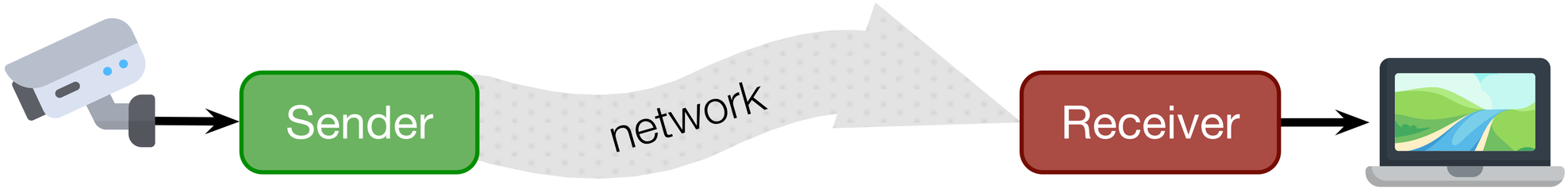
Salsify:
Low-Latency Network Video Through Tighter Integration Between a Video Codec and a Transport Protocol

Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, Keith Winstein

https://snr.stanford.edu/salsify

Stanford

Internet video delivery



Internet video delivery

- Different classes of Internet video delivery applications:
  - **Video streaming** (previous class)
  - Live video
  - **Real-time video** (today's class)

Each of these classes has a different latency target

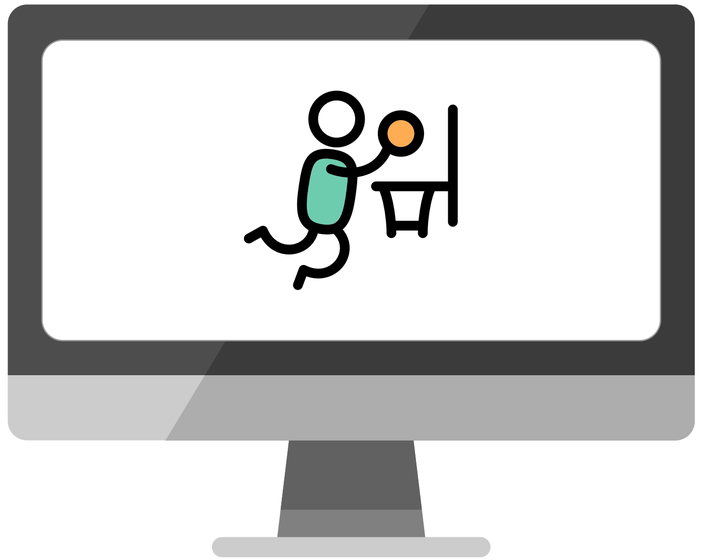
- Latency target: the amount of time between when

something happens



and

you see it



Class 1: Video streaming

- Latency target: **tens of minutes**
- 

- **BBA** [T.-Y. Huang et al., SIGCOMM ’14]
  MPC [X. Yin et al., SIGCOMM ’15]
  Pensieve [H. Mao et al., SIGCOMM ’17]
  Oboe [Z. Akhtar, SIGCOMM ’18]

Class 2: Live video

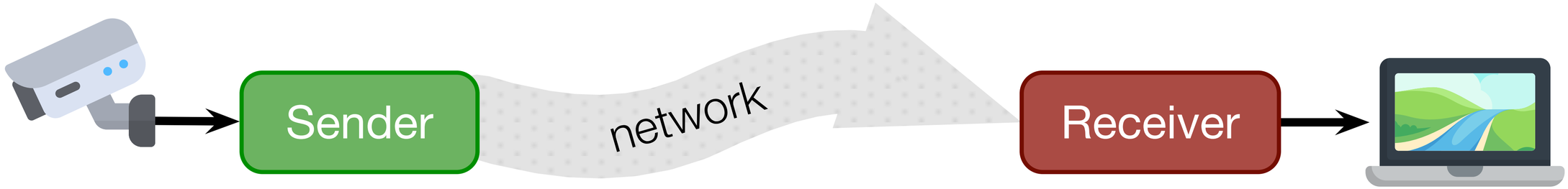
- Latency target: **5—30 seconds**
- 

- **Vic** [S. McCanne et al., MULTIMEDIA ’95]
  VDN [M. K. Mukerjee et al., SIGCOMM ’15]

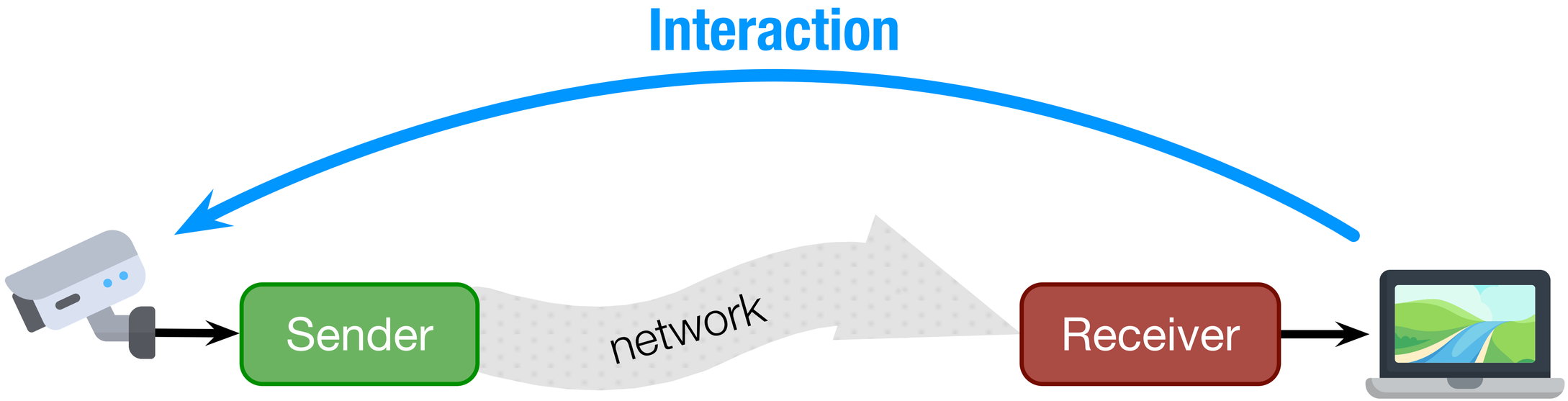
Class 3: Real-time Video

- Latency target: **tens of milliseconds**
- Why would an application need that?

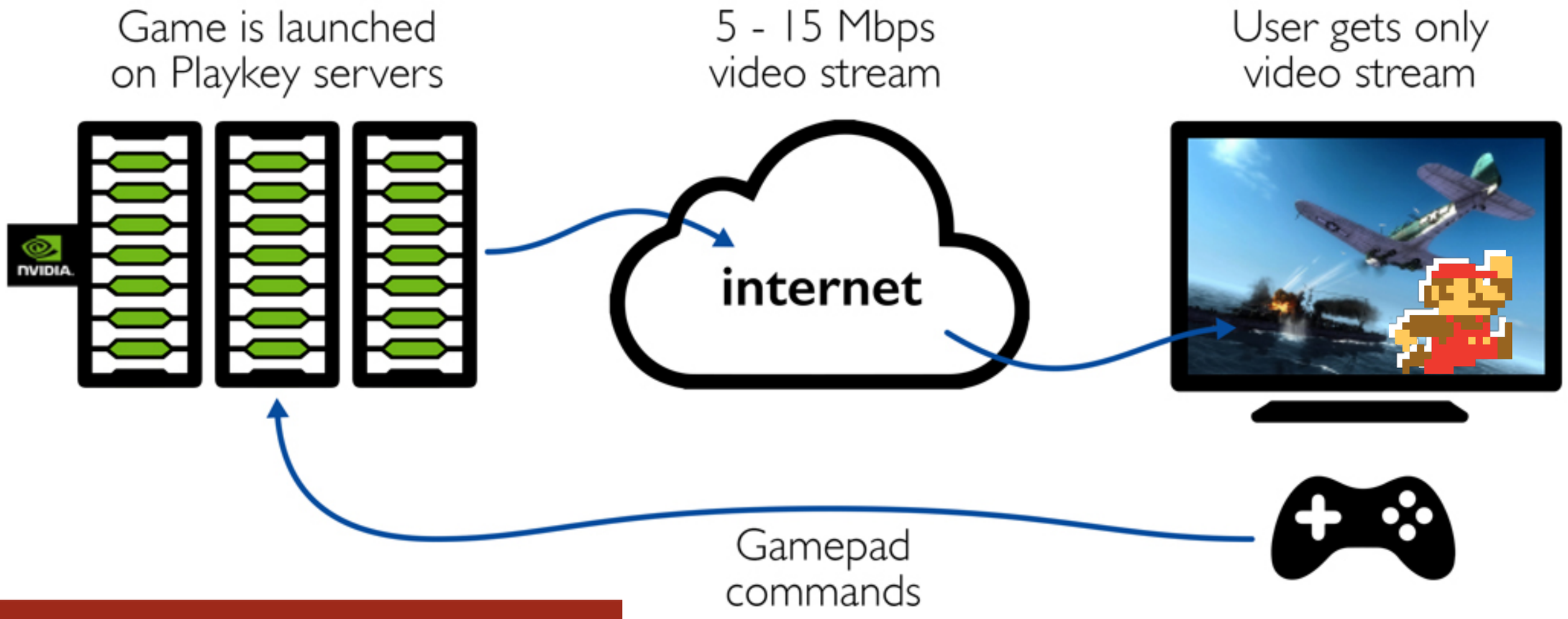
Real-time video systems transmit video with low latency…



…to maintain the interactivity of the application.



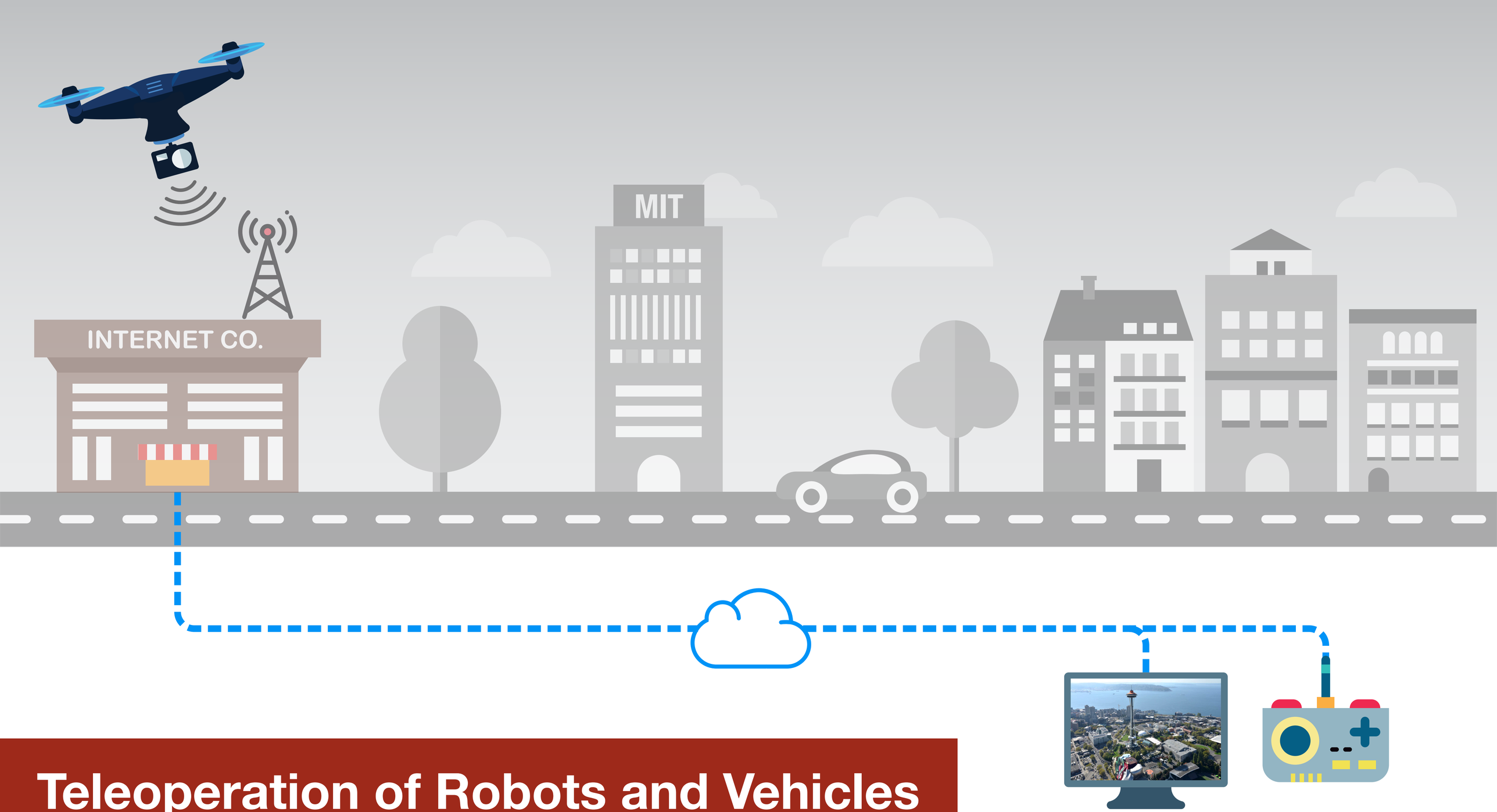
How cloud gaming works



Cloud Video Gaming

Remote Surgery





Teleoperation of Robots and Vehicles

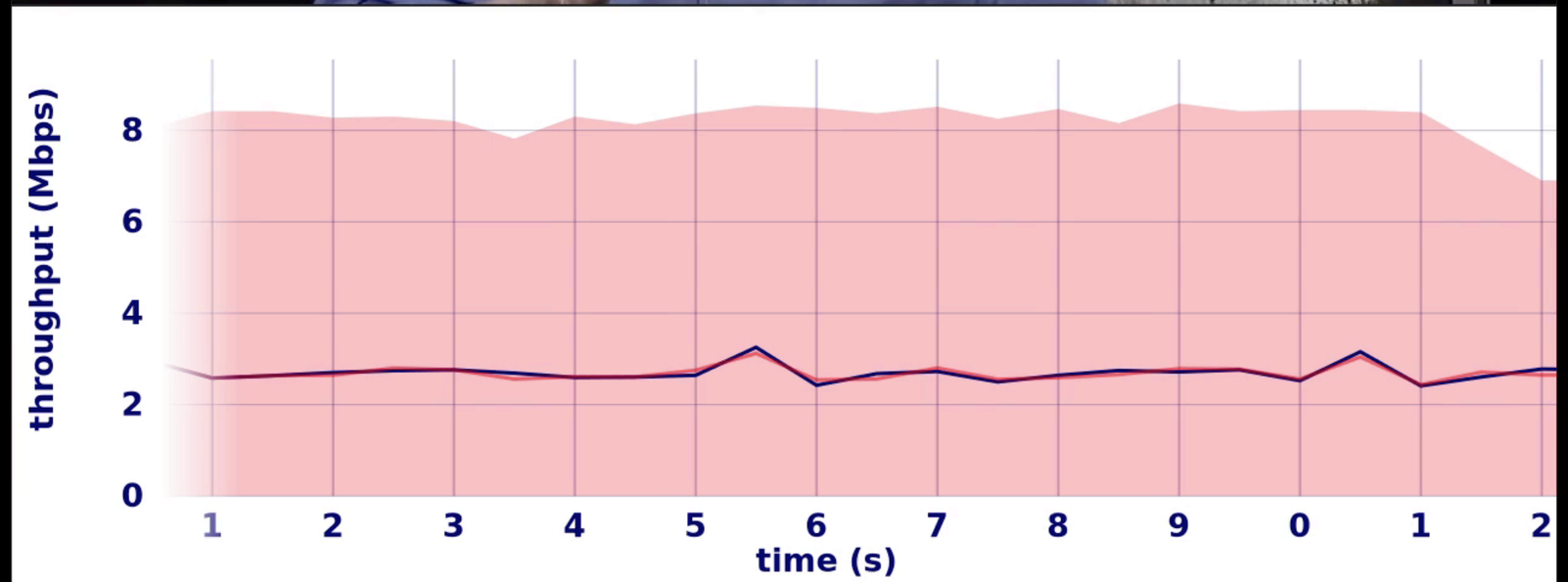

Sarah, Carol
3 in call | 00:04
Video Conferencing


Video Conferencing


Video Conferencing (reality)



Watch the video at: https://snr.stanford.edu/salsify



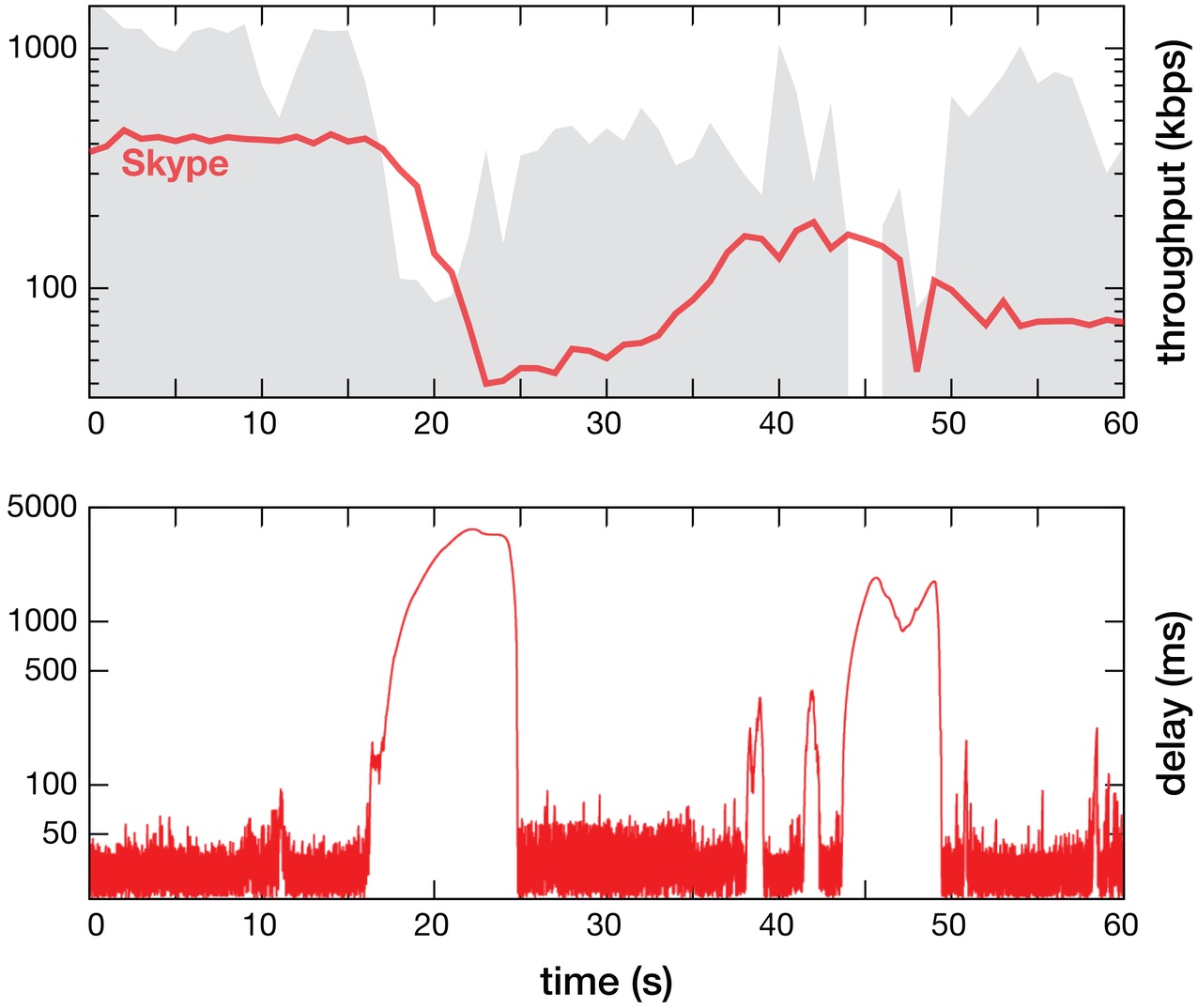
WebRTC
(Chrome 65)

Current systems do not react **fast enough** to **network variations**, end up congesting the network, causing **stalls and glitches**.



