Stateless encoders: VP8

Andrzej Pietrasiewicz
andrzej.p@collabora.com

Agenda

- Definitions
- uAPI (VP8 RFC)
- Rate control (VP8 RFC)
- Challenges

Definitions

Stateful vs Stateless

Stateful	**Stateless**
State kept and maintained **in** hardware	State kept and maintained **outside** hardware

So what?

Stateful vs Stateless

Stateful	Stateless
More complex hw	**Less complex hw**
Sw needs to interact with codec firmware	More registers to cope with
More expensive context change	**Less expensive context change**
Less flexibility	**More flexibility**

uAPI

First encoders

- 2020: H.264
  - https://github.com/bootlin/linux/tree/hantro/h264-encoding-v5.11
  - https://github.com/bootlin/v4l2-hantro-h264-encoder

- 2023: VP8
  - https://lore.kernel.org/linux-arm-kernel/20230309125651.23911-1-andrzej.p@collabora.com/T/
  - https://gitlab.collabora.com/linux/for-upstream/-/tree/vp8-rfc-v6.4-rc6
  - https://gitlab.freedesktop.org/gstreamer/gstreamer/-/merge_requests/3736

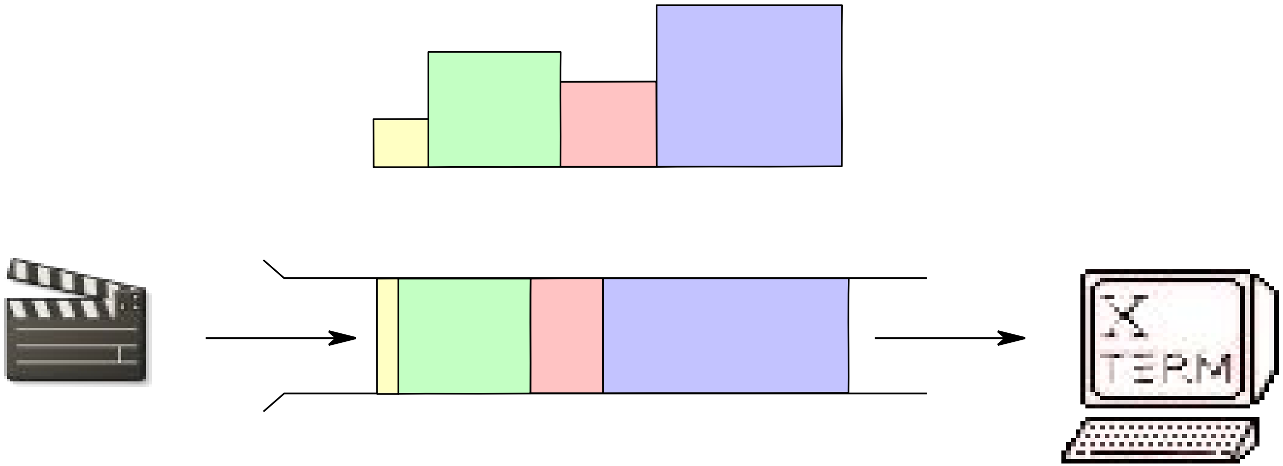
uAPI decisions (VP8 RFC)

- Request API
- Frame header built by the kernel
- GOP length decided by userspace
- Single reference frame (simple!)

Rate control

Rate control

Encoder perspective



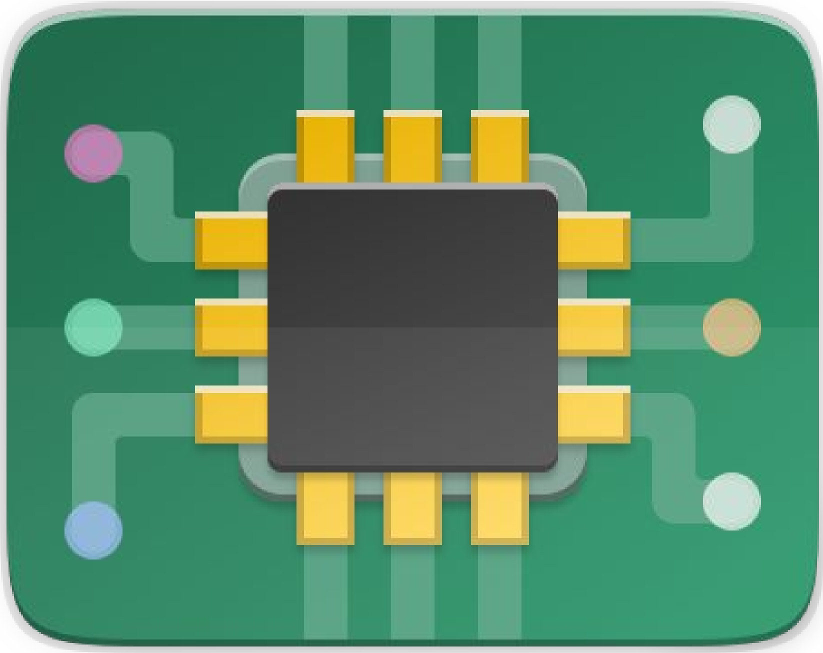
Decoder perspective

uAPI decisions (VP8 RFC)

- Per-frame constant QP
- Dedicated QP control

Challenges

- More VP8 hw?



Challenges

- How we review the new uAPI and drivers
- How (not?)to obey the “2 drivers rule”
- Where should frame header be assembled (kernel/user)?
- More than 1 reference frame
- Unify QP range?
- Compliance testing?

Challenges cont’d

- Rate control
  - Stateful!
  - But: opt-in for hardware-specific mechanisms
  - But: per-frame constant QP for userspace
- ROI from stateful uAPI?

AI?

- Possible use
  - Reference frame selection
  - Rate control
- Requirements
  - Looking at reconstructed buffer?

Thank you!