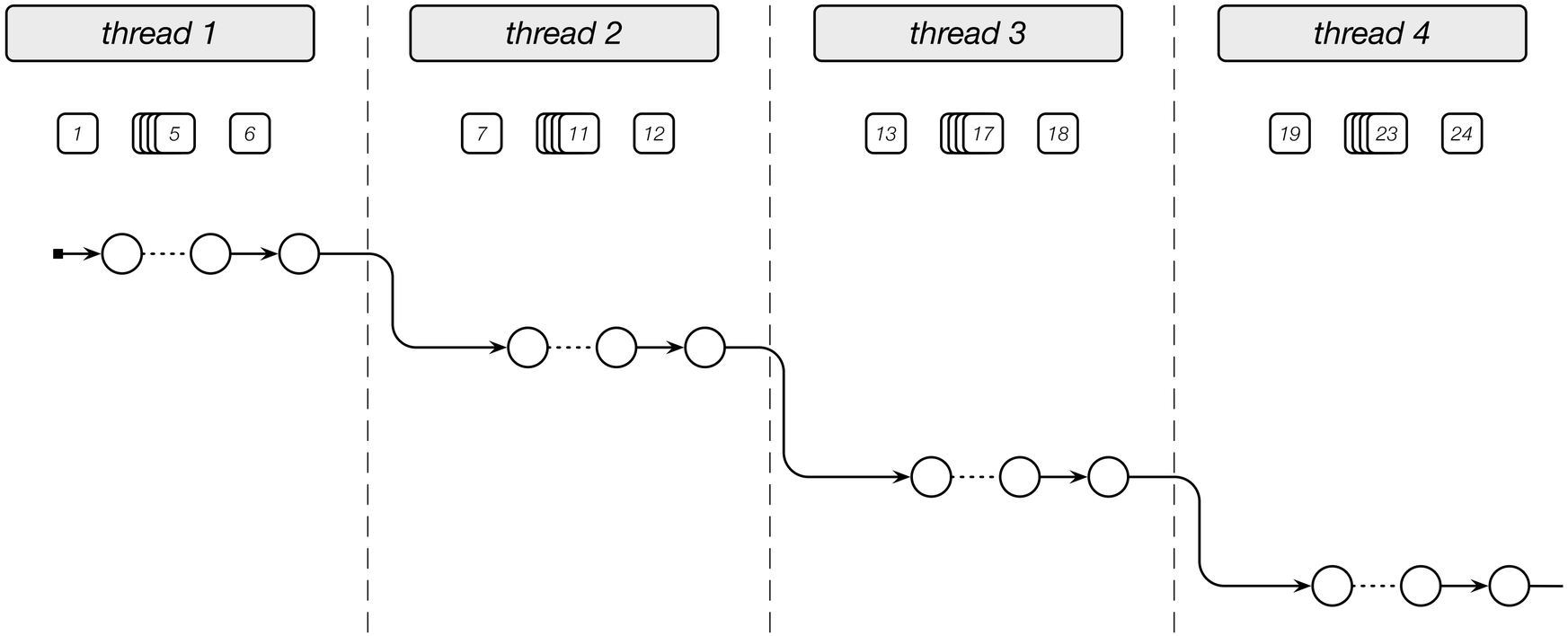
5. [Serial] *last thread's state* ⇝ `rebase` → state ⇝ *next thread*