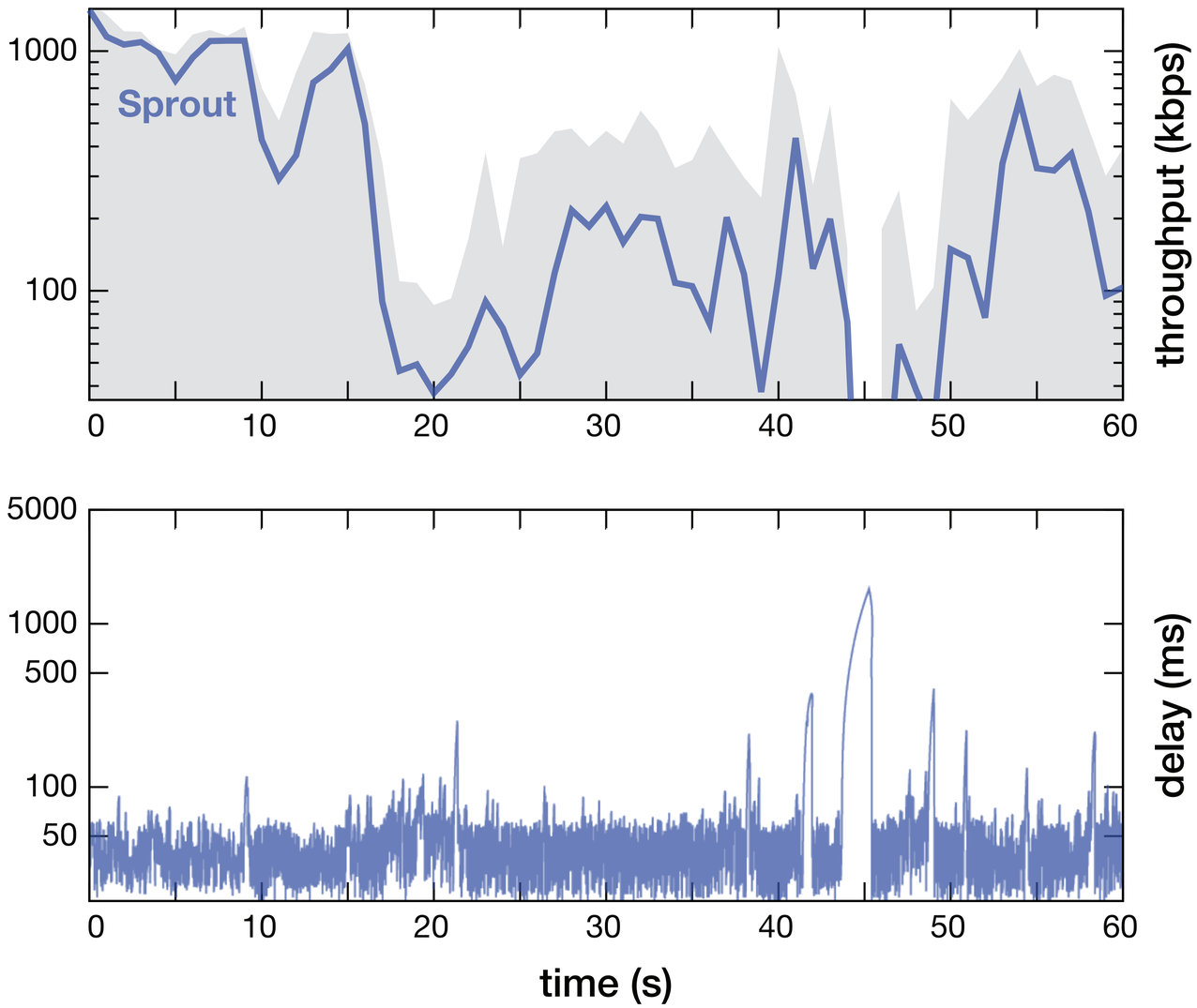
Researchers already knew about this…

K. Winstein, A. Sivaraman, H. Balakrishnan, ***Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks***, NSDI ’13



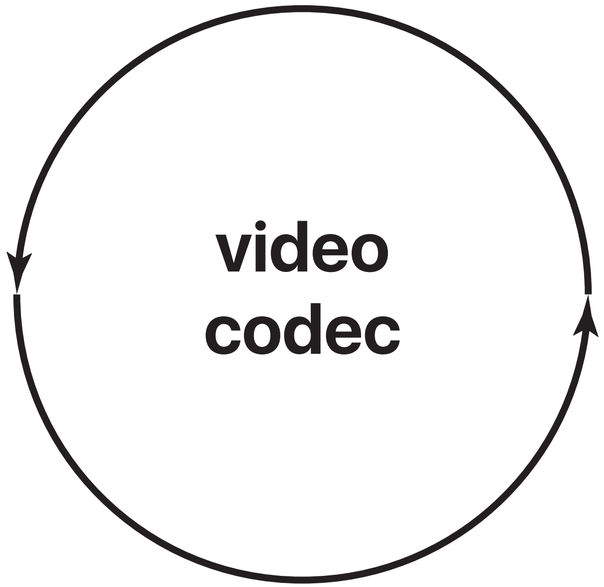
…and designed better throughput prediction algorithms

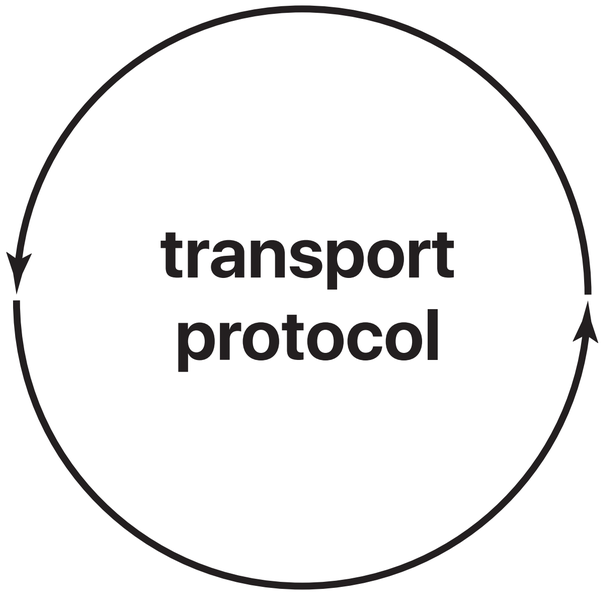
K. Winstein, A. Sivaraman, H. Balakrishnan, ***Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks***, NSDI ’13



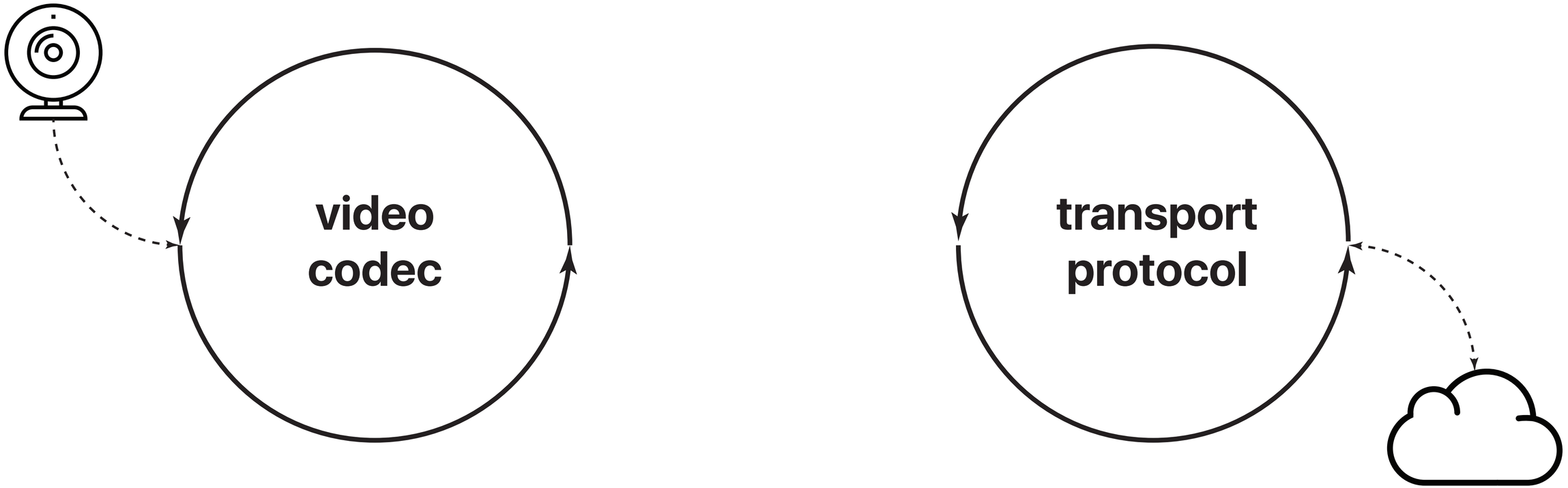
But better congestion control alone couldn’t save the day.

Conventional design: two control loops at arm’s length

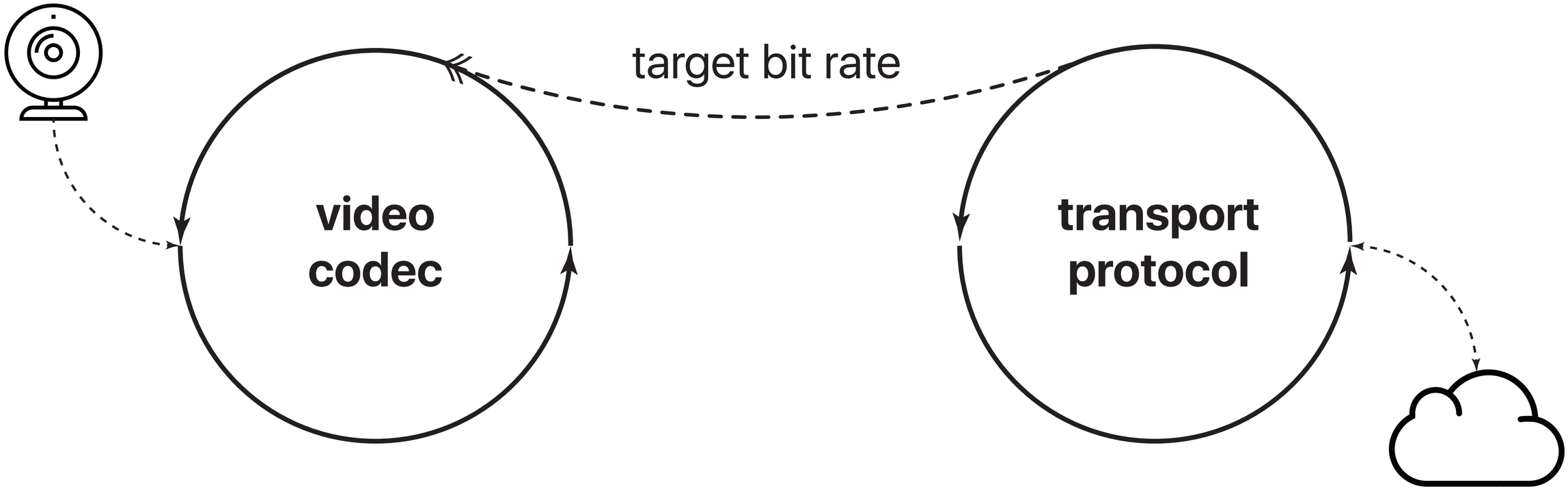




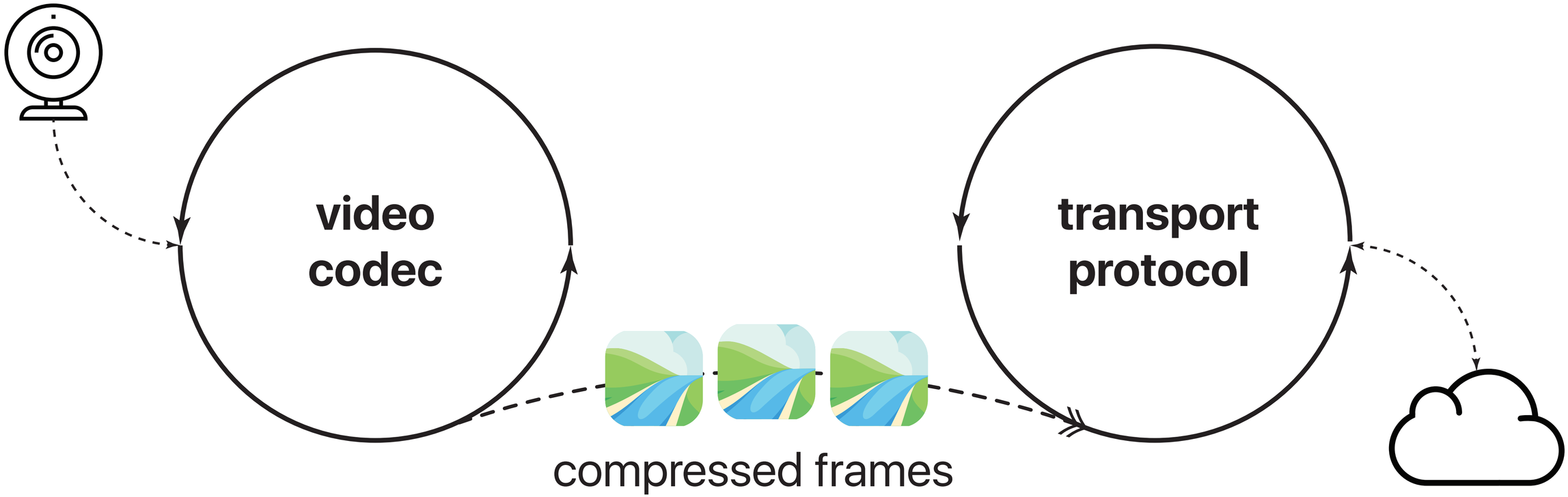
Conventional design: two control loops at arm's length



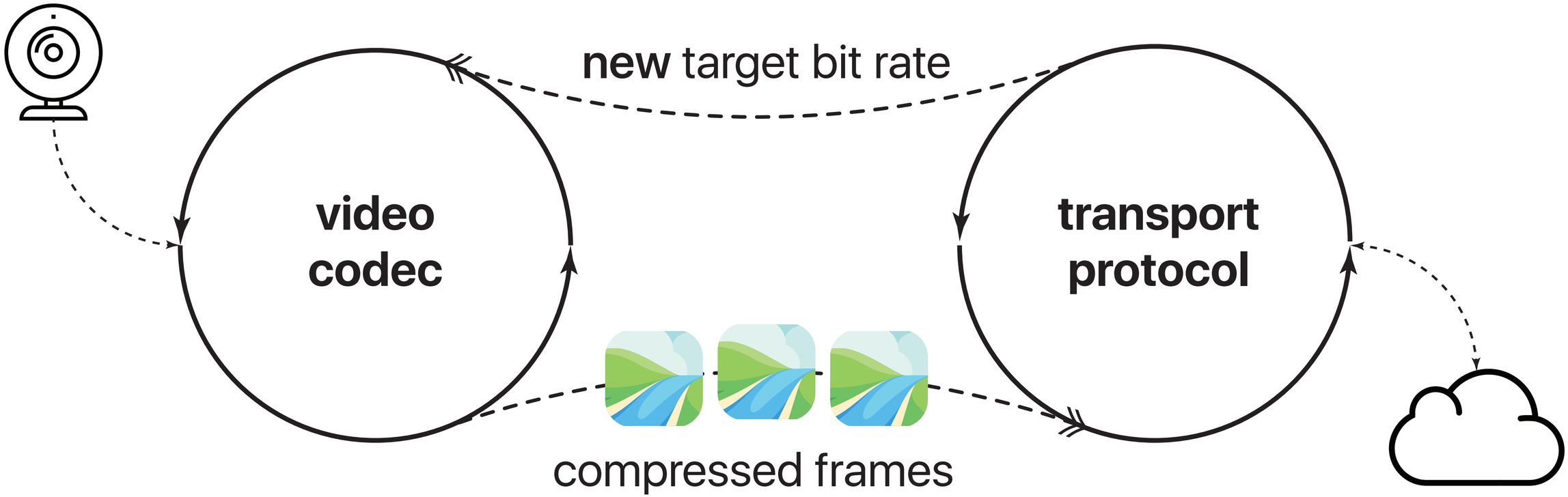
Transport estimates the network throughput and communicates that to the codec



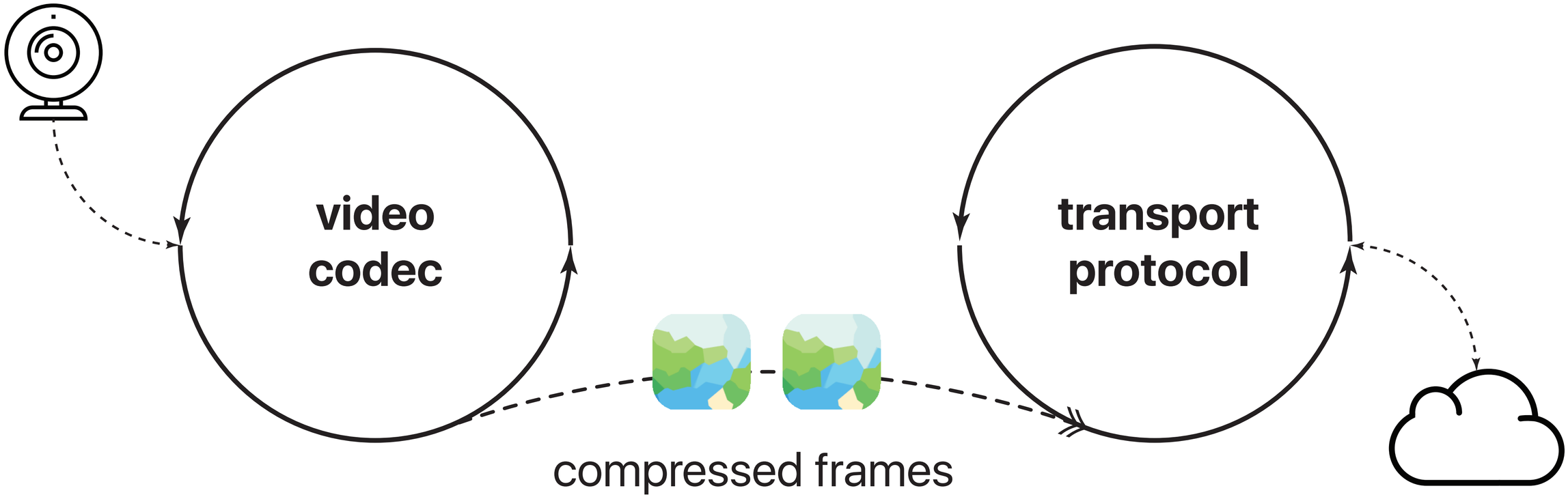
Codec produces compressed frames, targeting that bit rate



Transport occasionally updates that estimate



Codec updates frame rate and quality accordingly

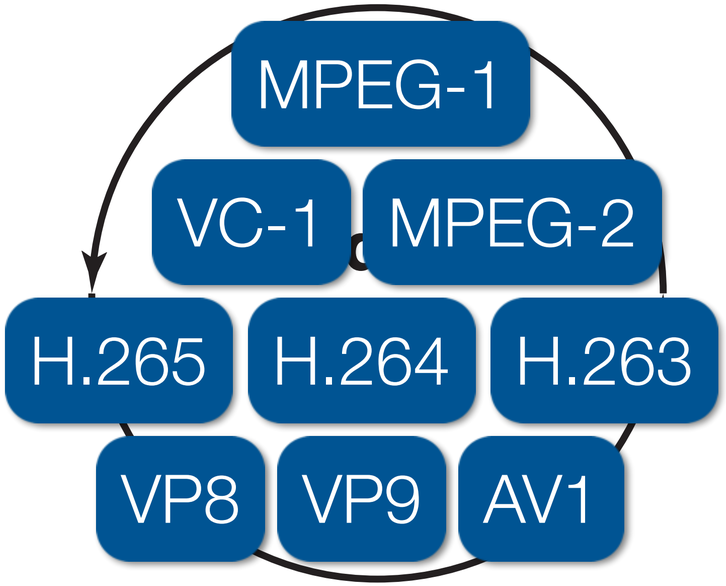


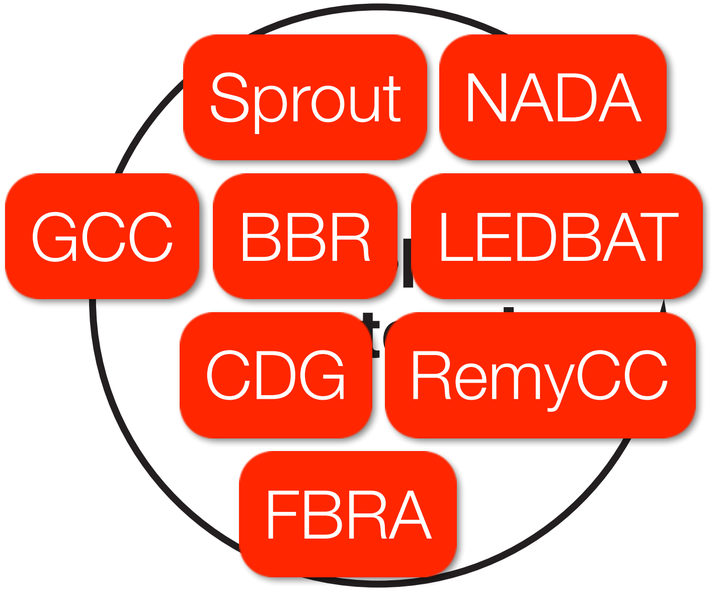
The problem: codec and transport are *too* decoupled

- The codec can only respond to changes in target bit rate over ***coarse time intervals***.
  - Individual frames may cause packet loss/queueing.
- The transport has little control over what codec produces.

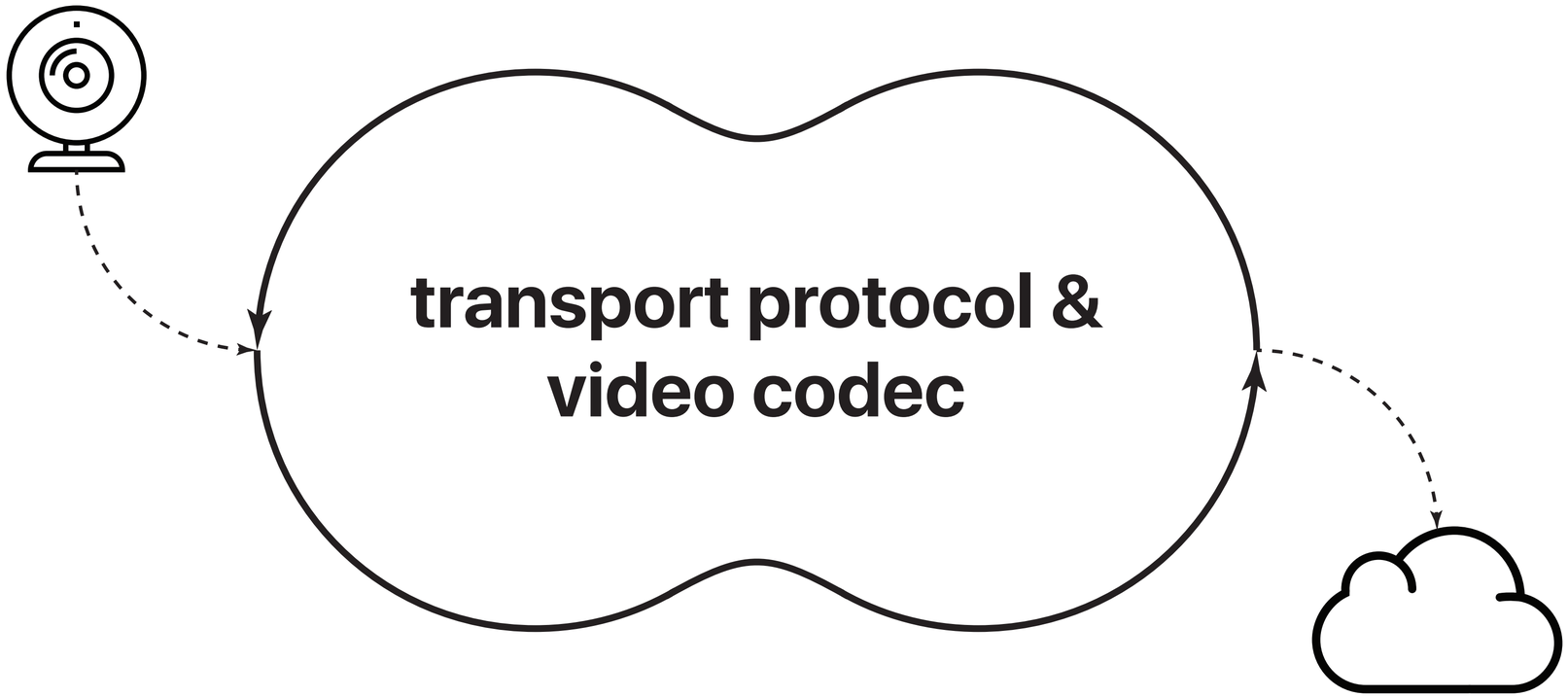
⇒ The resulting system is slow to react to network variations.

Decades of research and development on these components…





Salsify explores a more tightly-integrated design

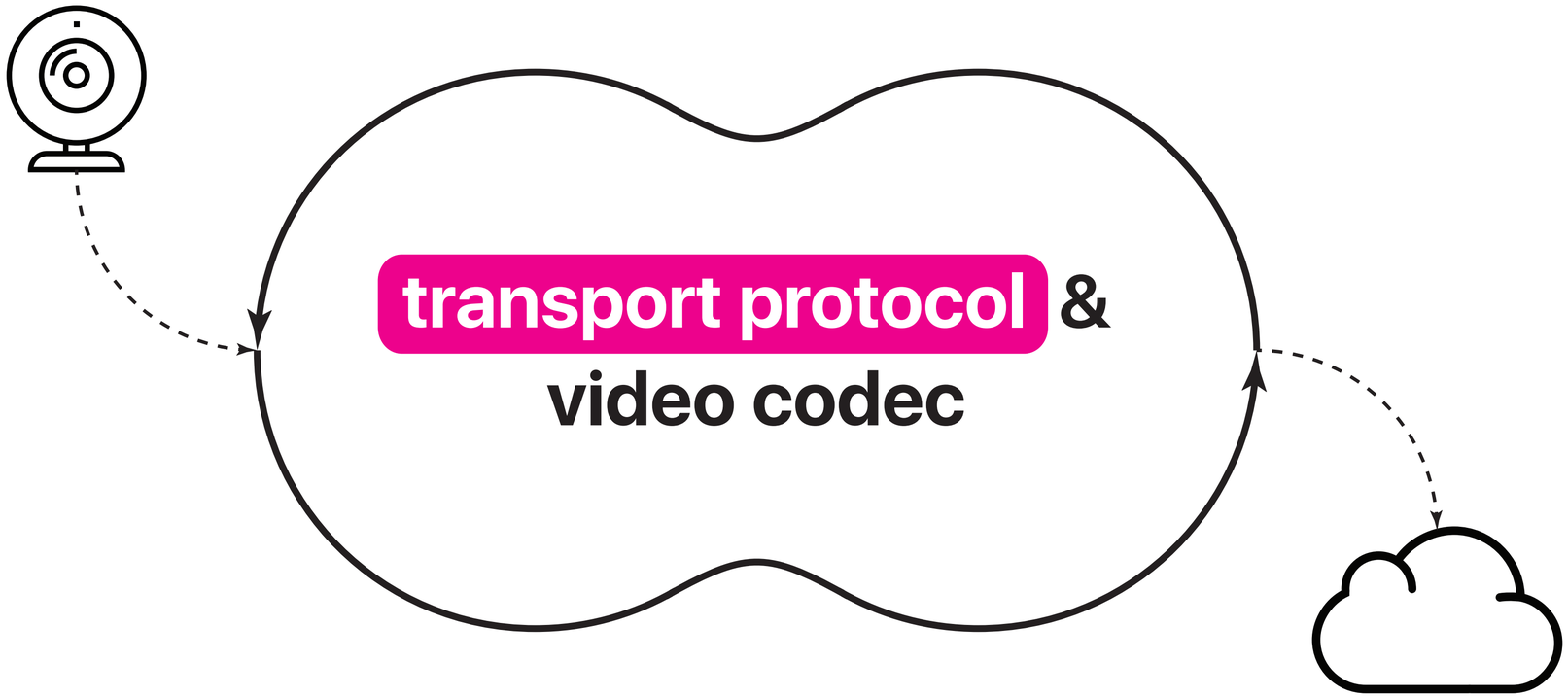


Salsify, a new heart from old parts

- Individual component of Salsify are not exactly new:
  - The transport protocol is inspired by “packet pair” and “Sprout-EWMA”.
  - The video format, VP8, was finalized in 2008.
  - The functional video codec was introduced in [Fouladi et al., NSDI ’17].
- Salsify is a **new architecture** for real-time video that integrates these components in a way that responds quickly to network variations.

Salsify's architecture:

Video-aware transport protocol



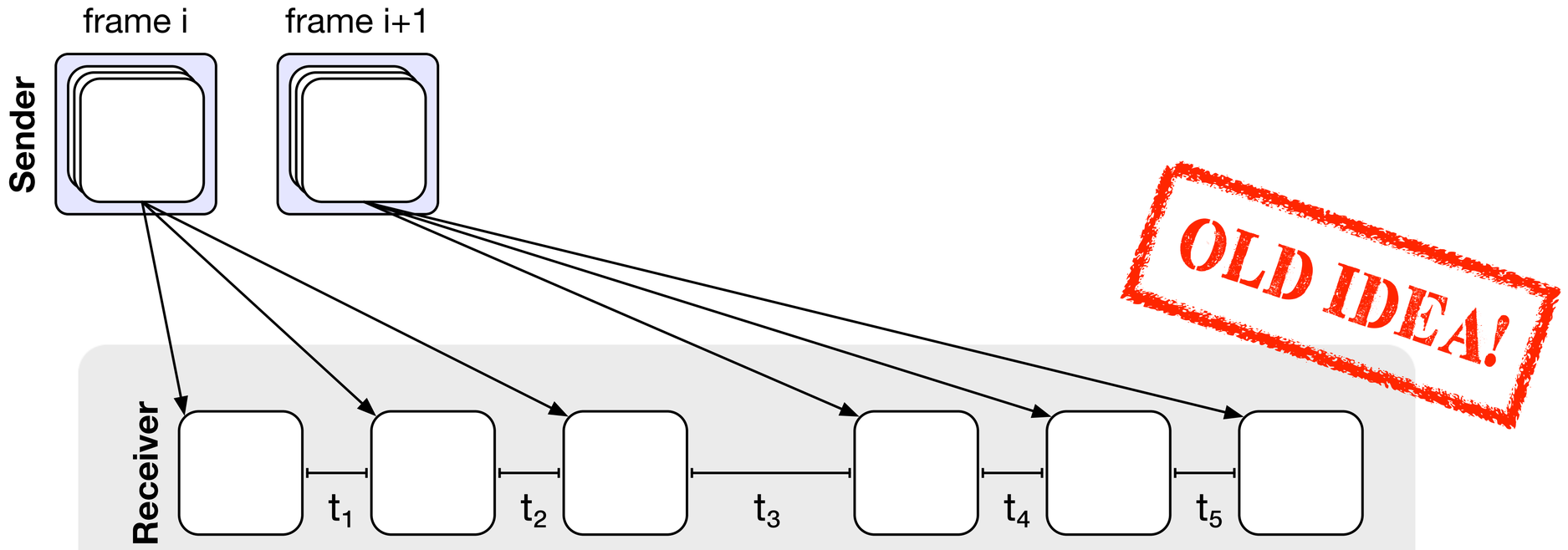
Video-aware transport protocol

“What should be the size of the next frame?”

∗ without causing excessive delay

- There’s no notion of bit rate, only the next frame size!
- Inspired by *packet pair* and *Sprout-EWMA*, transport uses **packet inter-arrival time**, reported by the receiver.

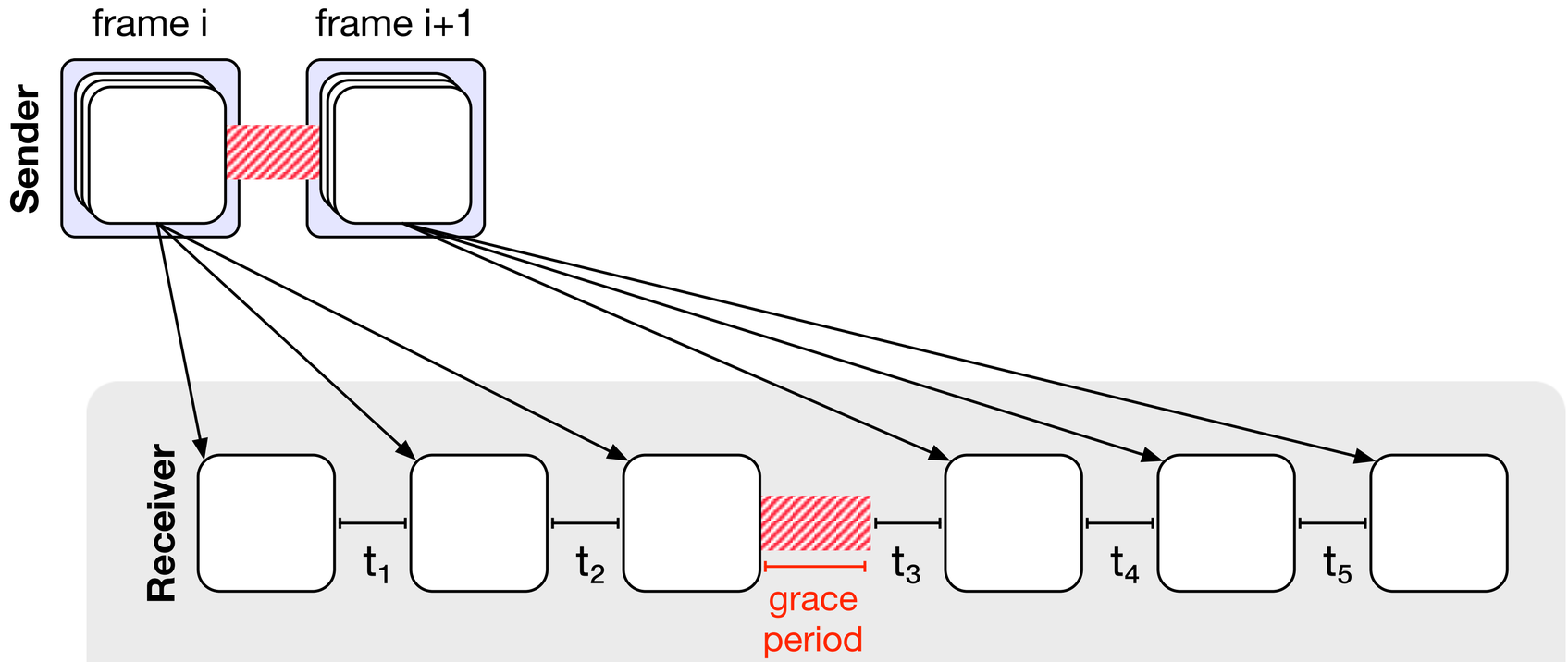
Receiver keeps a moving average of packet inter-arrival times



$$T \leftarrow \alpha \cdot t_i + (1 - \alpha) \cdot T$$

T : average packet inter-arrival time

Sender does not transmit continuously



$$T \leftarrow \alpha \cdot (t_i - g_i) + (1 - \alpha) \cdot T$$

What should be the size of the next frame?

- The receiver sends back an acknowledgement for every received packet, containing the latest inter-arrival time.

- The sender calculates the next frame size as follows:

T : average inter-arrival time

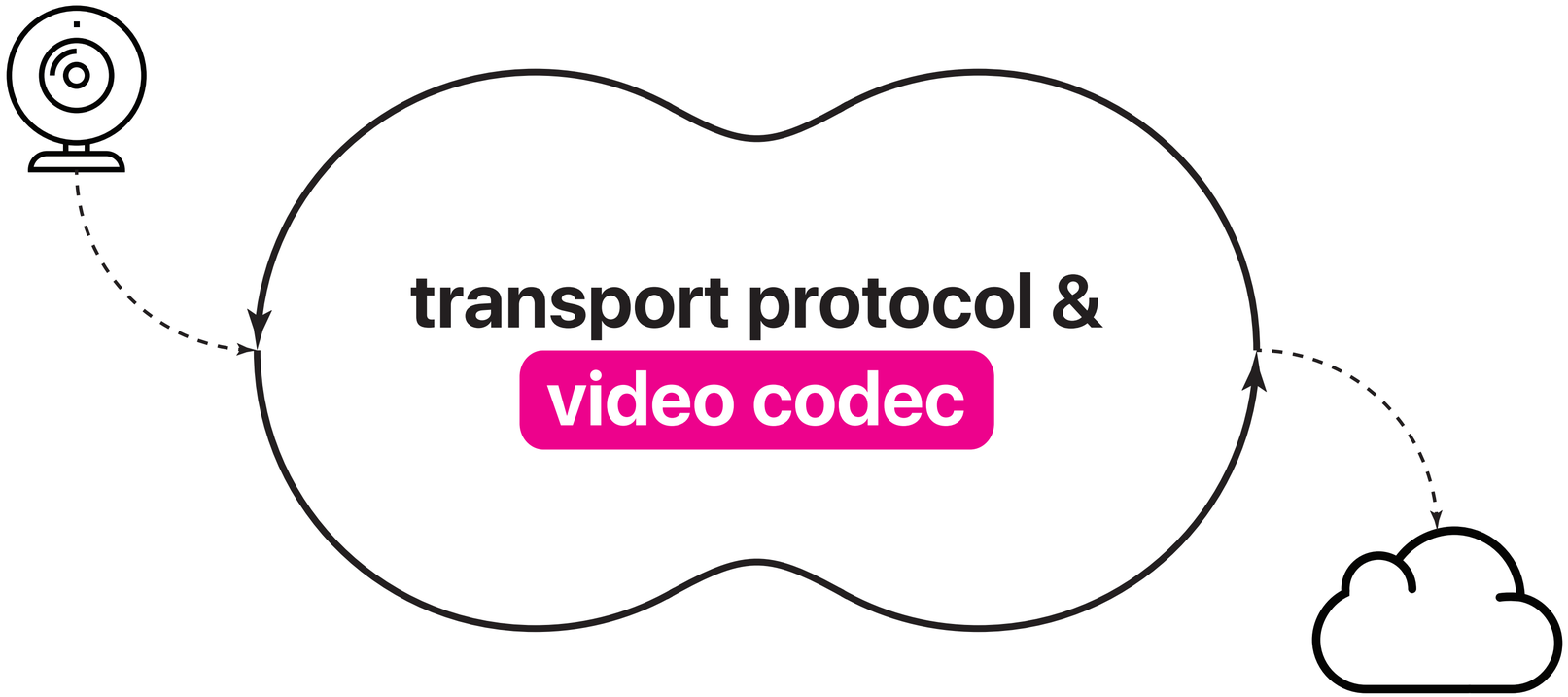
d : target delay

N : number of packets in flights

$$\text{frame size} = \left(\frac{d}{T} - N\right) \text{ packets}$$

Salsify's architecture:

Functional video codec

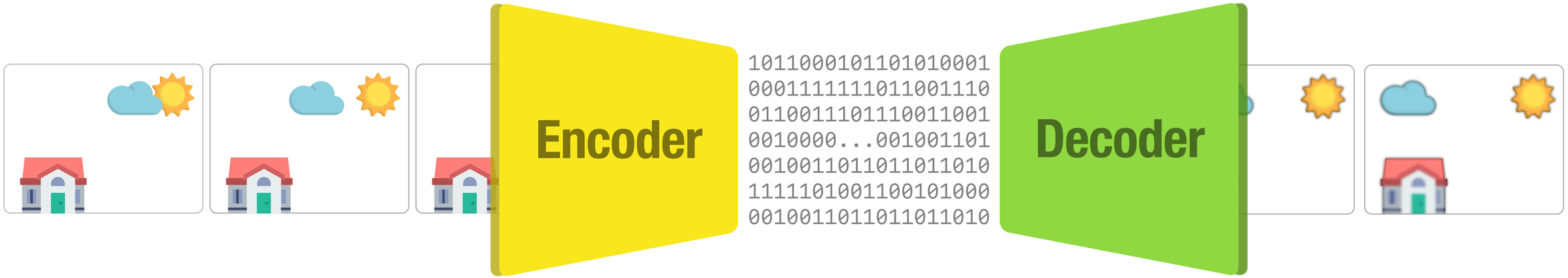


The encoder can only know the output size *after the fact.*

It’s challenging for **any codec** to choose the appropriate quality settings upfront to meet a **target size**—they tend to over-/undershoot the target.