Adapt a frame to a different source state

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

6. [Parallel] **Upload finished video**



Wide range of different configurations

```
ExCamera[n, x]
```

number of frames in each chunk

Wide range of different configurations

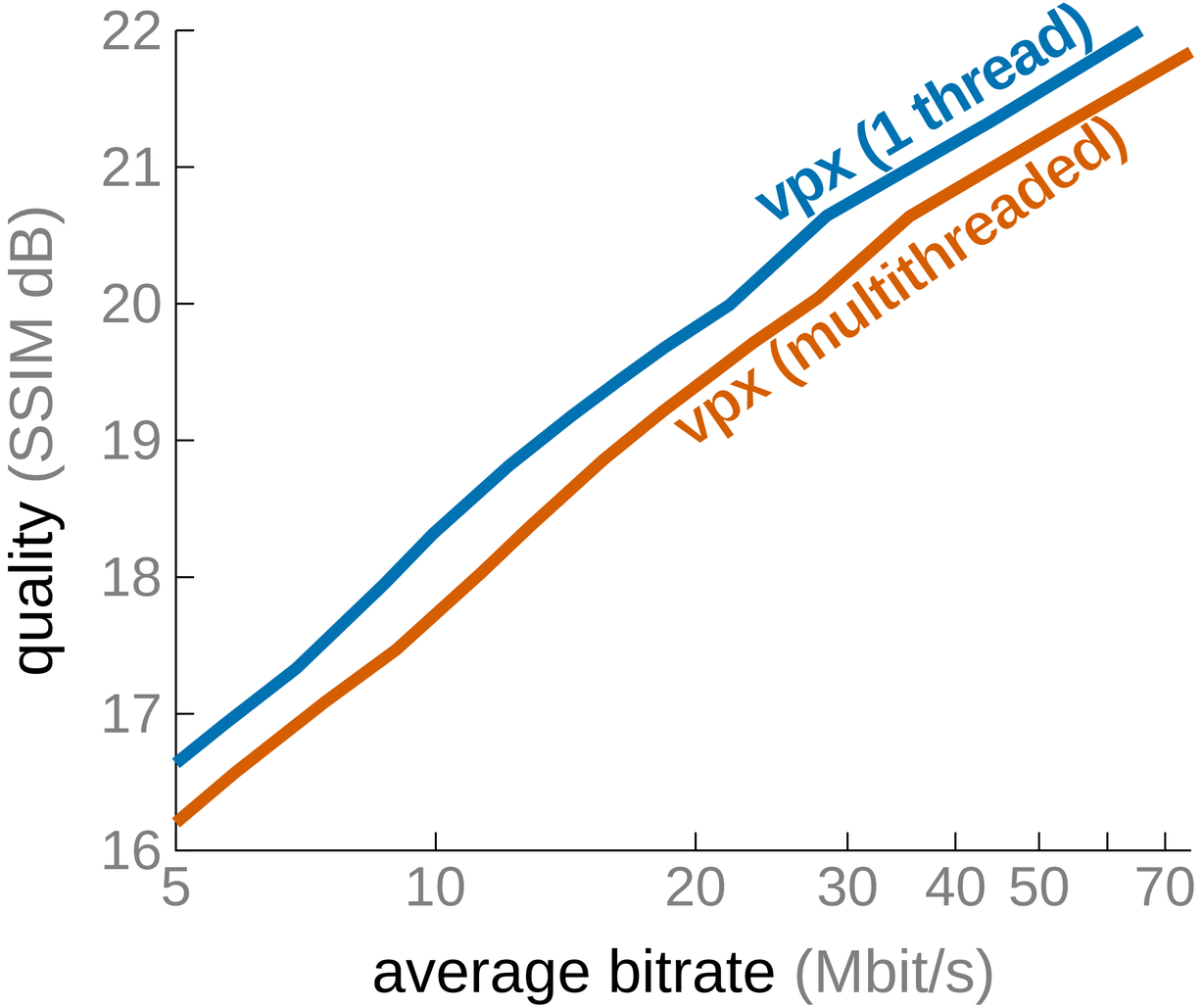
ExCamera[n, x]