Video codec

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

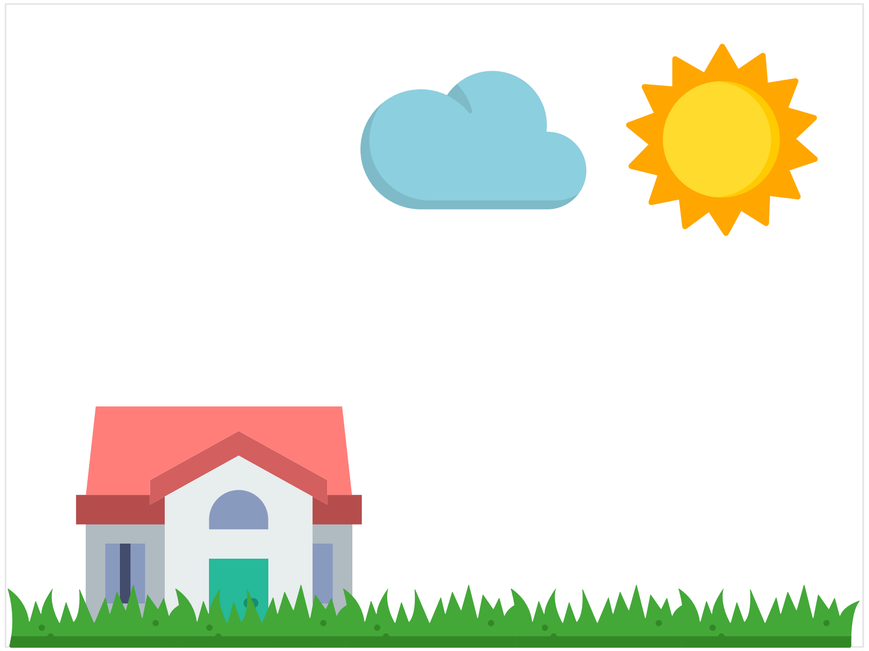


Video codecs provide a simple interface

`encode([🌇,🏙️,...]) → [frames...]`

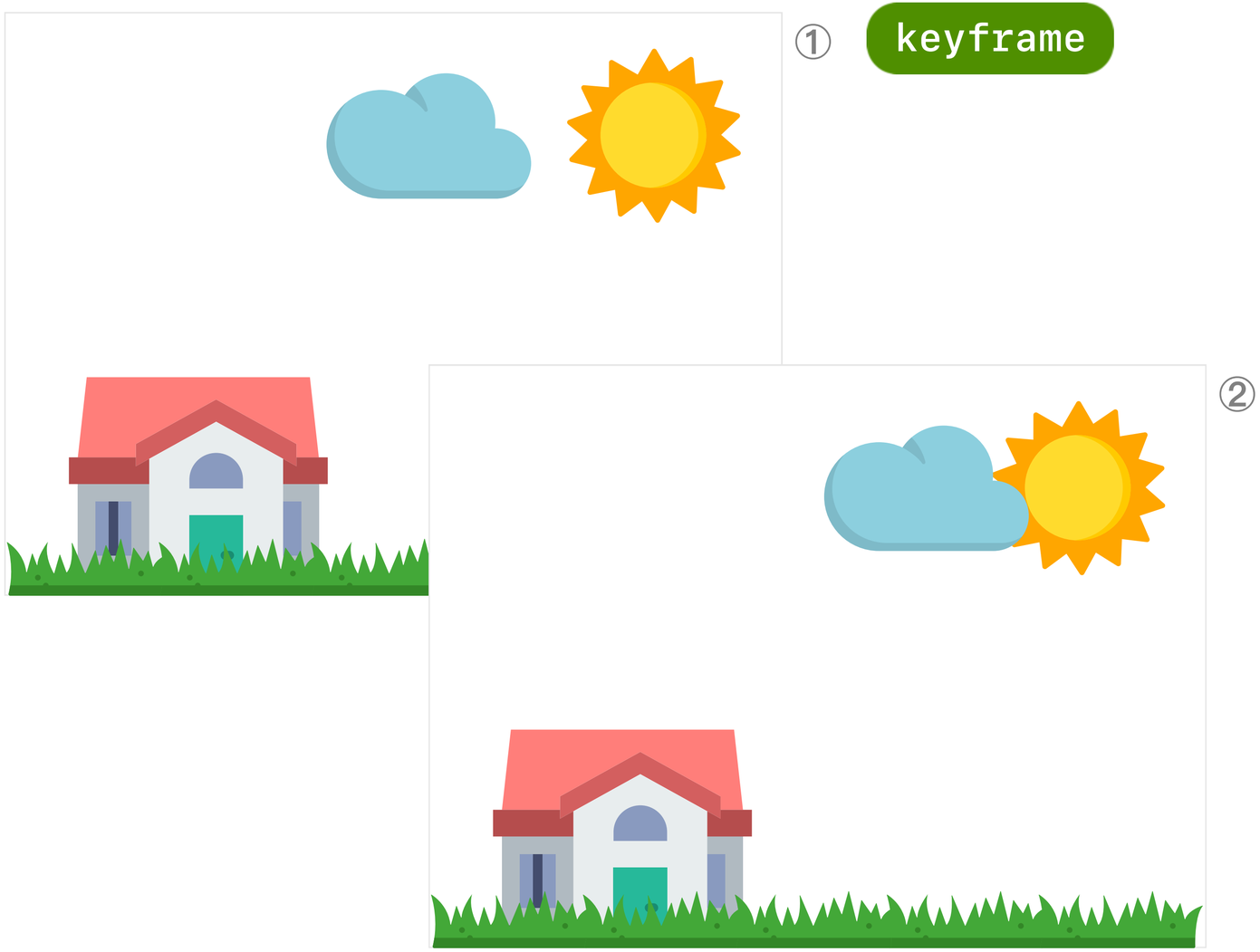
`decode([frames...]) → [🌇,🏙️,...]`

First frame in the sequence is stored in its entirety

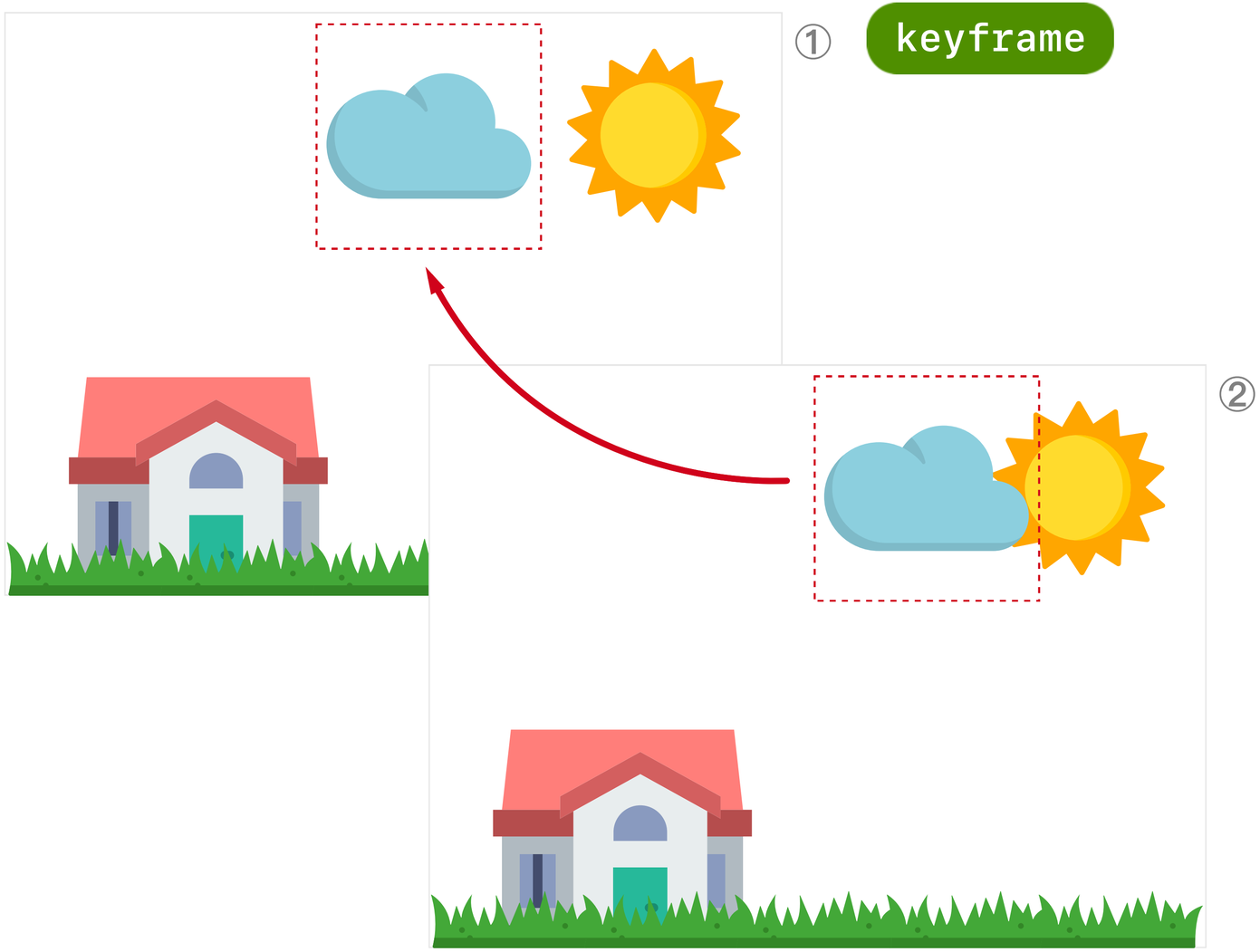


① keyframe

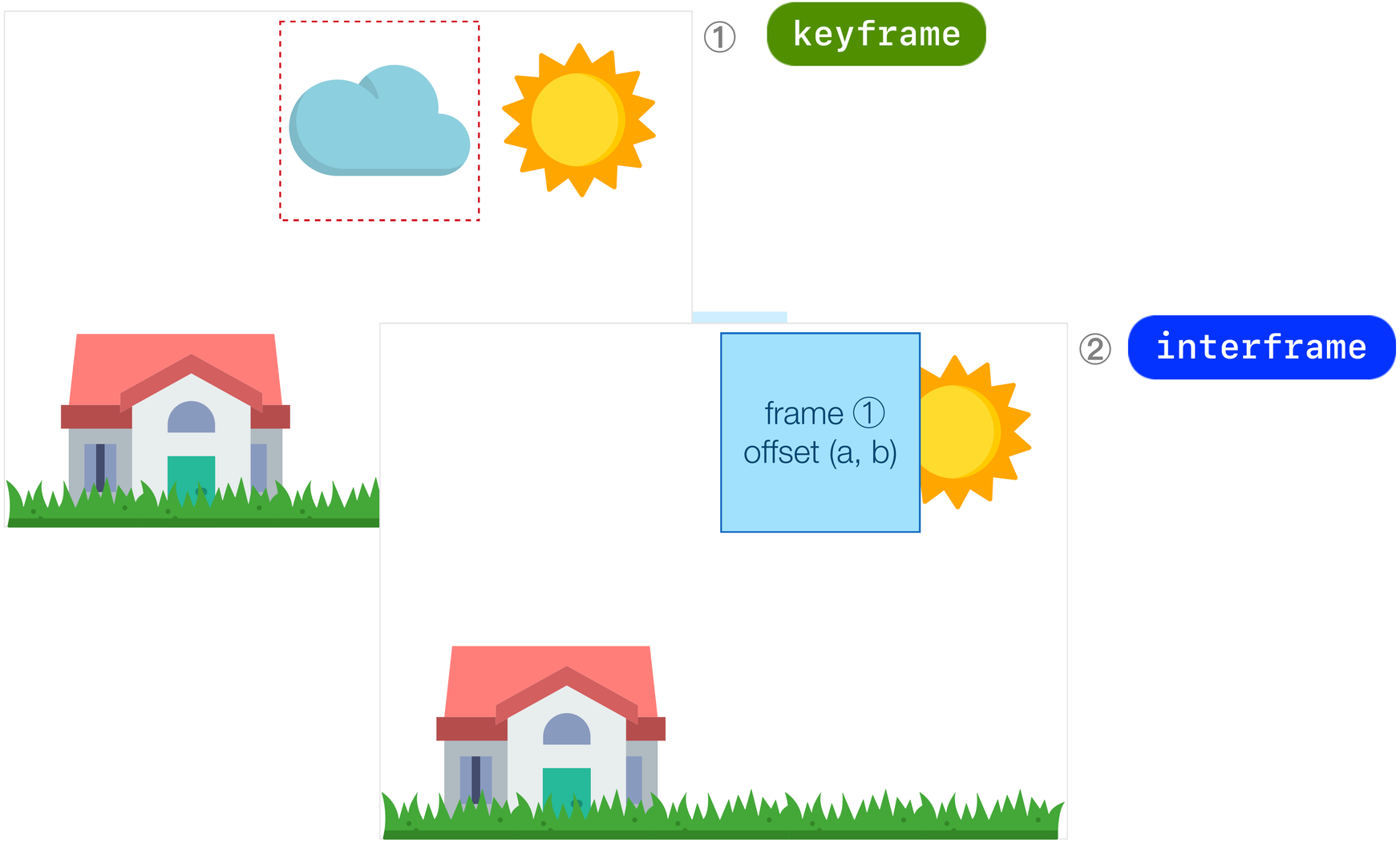
The encoder exploits the similarities between frames to compress the video



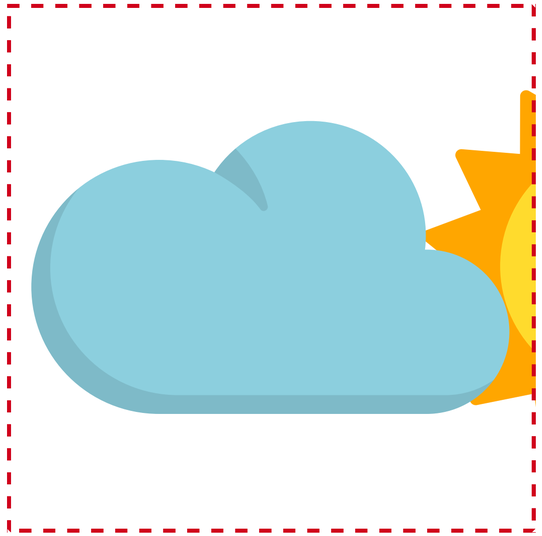
The encoder exploits the similarities between frames to compress the video



The encoder exploits the similarities between frames to compress the video

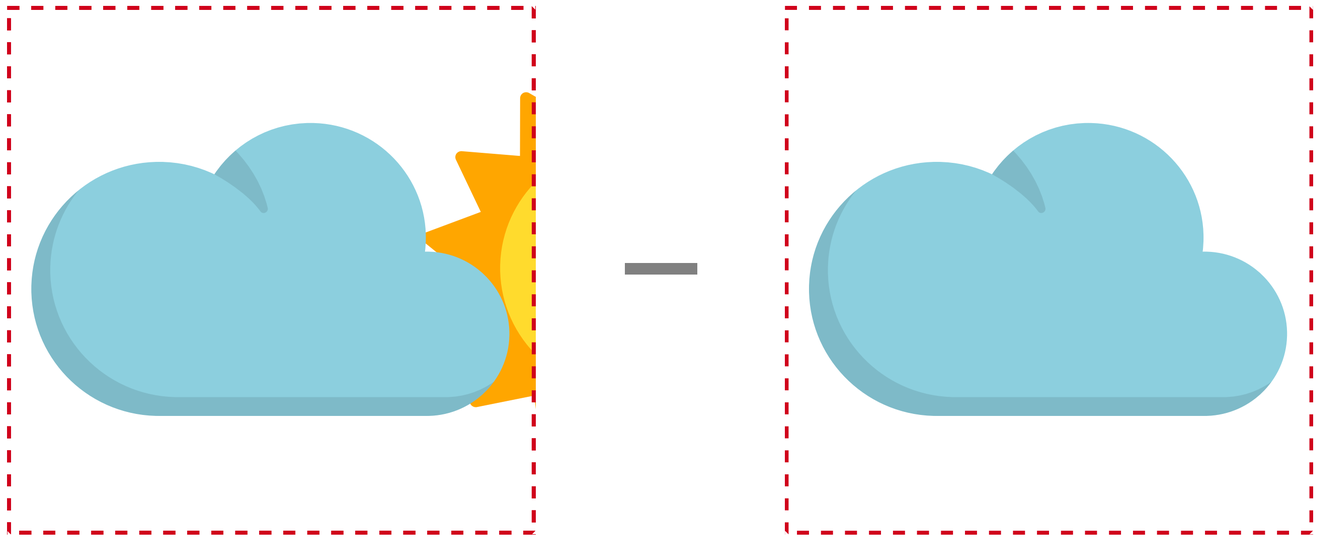


Predictions are not always perfect



current block

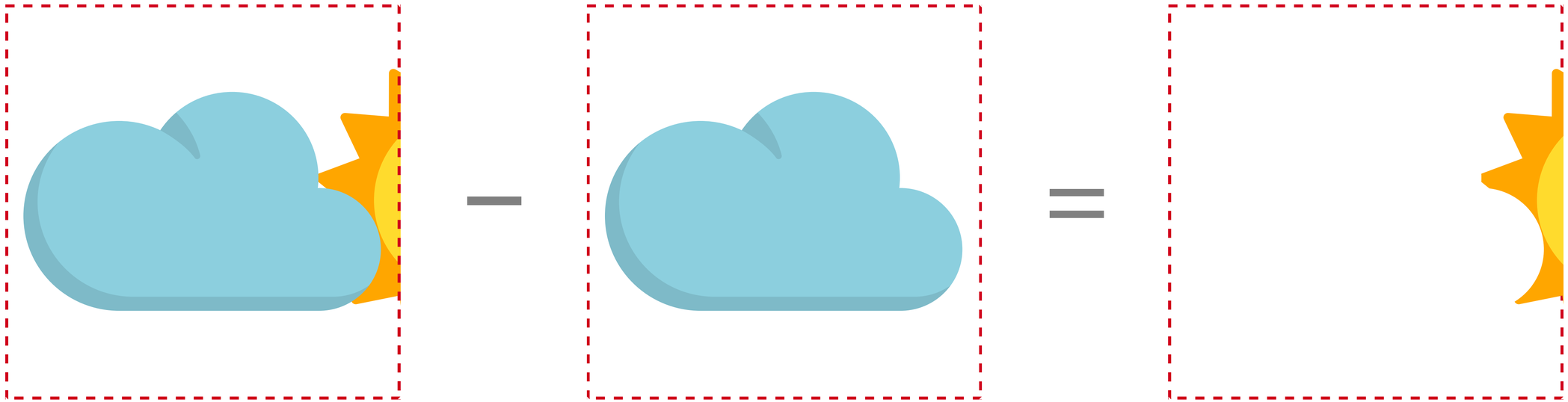
Predictions are not always perfect



current block

closest block in previous frame

The encoder also needs to store the *residues*

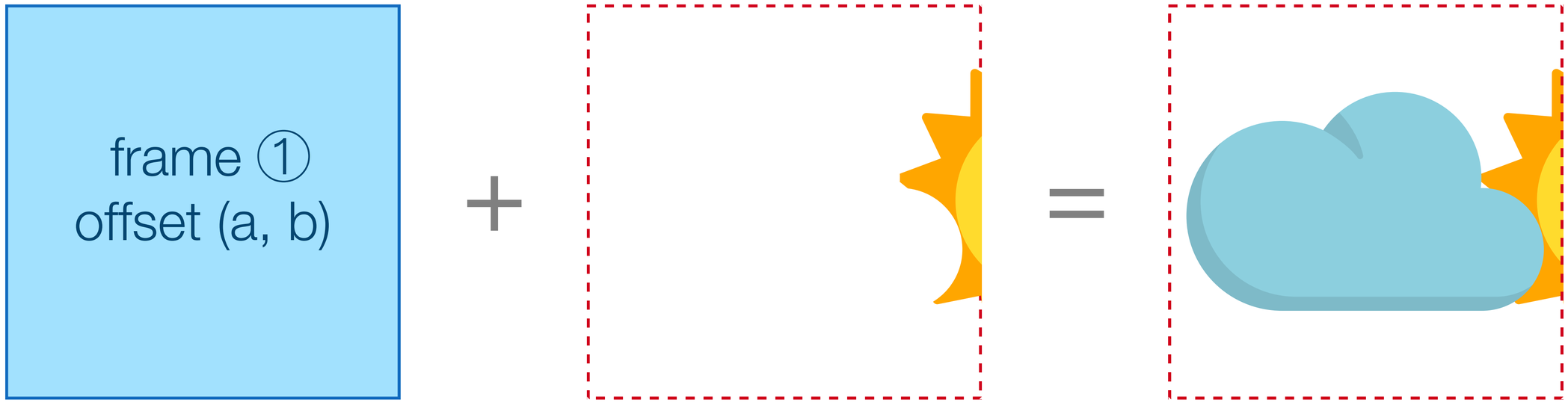


current block

closest block in previous frame

residue

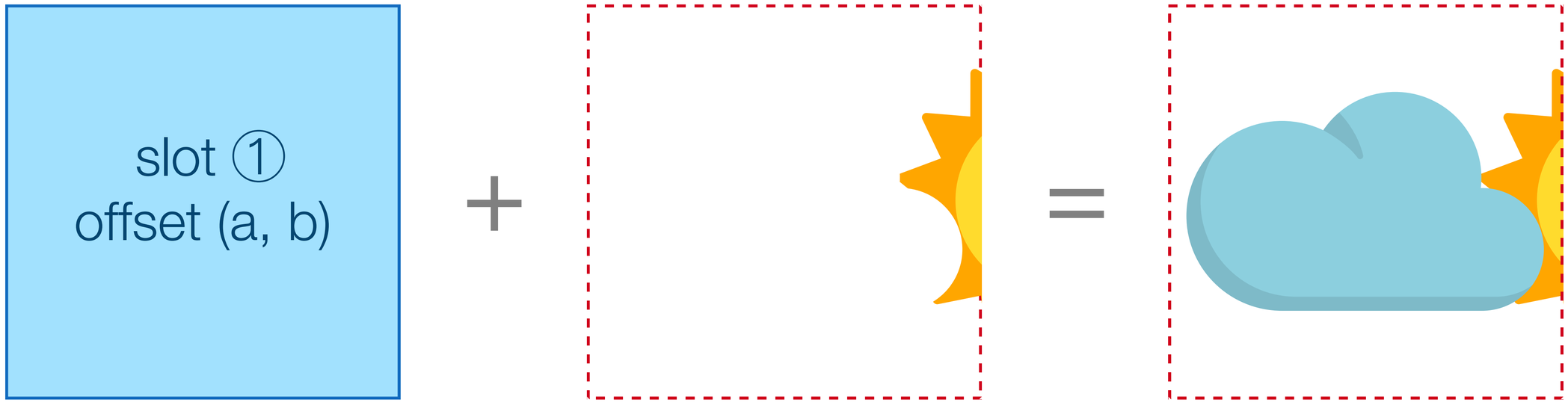
Decode process



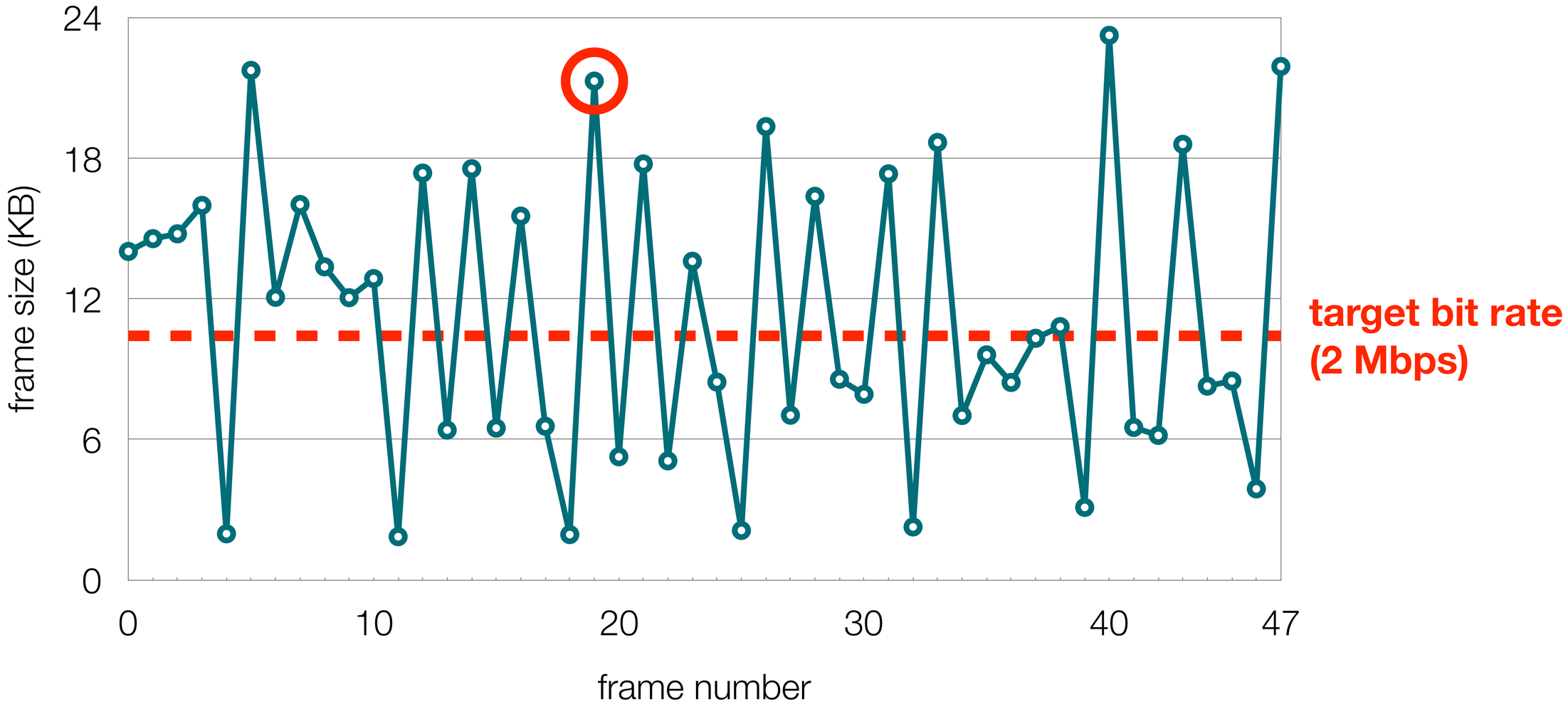
What frames can be referenced?

- The codec keeps a list of references that can be used by frames.
- Each frame can replace one of the reference slots with its output.

Decode process



The encoder can only achieve the bit rate *on average*



Why real-time video is a different problem

1. Video streaming
   Latency target: **tens of minutes**

2. Live video
   Latency target: **5—30 seconds**

   Encoder timescale to achieve target bit rate: **1-2 seconds**
   How often a keyframe can be inserted: **2 seconds**

3. Real-time video
   Latency target: **tens of milliseconds**

The challenge:
Getting an accurate frame out of an inaccurate codec

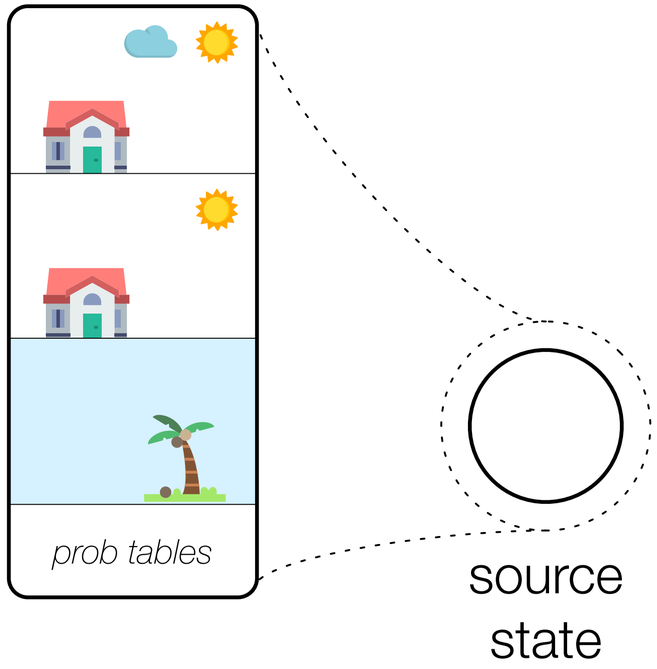
- **Trial and er**
  Encode with hat fits.



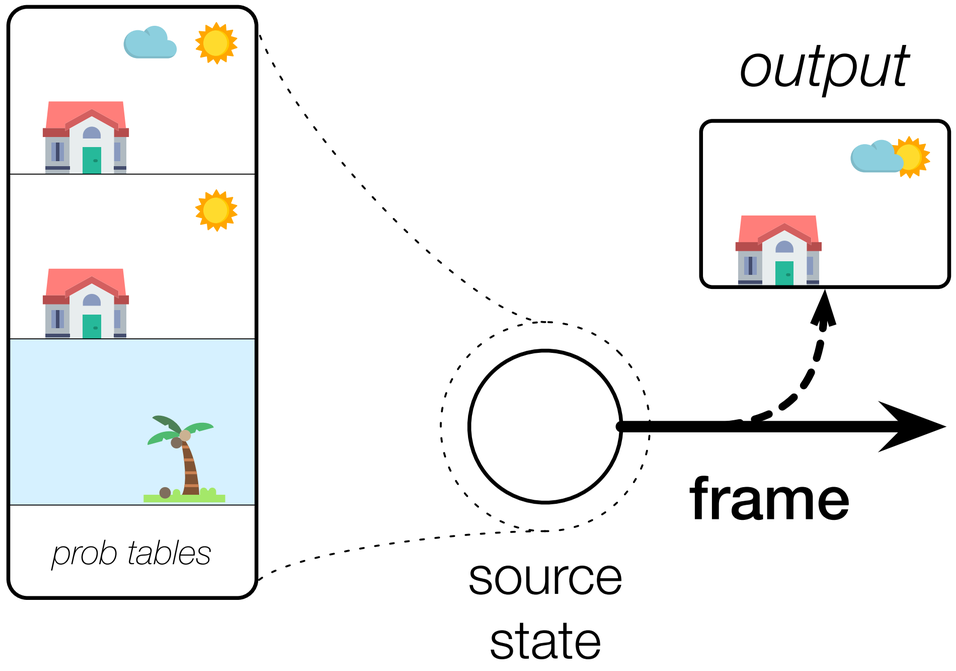
The video codec is stateful

- Frames are depending on references that live in the mind of the decoder.
- Every frame can change this set of references, and future frames are counting on that.

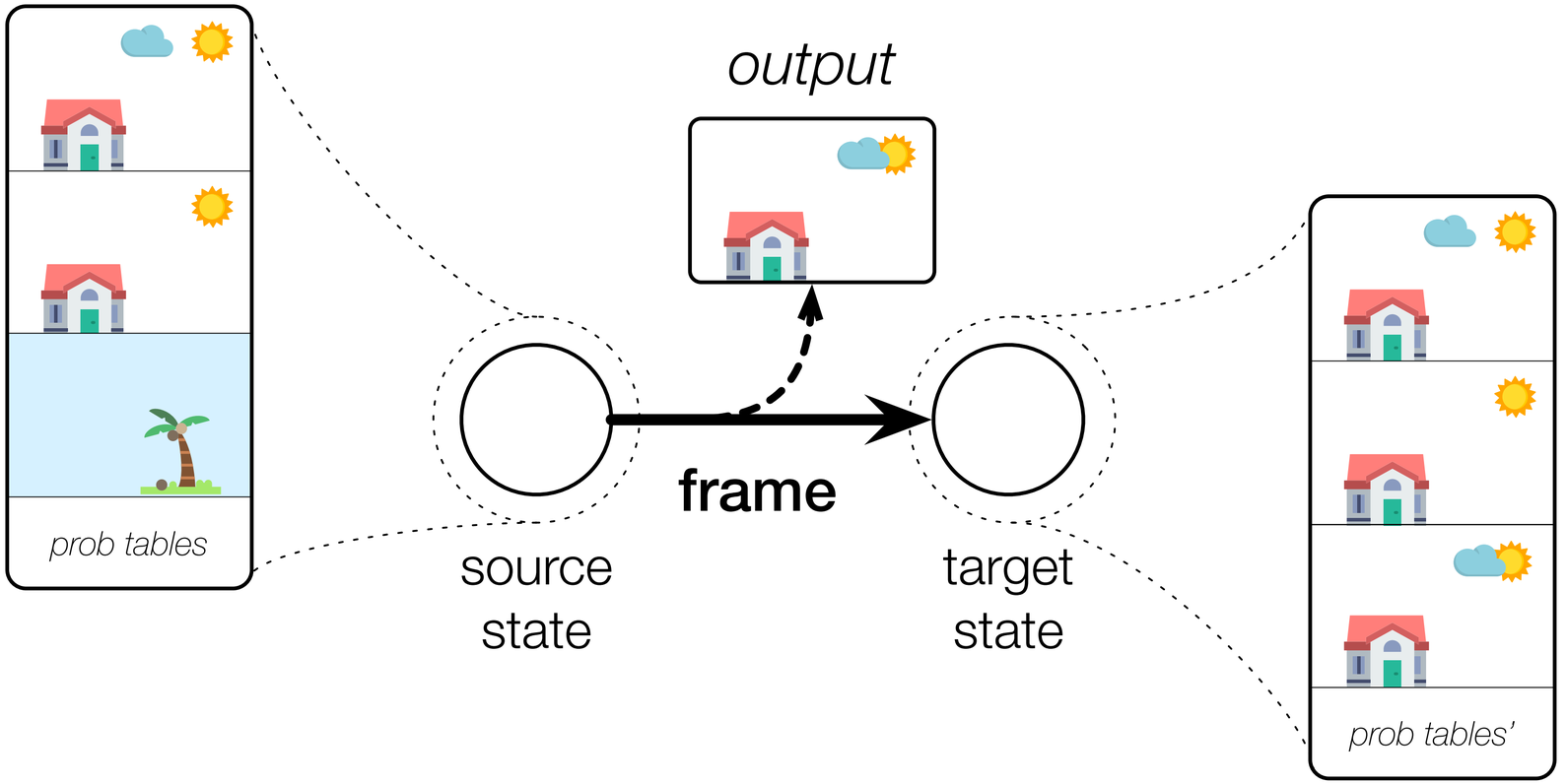
Video codec



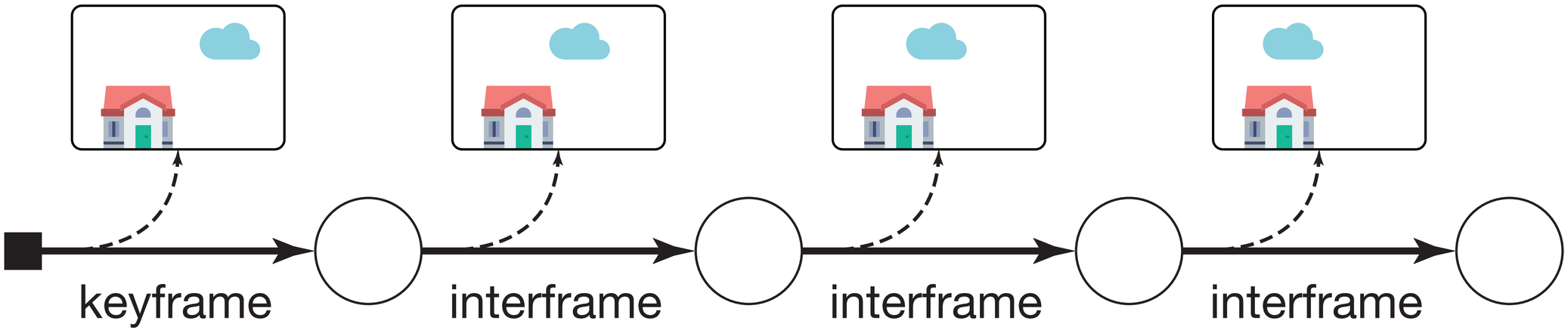
Video codec



Video codec


output
prob tables
source
state
frame
target
state
prob tables'

Video codec is an automaton



There’s no way to undo an encoded frame in current codecs

`encode([🌇,🏙️,...]) → [frames...]`

`decode([frames...]) → [🌇,🏙️,...]`

The state is internal to the encoder—no way to save/restore the state.

Functional video codec to the rescue

encode(*state*, 🌆) → *state′*, frame

decode(*state*, frame) → *state′*, 🌆

Salsify's functional video codec exposes the state that can be saved/restored.

Described in ***Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads***, NSDI '17

Order two, pick the one that fits!

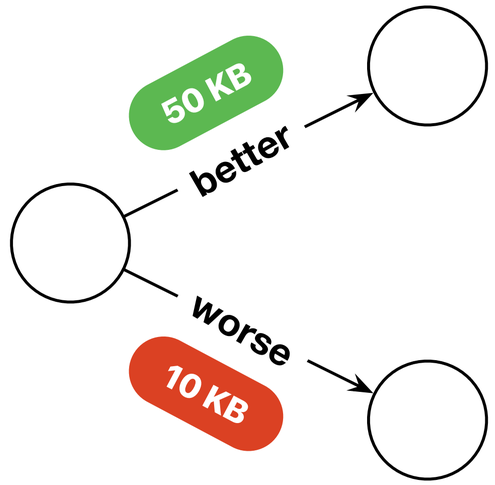
- Salsify's functional video codec can **explore different execution paths** without committing to them.

- For each frame, codec presents the transport with *three* options:

  🔺 A slightly-higher-quality version,

  🔻 A slightly-lower-quality version,

  ❌ **Discarding the frame.**

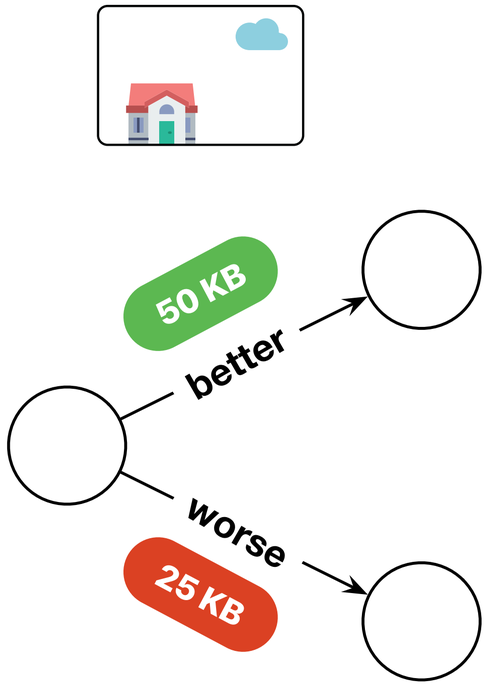


Salsify's architecture:
Unified control loop



Codec → Transport

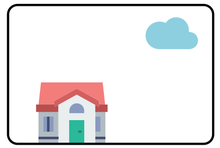
“Here’s two versions of the current frame.”

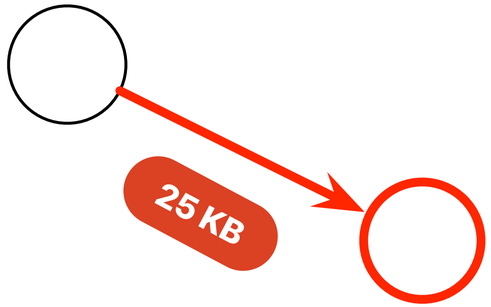


target frame size 30 KB

Transport → Codec

“I picked option 2. Base the next frame on its exiting state.”

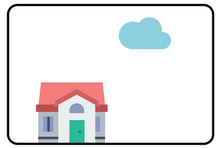


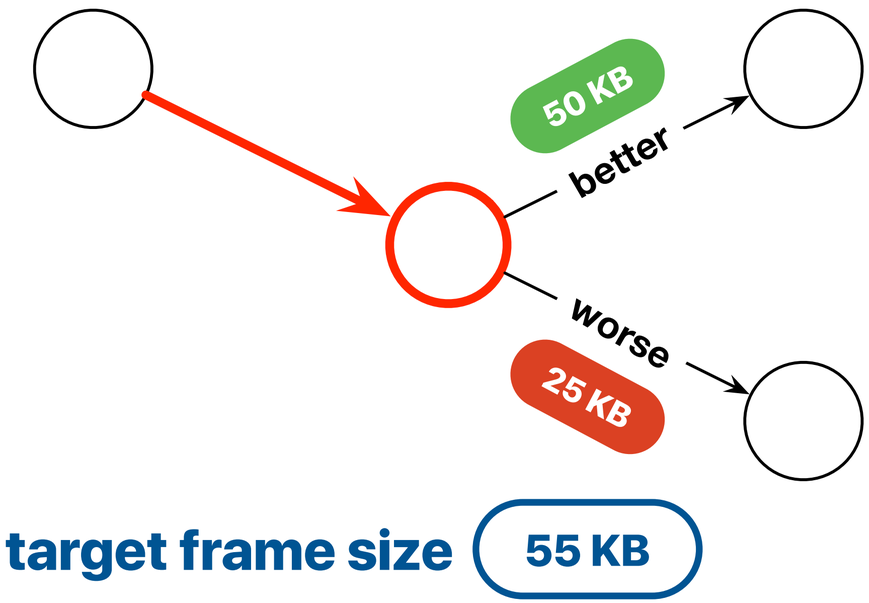


target frame size 30 KB

Codec → Transport

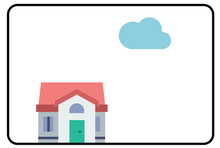
“Here’s two versions of the latest frame.”

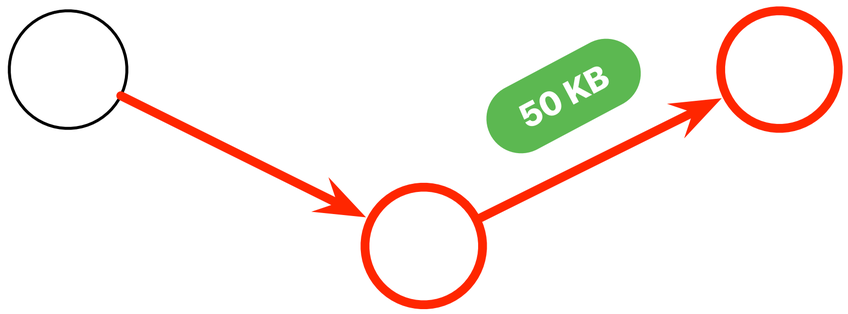




Transport → Codec

“I picked option 1. Base the next frame on its exiting state.”

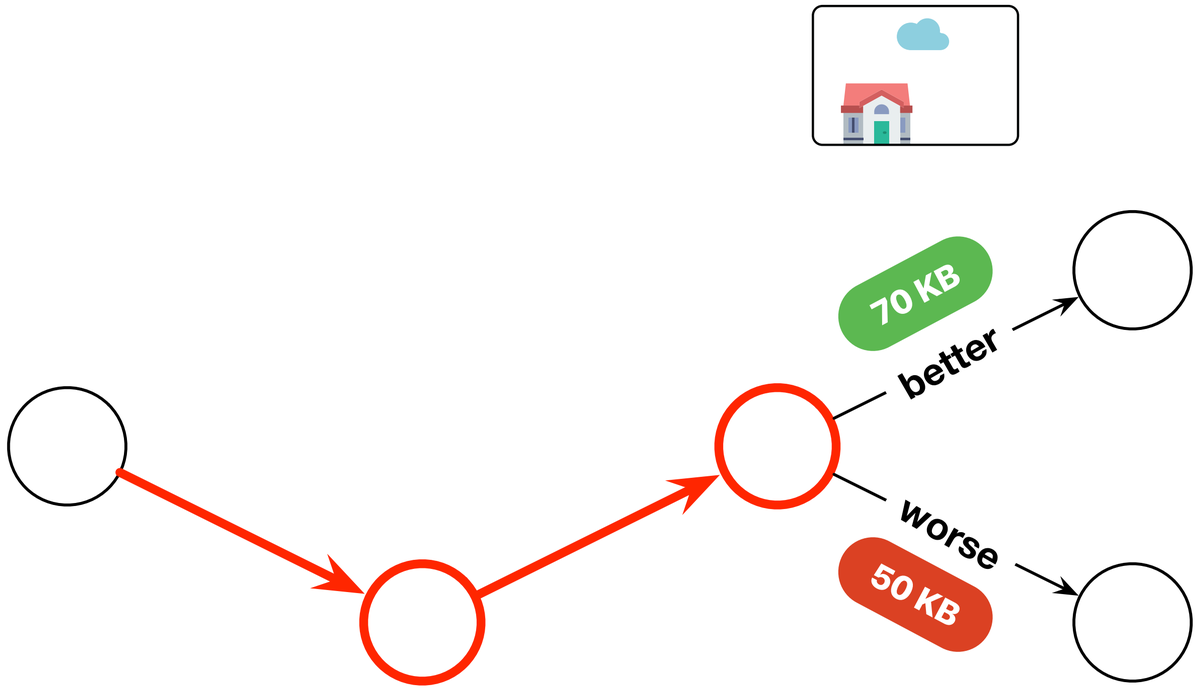




target frame size **55 KB**

Codec → Transport

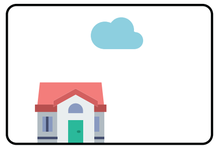
“Here’s two versions of the latest frame.”

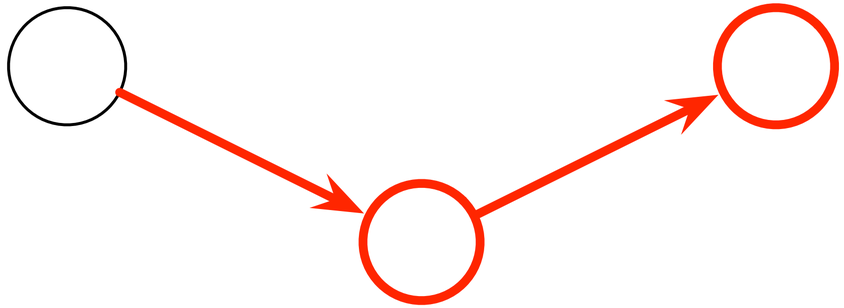


target frame size 5 KB

Transport → Codec

“I cannot send any frames right now. Sorry, but discard them.”

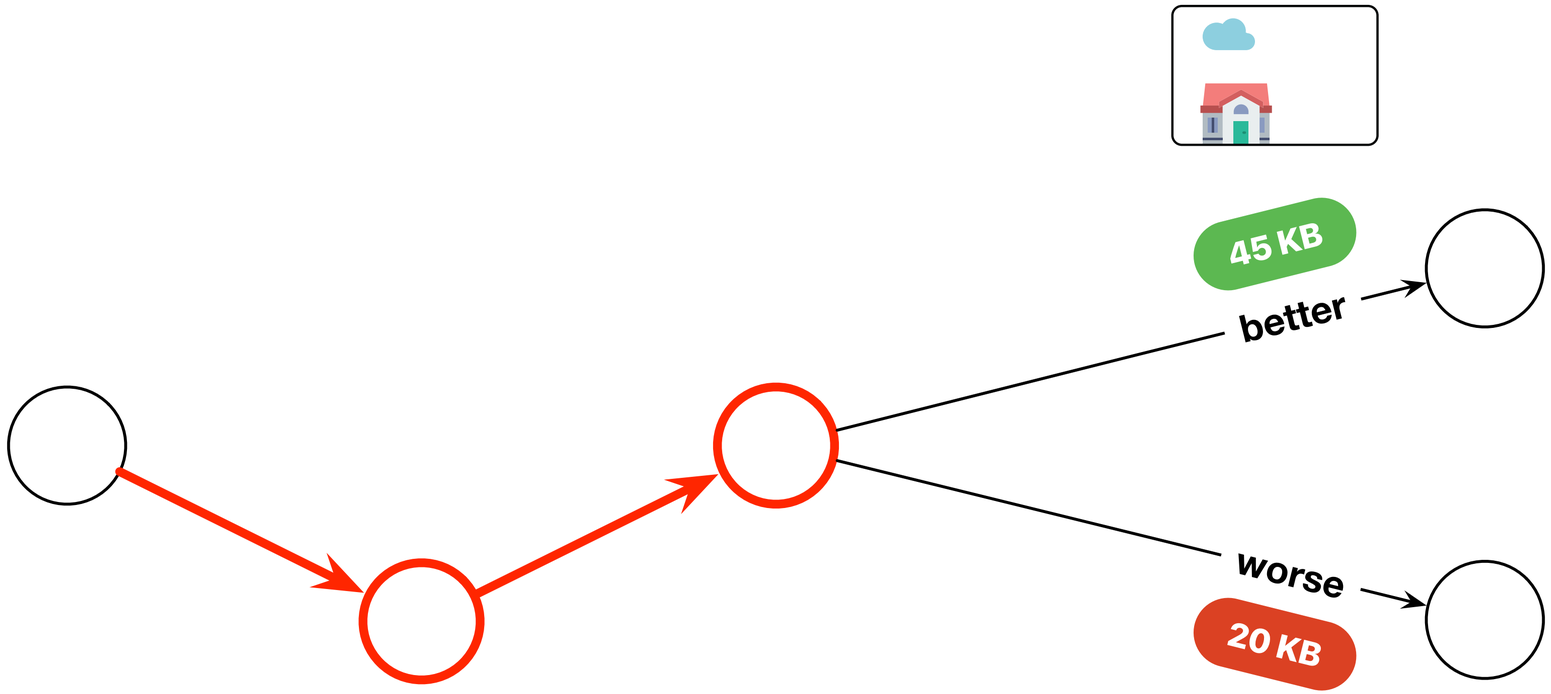




target frame size **5 KB**

Codec → Transport

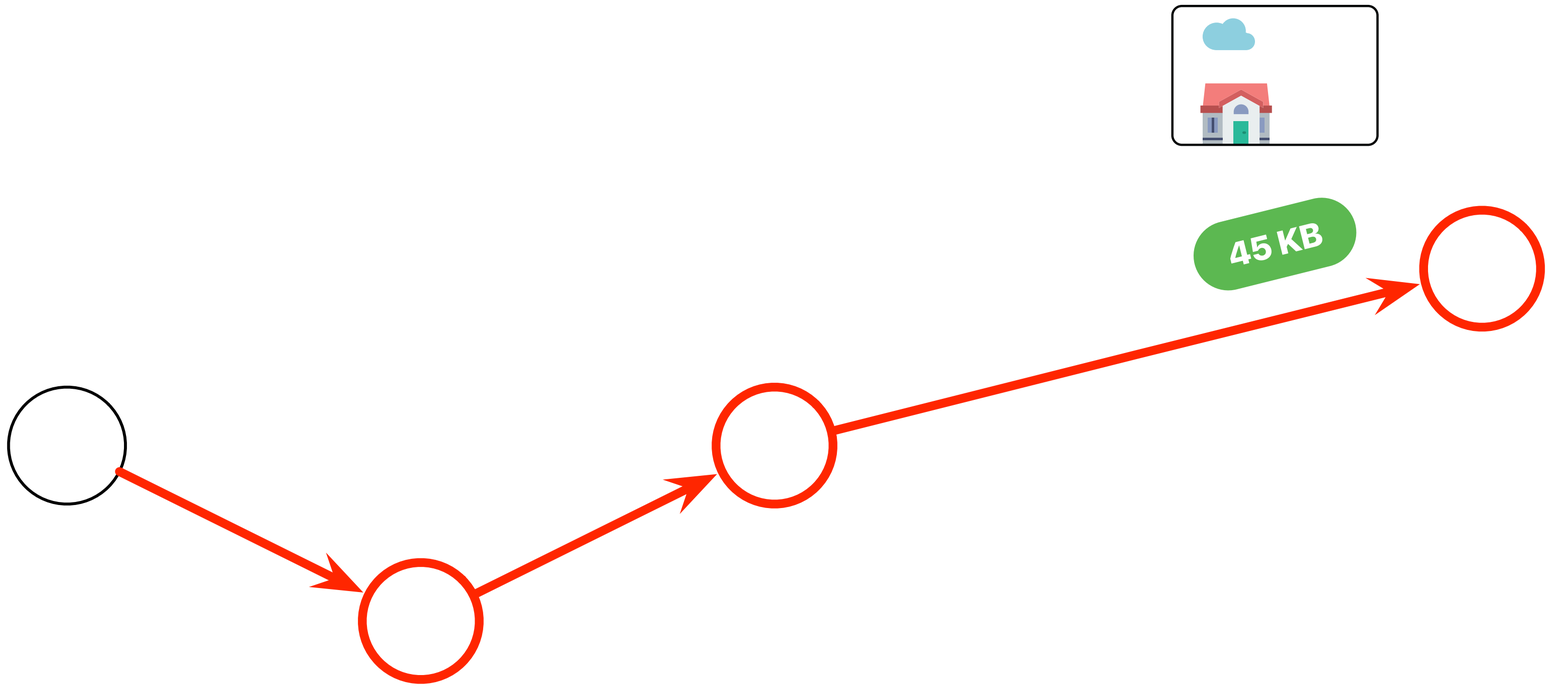
“Fine. Here’s two versions of the latest frame.”



target frame size 50 KB

Transport → Codec

“I picked option 1. Base the next frame on its exiting state.”