number of chunks "rebased" together

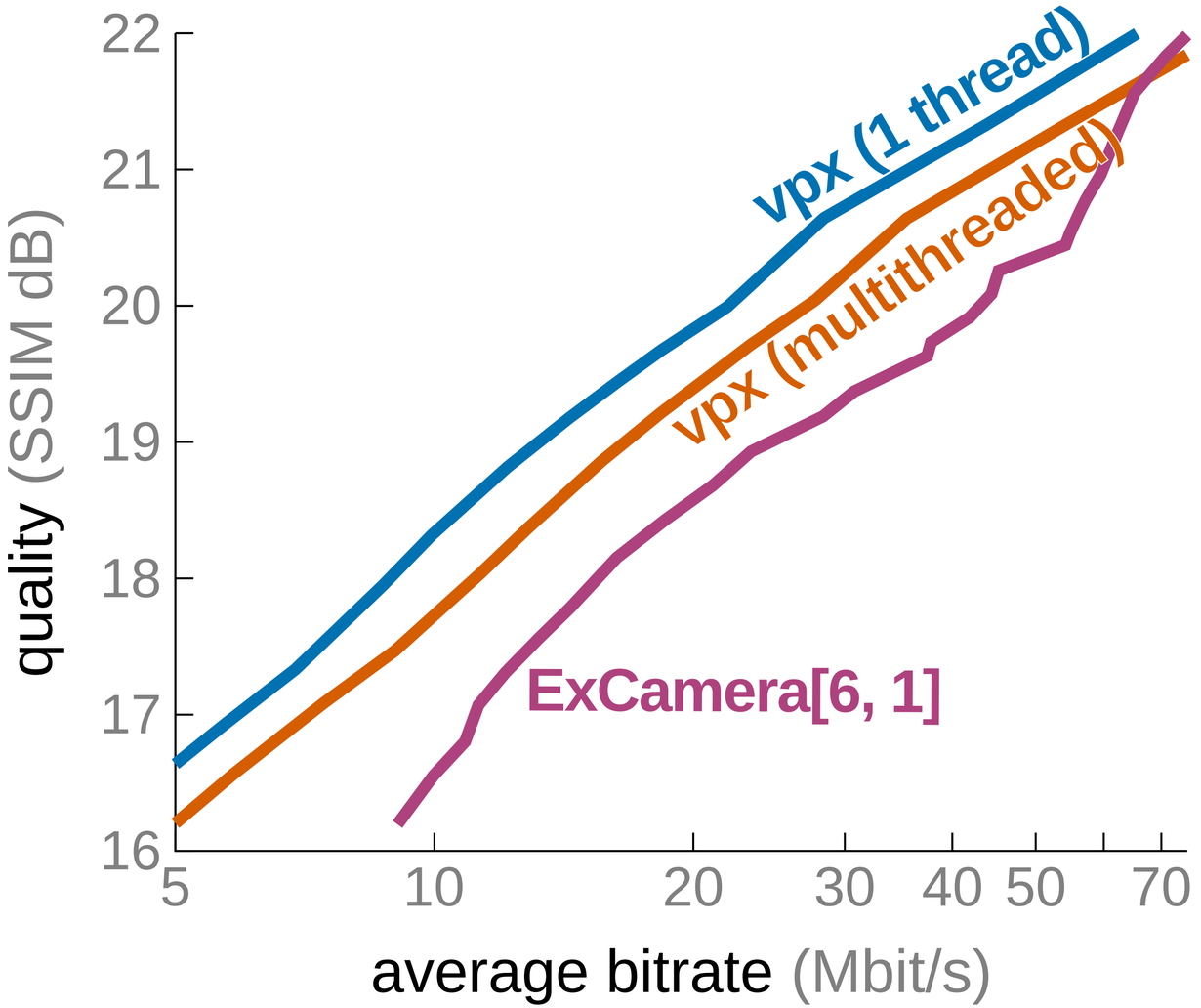
Outline

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

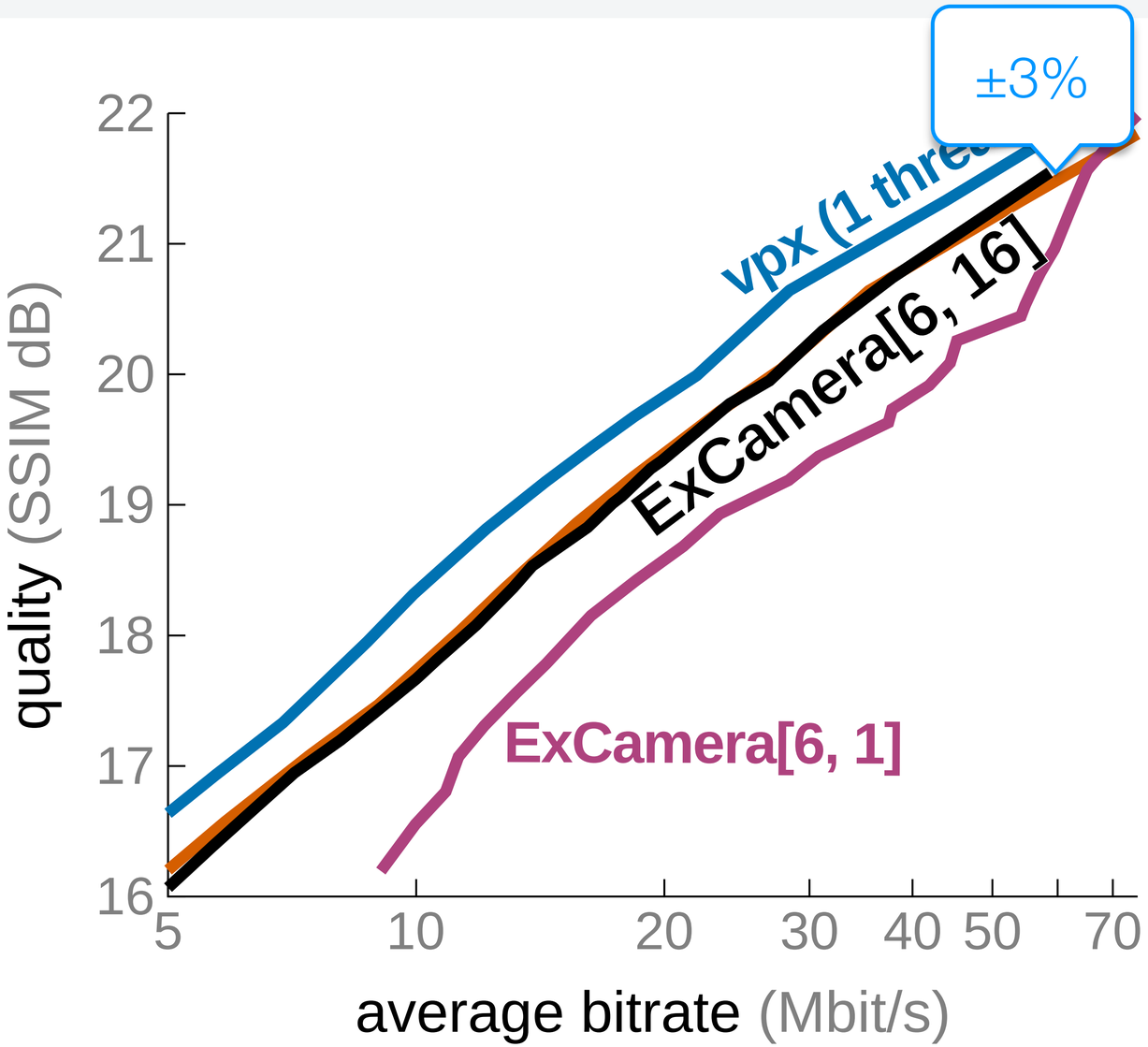
How well does it compress?



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**
YouTube (H.264)	**37 mins**
ExCamera[6, 16]	**2.6 mins**

Let's go back to the demo!