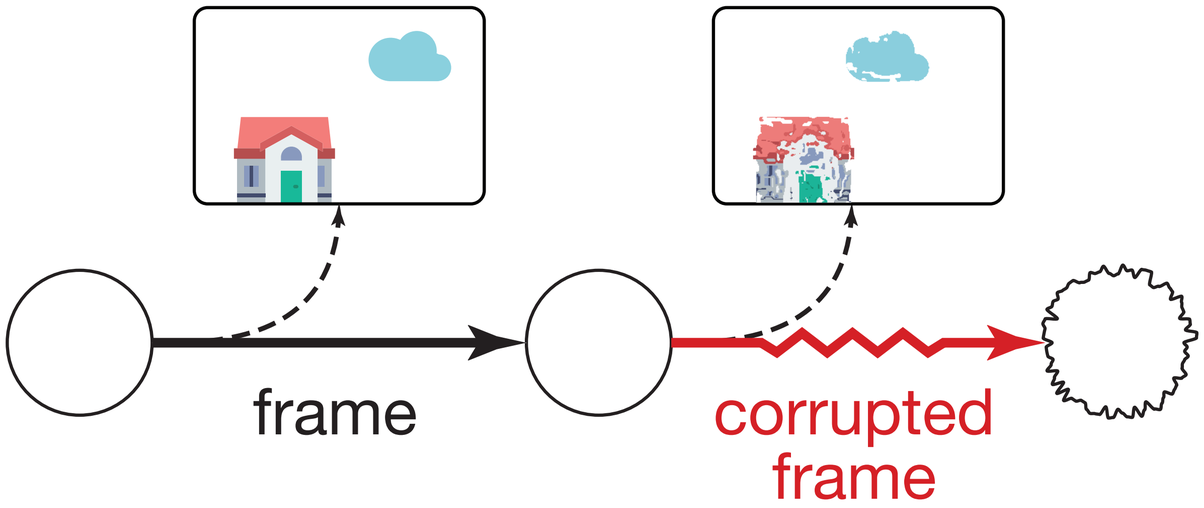
target frame size 50 KB

There’s no notion of **frame rate** or **bit rate** in the system.
Frames are sent when the network can accommodate them.

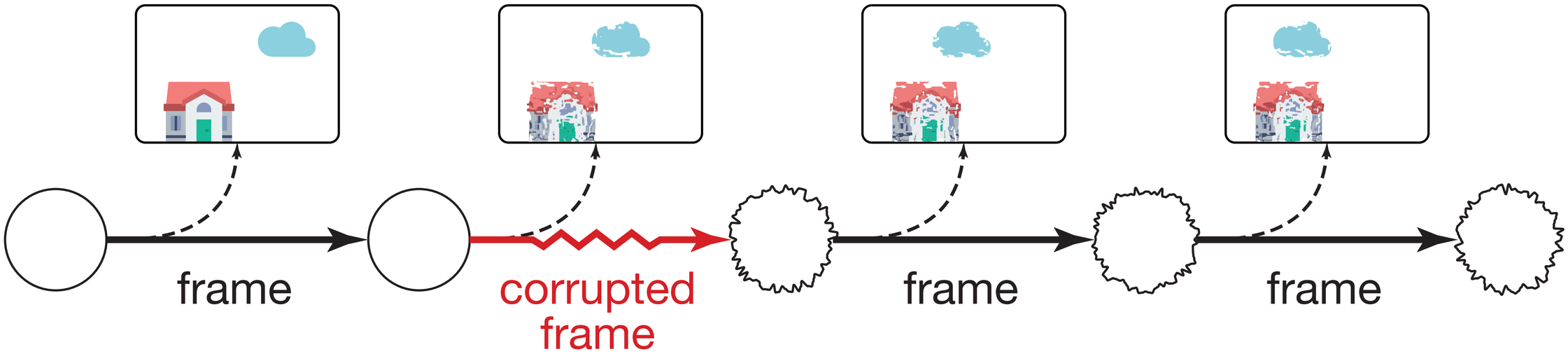
Loss recovery

- **Option 1:** Ignore dropped packets.
- **Option 2:** Retransmit dropped packets.
- **Option 3:** Restart the video stream with a keyframe or a “key-slice.”

Loss corrupts the current frame...



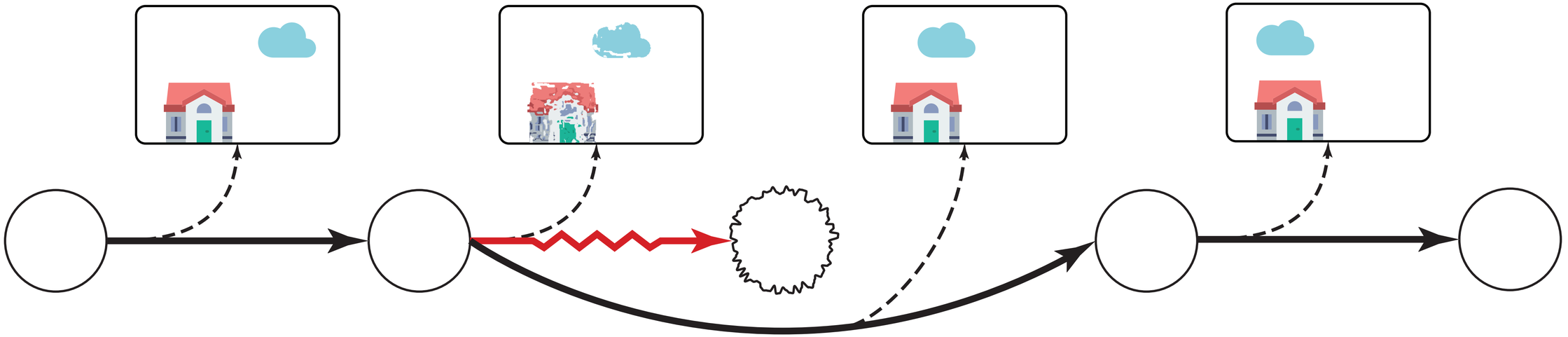
Loss corrupts the current frame... and the rest!



Loss recovery

- ~~**Option 1:** Ignore dropped packets.~~
- **Option 2:** Retransmit dropped packets.
- **Option 3:** Restart the video stream with a keyframe or a “key-slice.”

Option 5, Salsify's way: Jump back to the last correct state



Measurement Testbed

Goals for the measurement testbed

- A system with
  reproducible input video and
  reproducible network traces that runs
  unmodified version of the system-under-test.

- Target QoE metrics: **image quality** and **video delay**.

Goals for the measurement testbed

- A system with
  reproducible input video and
  reproducible network traces that runs
  unmodified version of the system-under-test.

- Target QoE metrics: **image quality** and **video delay**.

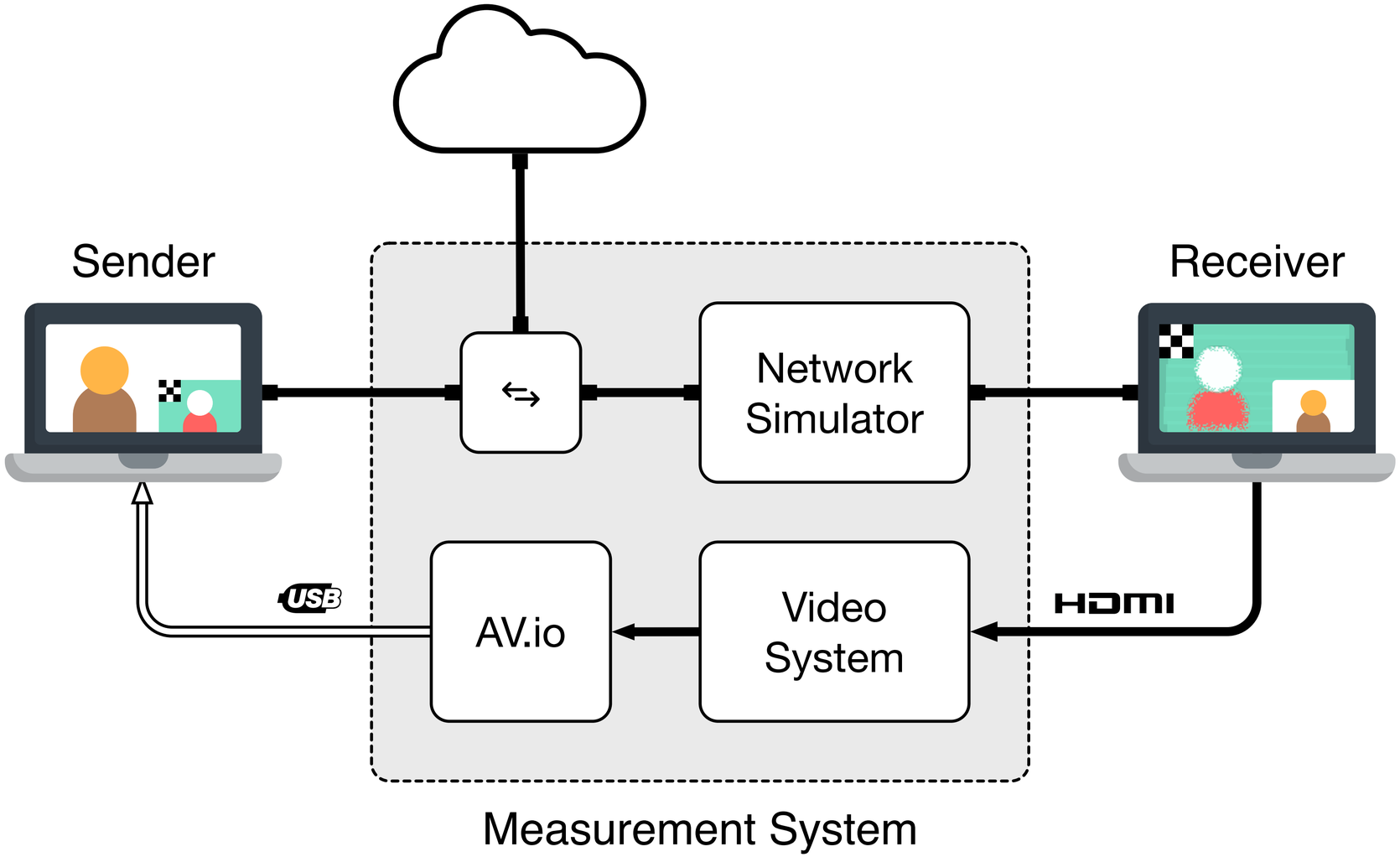
Goals for the measurement testbed

- A system with
  reproducible input video and
  reproducible network traces that runs
  unmodified version of the system-under-test.

- Target QoE metrics: **image quality** and **video delay**.

Video delay




Sender
Receiver
Network Simulator
AV.io
Video System
USB
HDMI
Measurement System

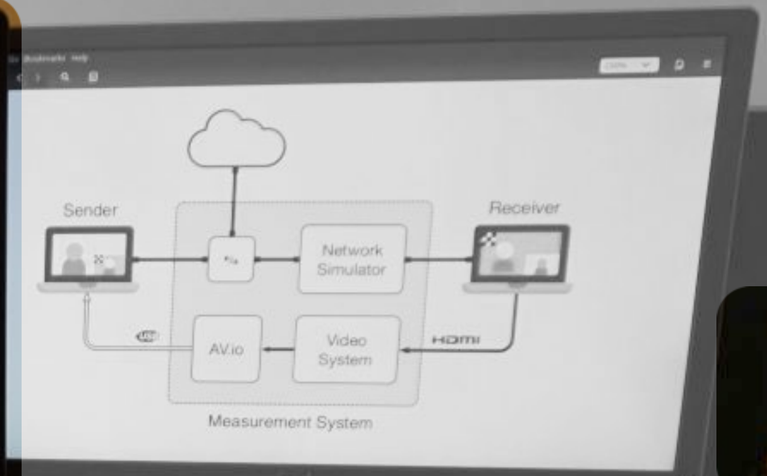

Sender
Receiver
Network Simulator
AV.io
Video System
HDMI
Measurement System
DELL
AV.io 4K

emulated network





barcoded video





receiver HDMI output



video in/out (HDMI)



HDMI to USB camera


emulated network
barcoded video
Sender
Receiver
Network Simulator
AV.io
Video System
HDMI
Measurement System
receiver HDMI output
video in/out (HDMI)
HDMI to USB camera

emulated network





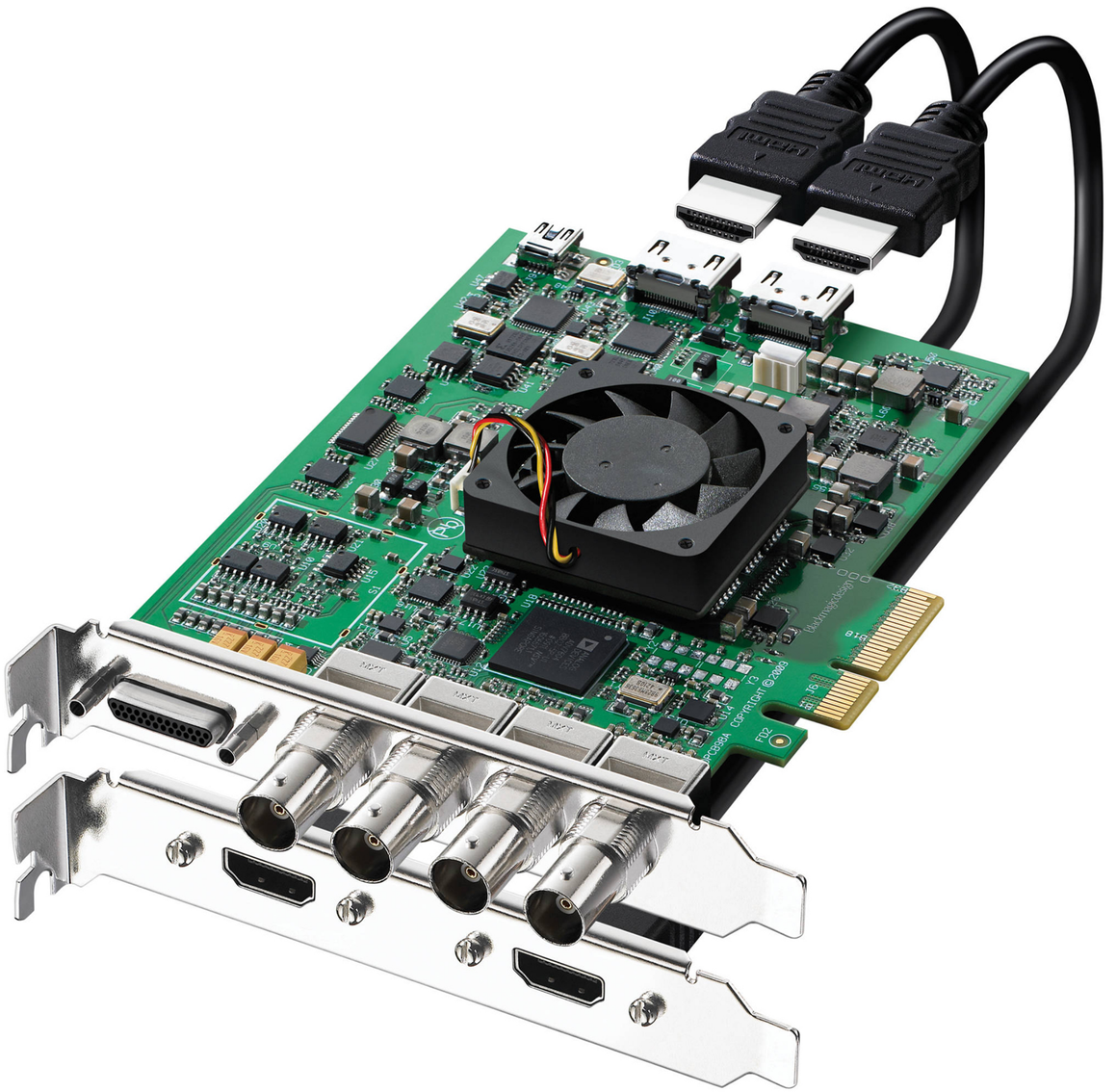
barcoded video



video in/out (HDMI)



HDMI to U

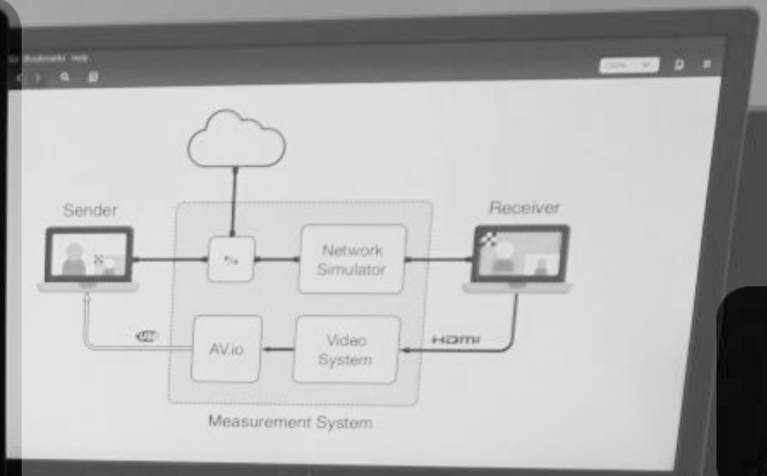


emulated network





barcoded video





receiver HDMI output



video in/out (HDMI)



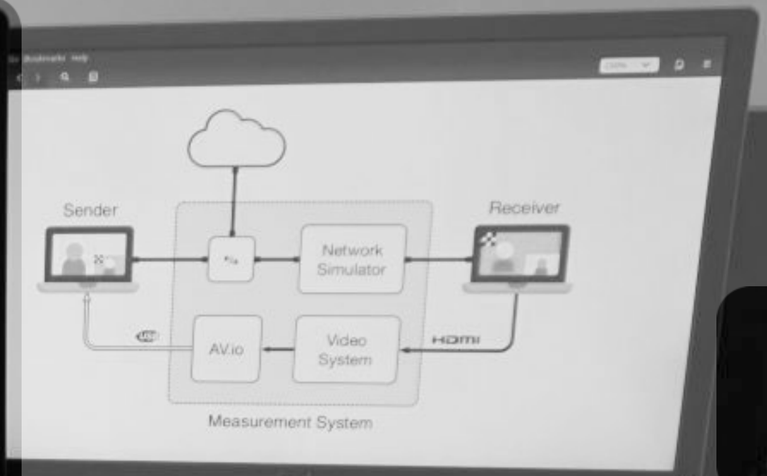
HDMI to USB camera

emulated network





barcoded video





receiver HDMI output



video in/out (HDMI)



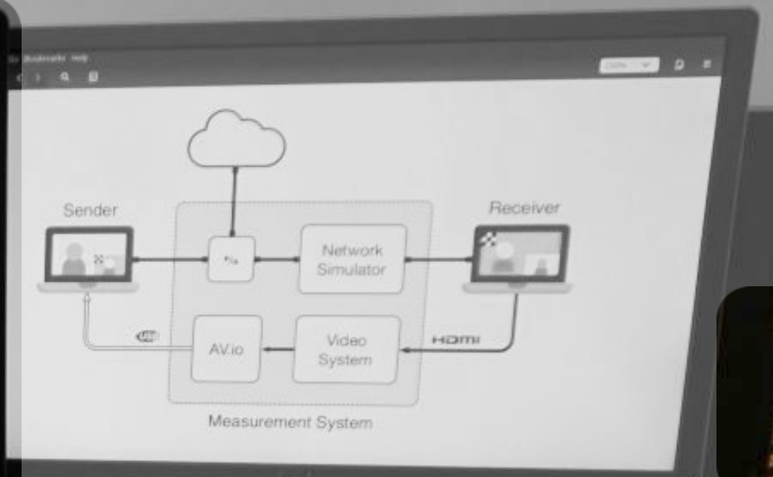
HDMI to USB camera

emulated network





barcoded video





receiver HDMI output



video in/out (HDMI)



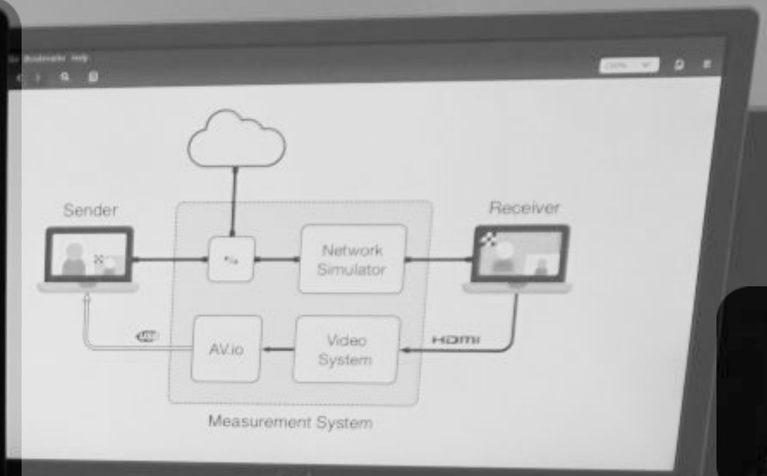
HDMI to USB camera

emulated network





barcoded video





receiver HDMI output



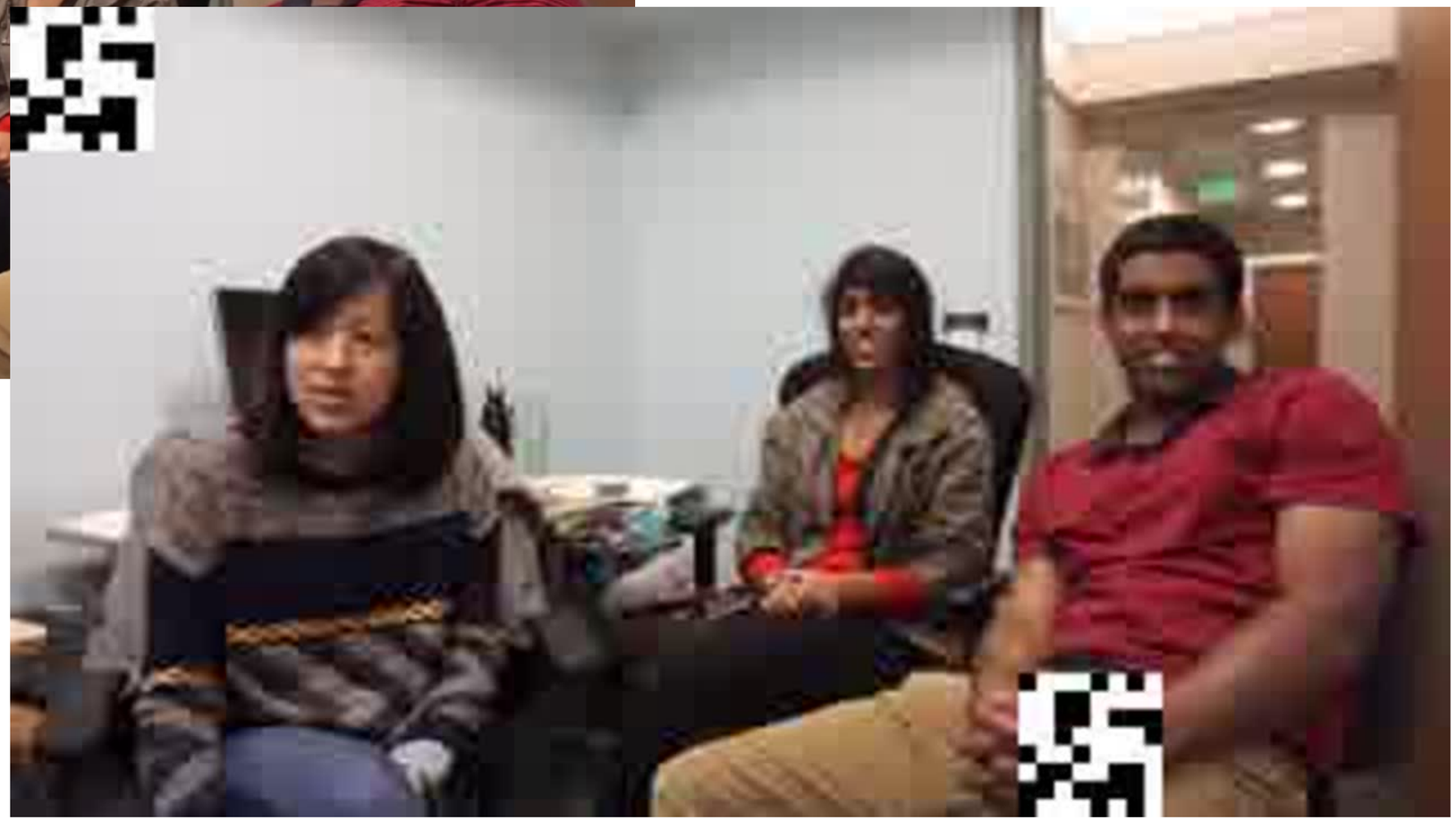
video in/out (HDMI)



HDMI to USB camera



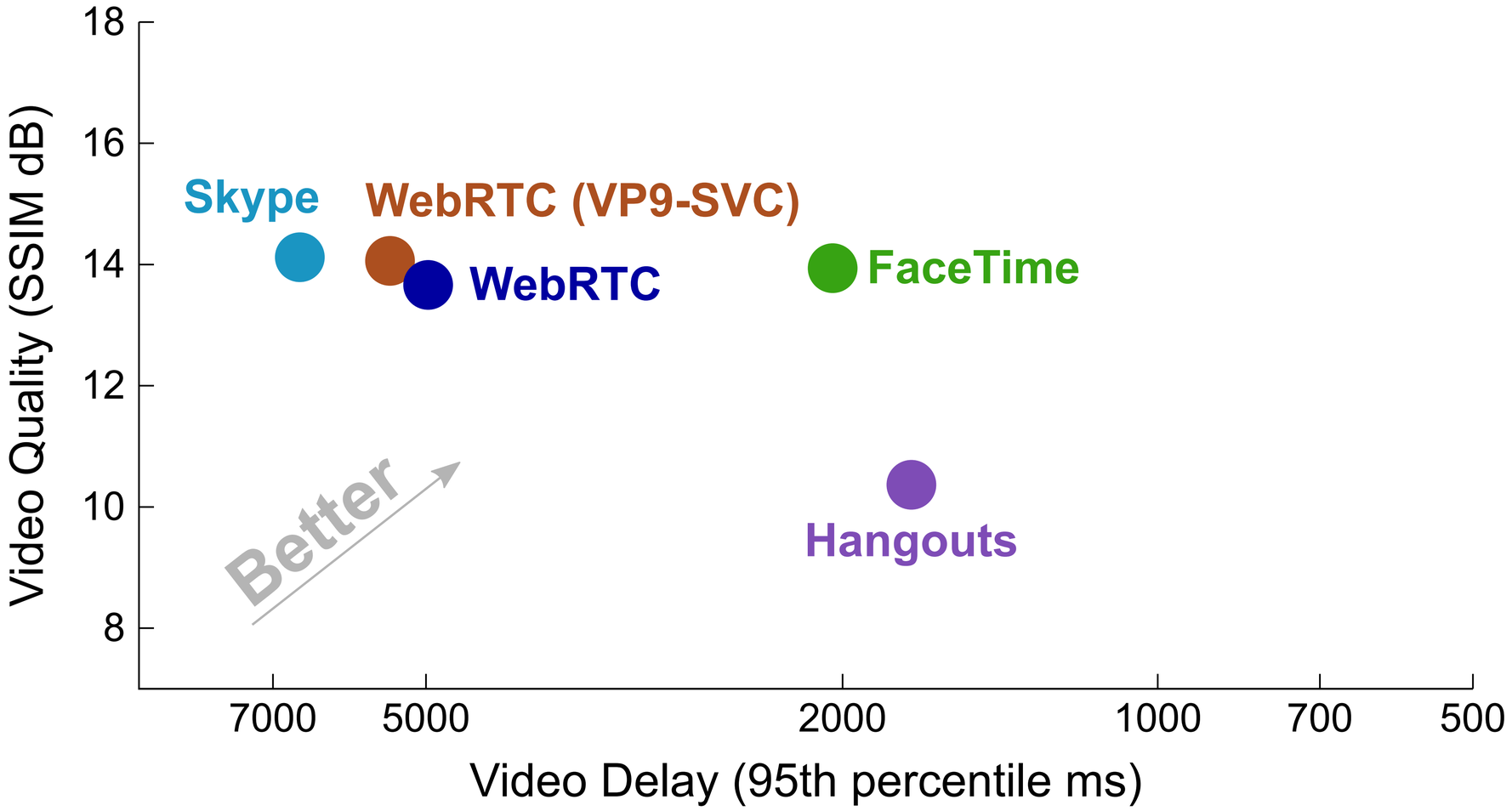
Sent Image
Timestamp: `T+0.000s`



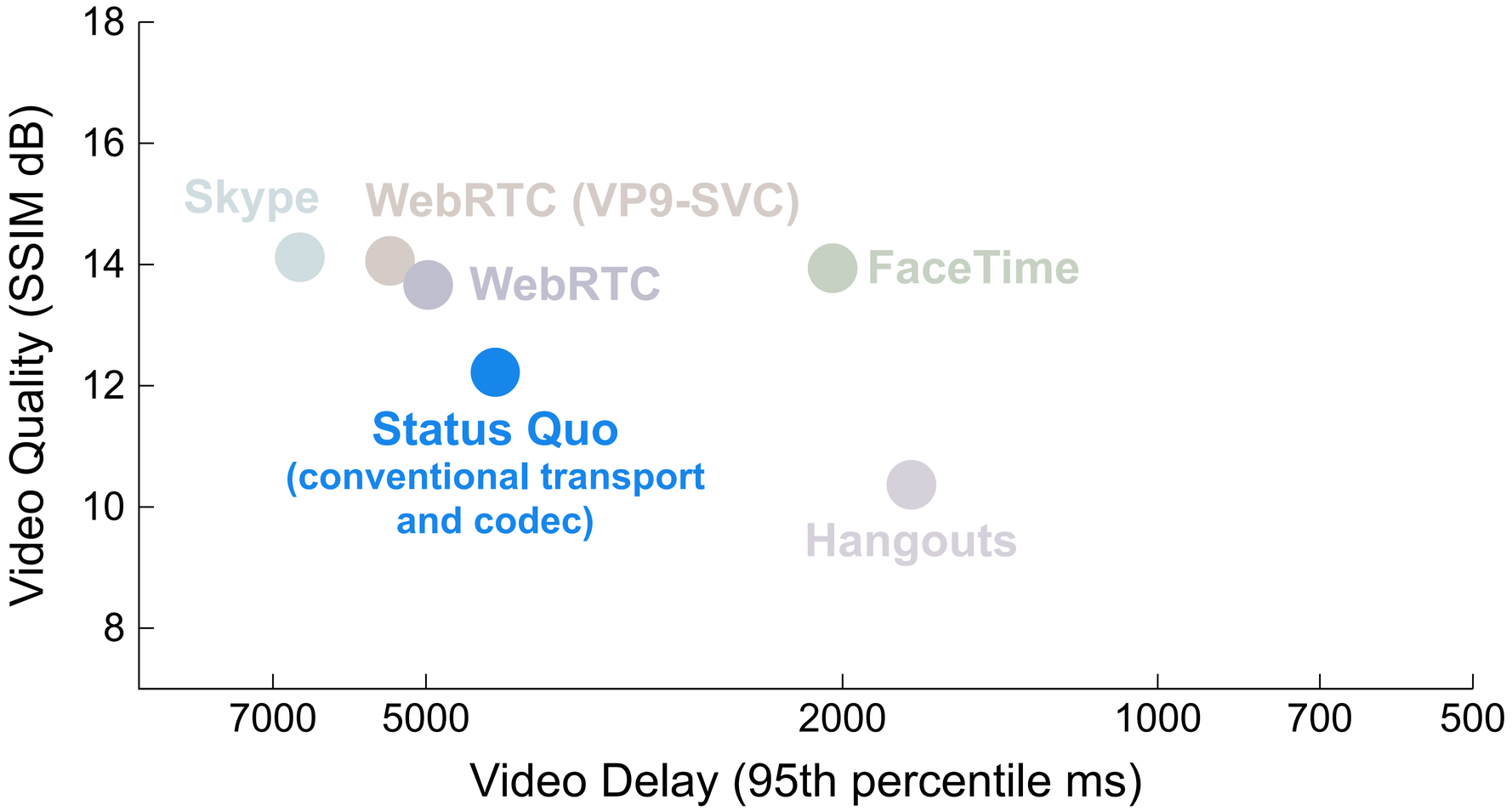
Received Image
Timestamp: `T+0.765s`
Quality: `9.76` dB SSIM

Evaluation of Salsify

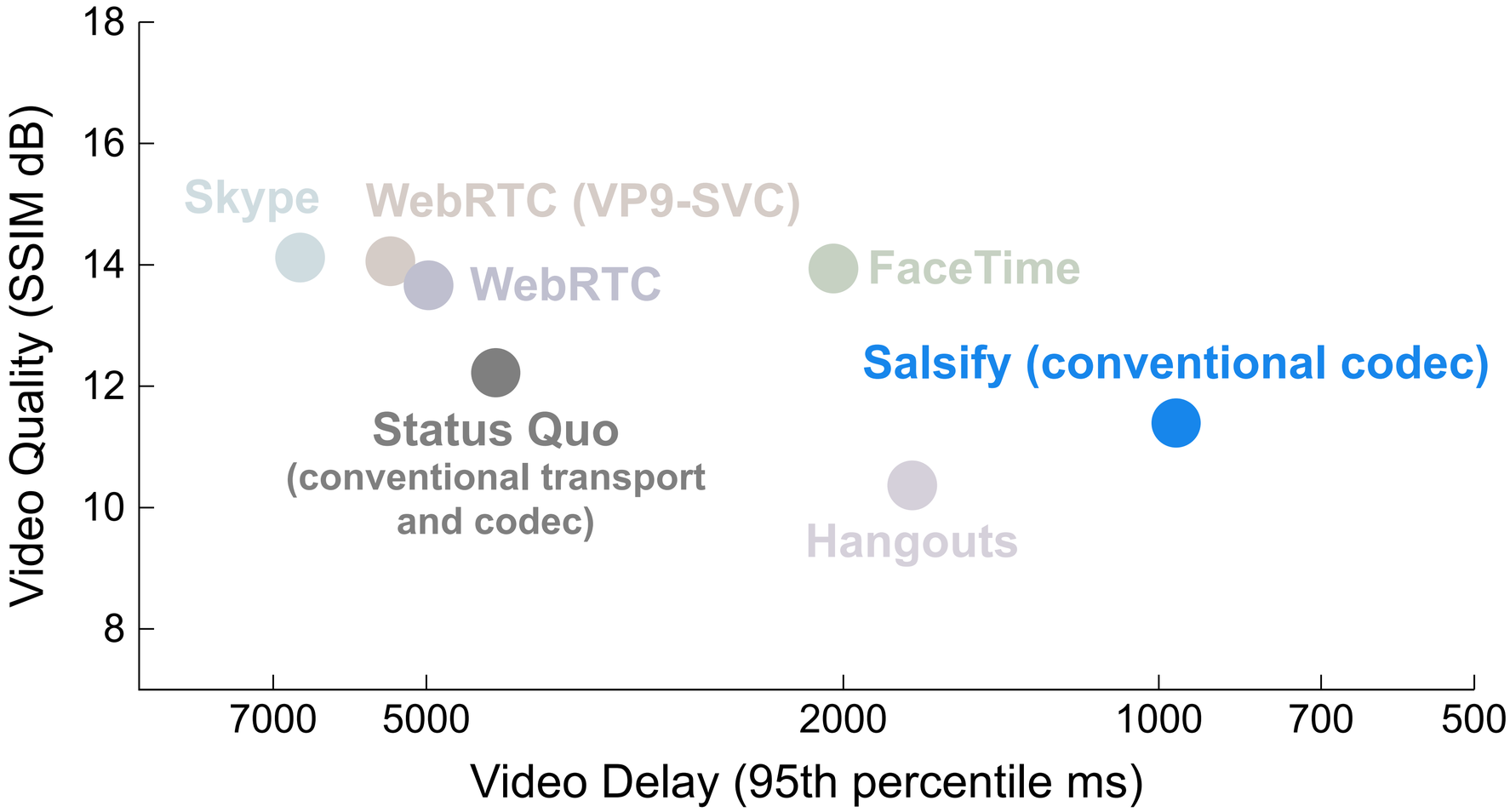
Evaluation results: **Verizon LTE Trace**



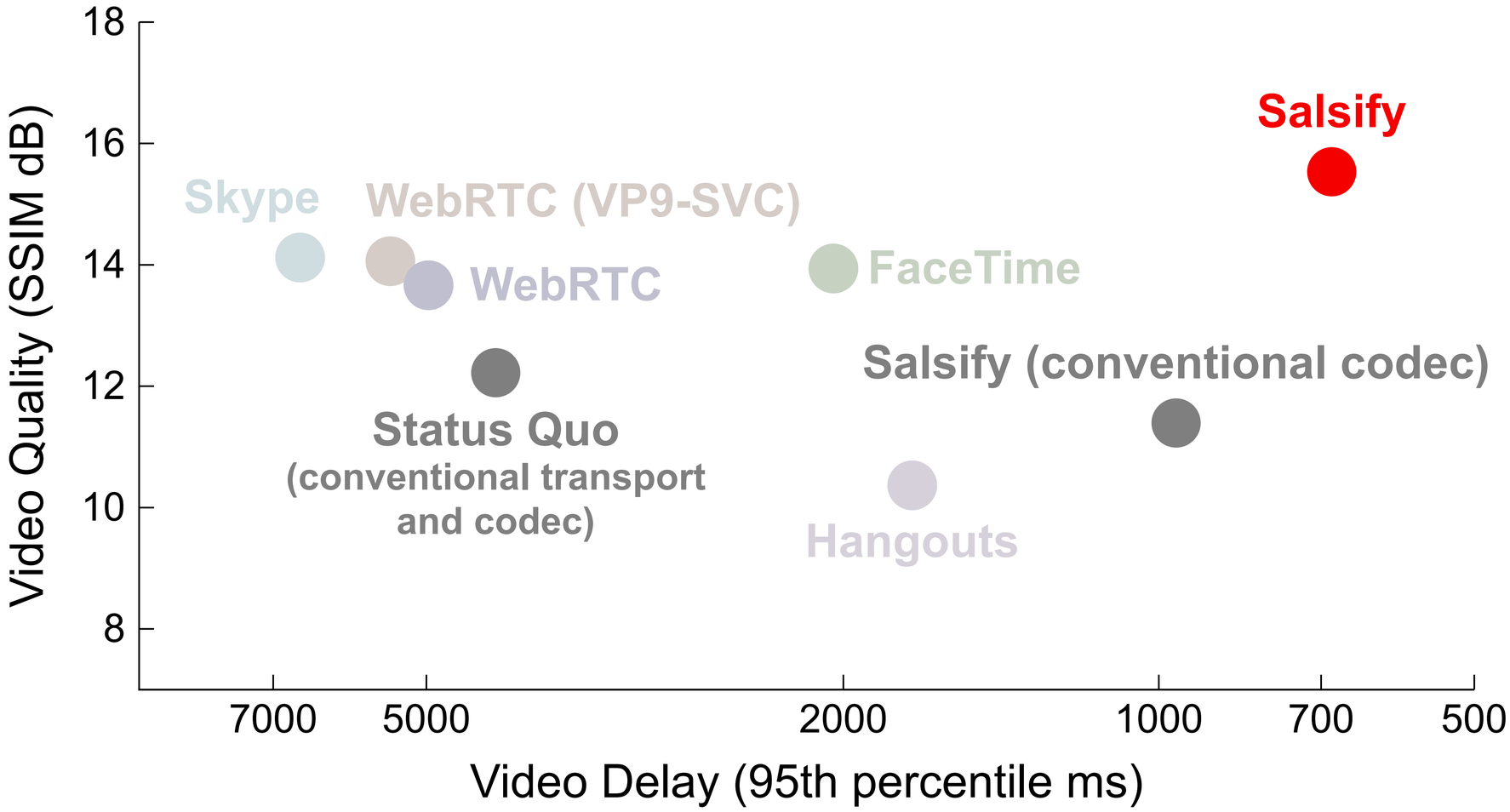
Evaluation results: **Verizon LTE Trace**



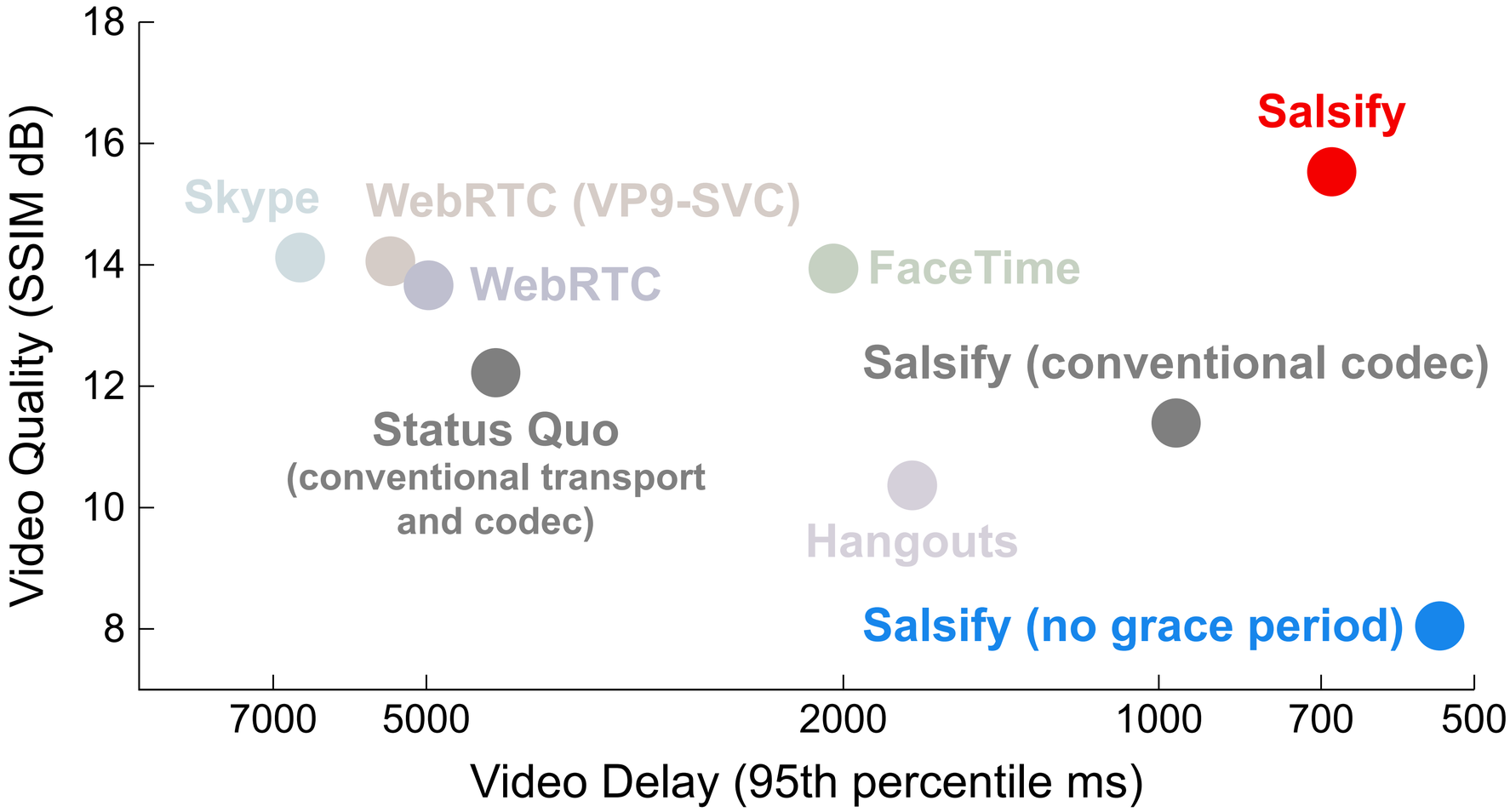
Evaluation results: **Verizon LTE Trace**