Outline

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

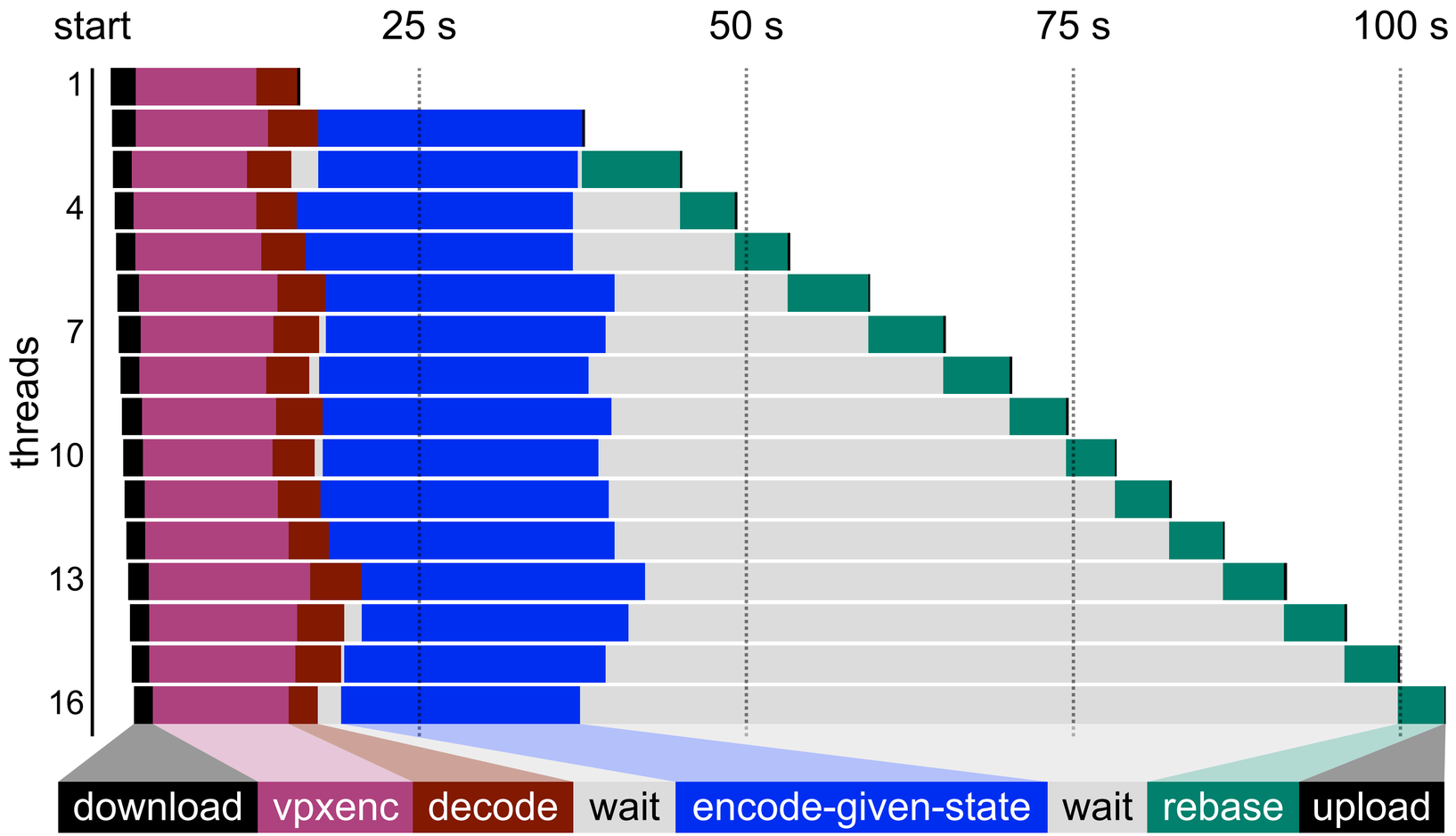
The future is granular, interactive and massively parallel

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

Takeaways

- Low-latency video processing
- Two major 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**.
- 56× faster than existing encoder, for <$6.

How time breaks down



Different ExCamera Configurations