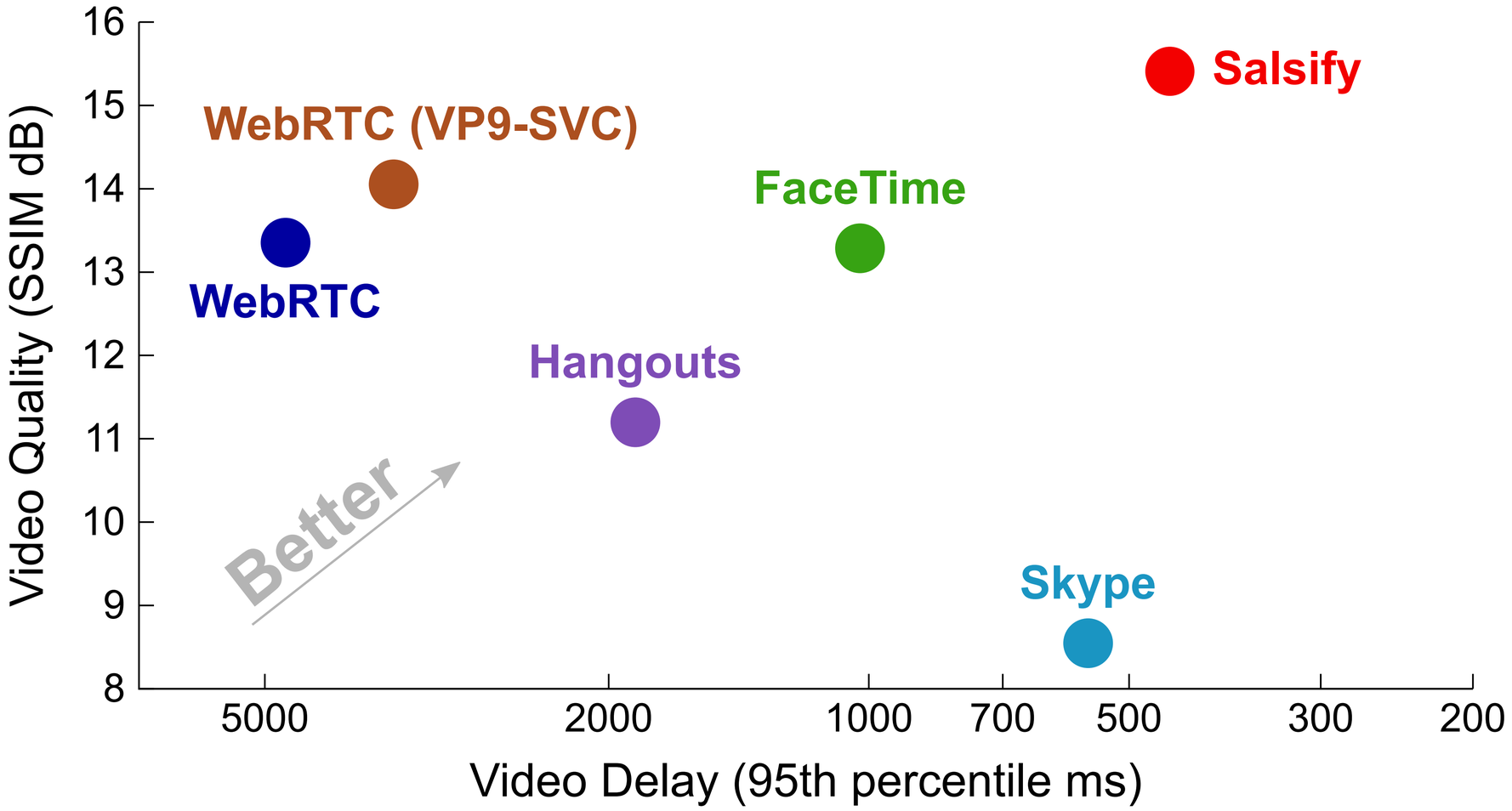
Evaluation results: **Verizon LTE Trace**



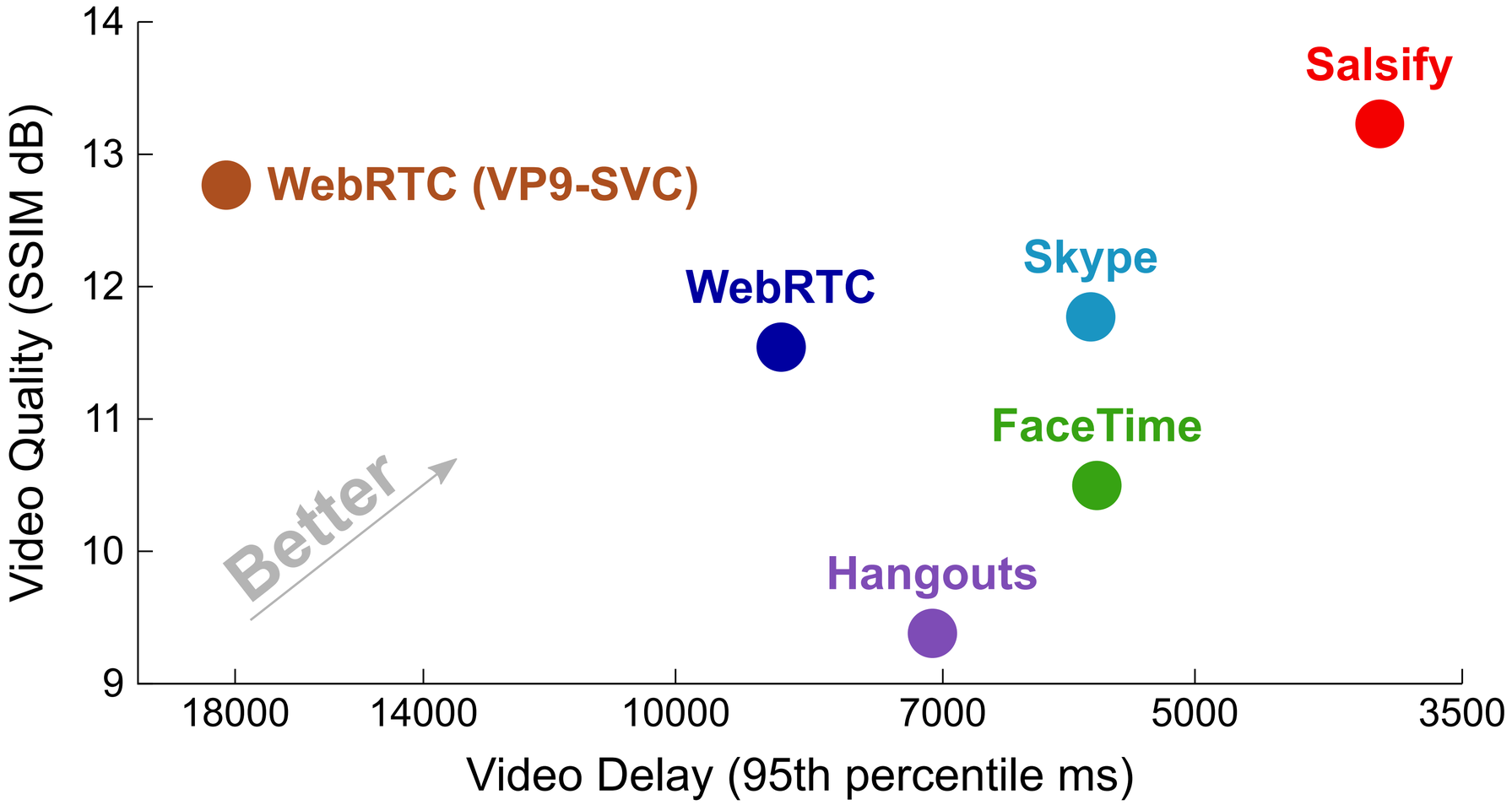
Evaluation results: **Grace Period**



Evaluation results: **AT&T LTE Trace**



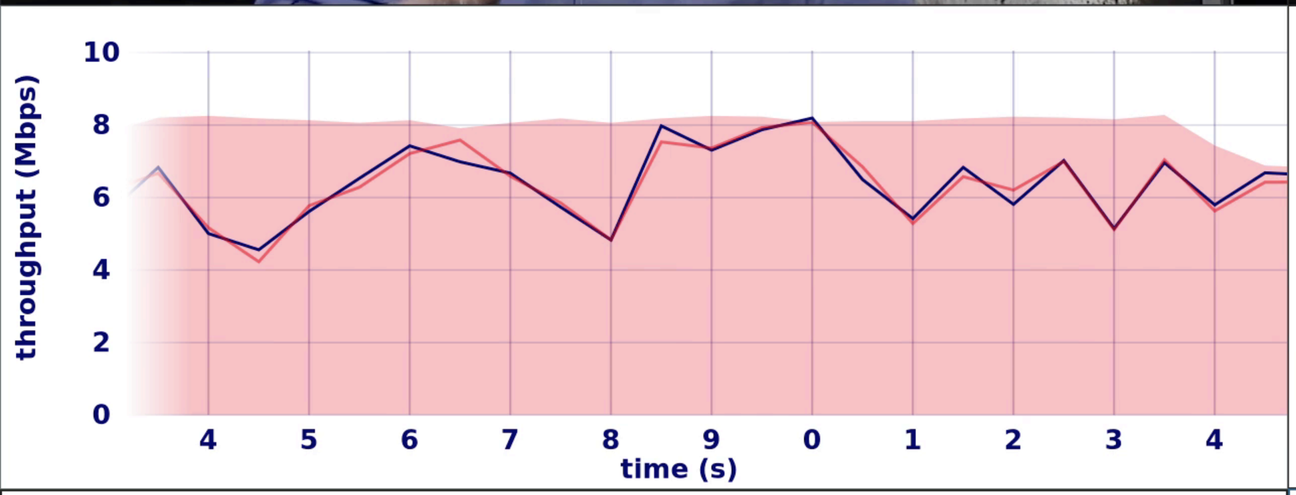
Evaluation results: **T-Mobile UMTS Trace**

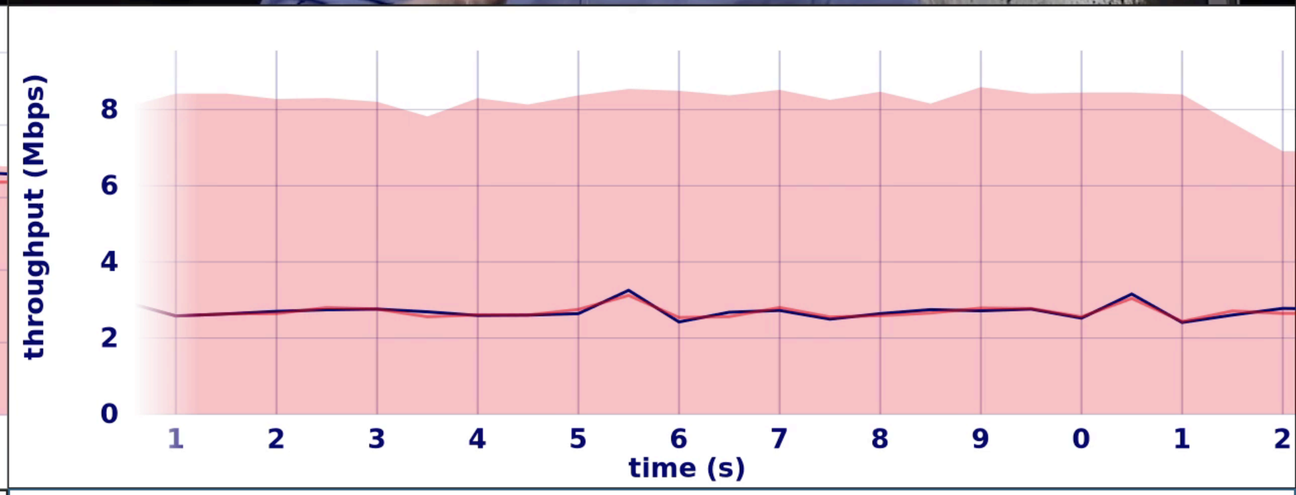




Network Variations

Watch the video at: https://snr.stanford.edu/salsify



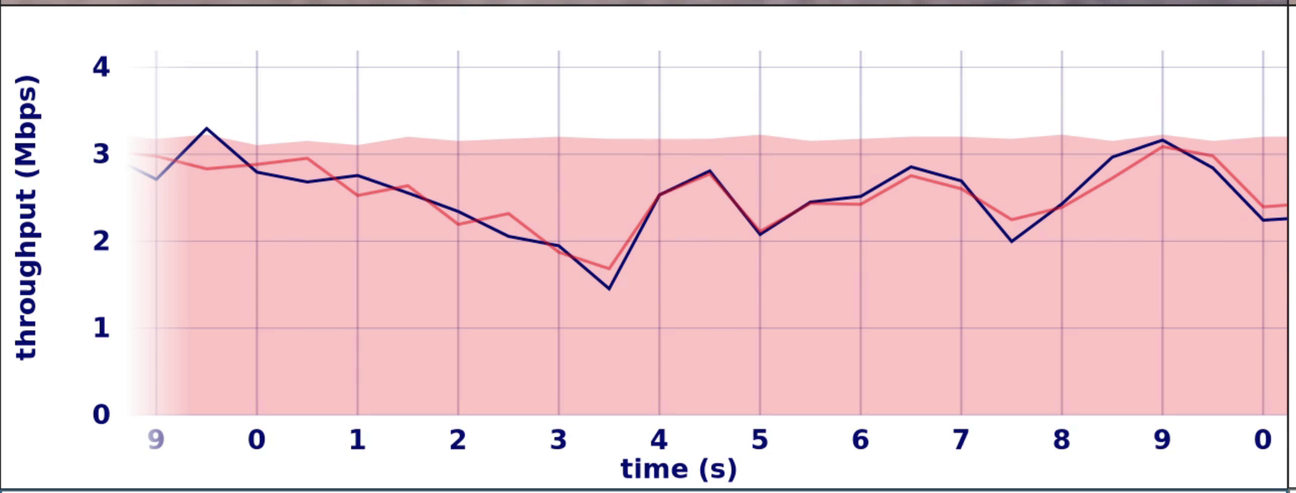


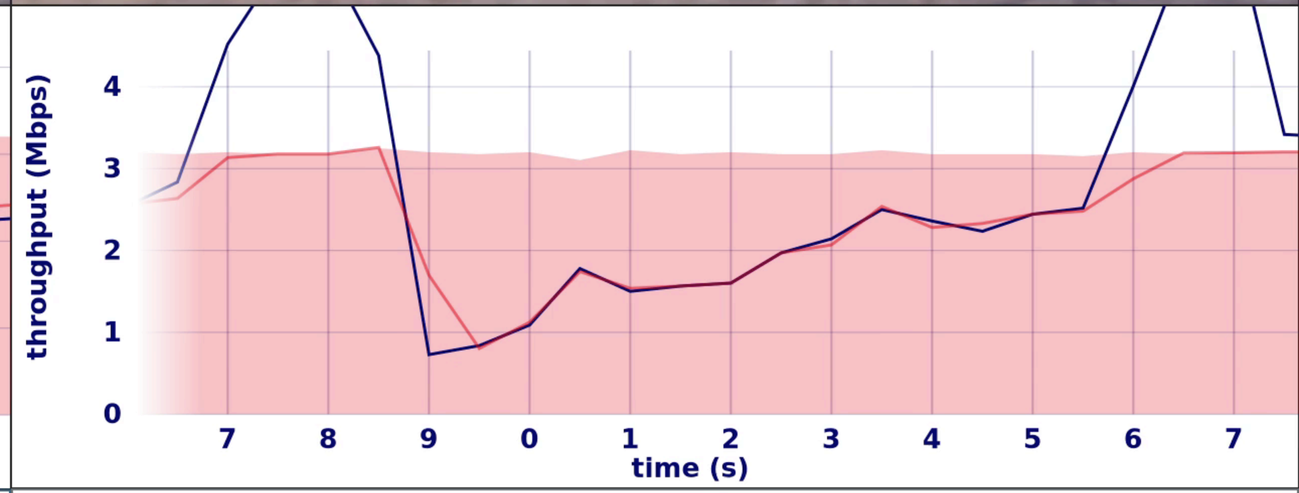
Salsify

WebRTC (Chrome 65)

Network Outages

Watch the video at: https://snr.stanford.edu/salsify





Salsify

WebRTC (Chrome 65)

Final Remarks

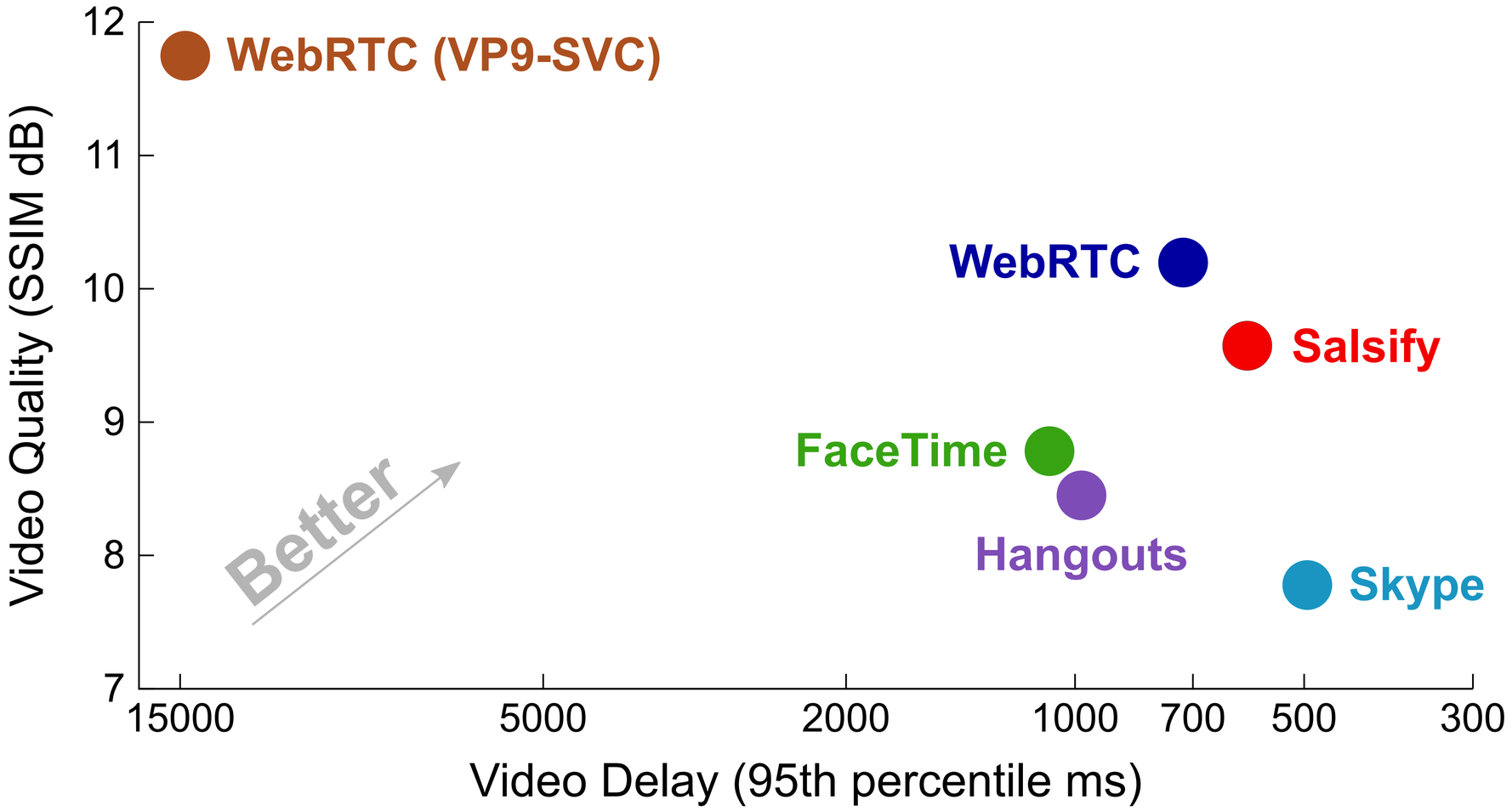
Where Salsify is not a good fit

- **Long latency budgets**
- Non-variable links
- Low-power devices

Where Salsify is not a good fit

- Long latency budgets
- **Non-variable links**
- Low-power devices

Where Salsify is not a good fit: **non-variable links**



Where Salsify is not a good fit

- Long latency budgets
- Non-variable links
- **Low-power devices**

Codecs have been treated as ***black boxes*** in video systems for a long time.

Takeaways

- Salsify is a new architecture for real-time Internet video.
  - Salsify tightly integrates a **video-aware transport protocol**, with a **functional video codec**, allowing it to **respond quickly to changing network conditions**.
- Changes to the architecture of systems can *sometimes* yield significant benefits.
- More info at: `snr.stanford.edu/salsify`

Thank you: NSF, DARPA, Google, Dropbox, VMware, Huawei, Facebook, Stanford Platform Lab, and James.



