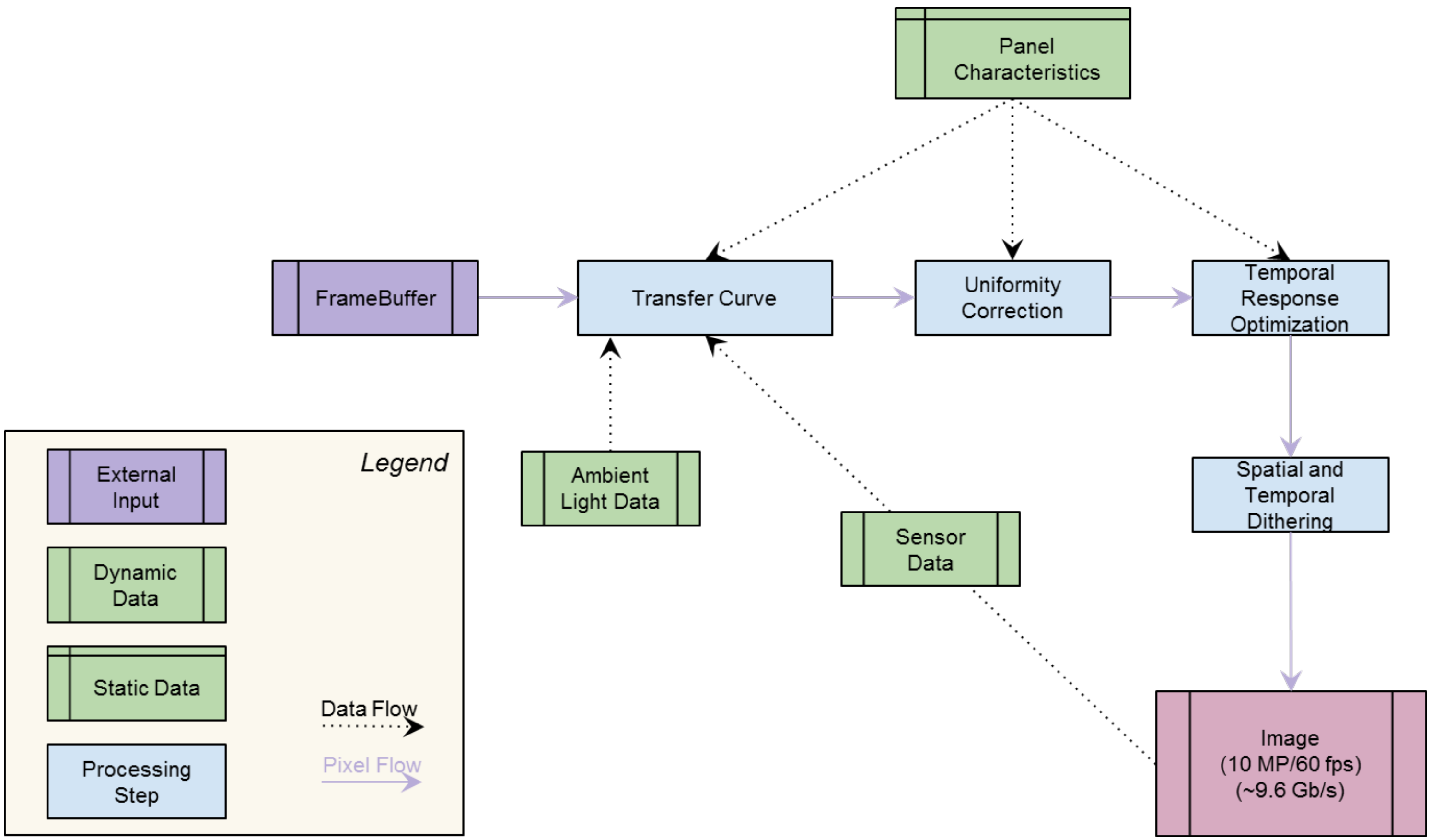
# Preliminary design and validation of a modular framework for predictable composition of medical imaging applications

7th July 2015

**Martijn van den Heuvel** – S.C. Cracana – H. Salunkhe – J.J. Lukkien – A. Lele – D. Segers

# Image display

- An input signal needs several transformations before being displayed

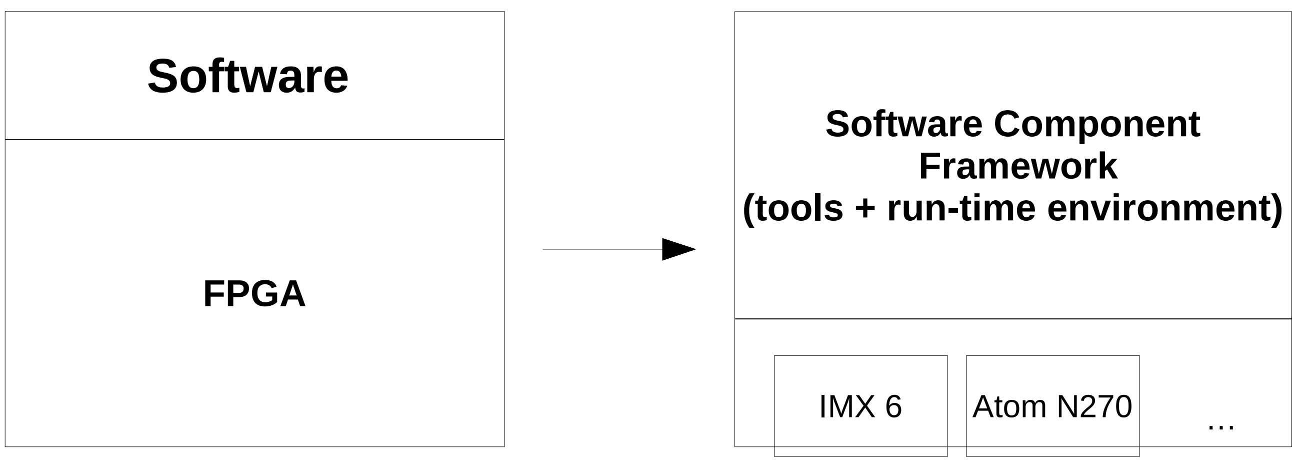



[1]

# Envisioned new design

**Main modifications**

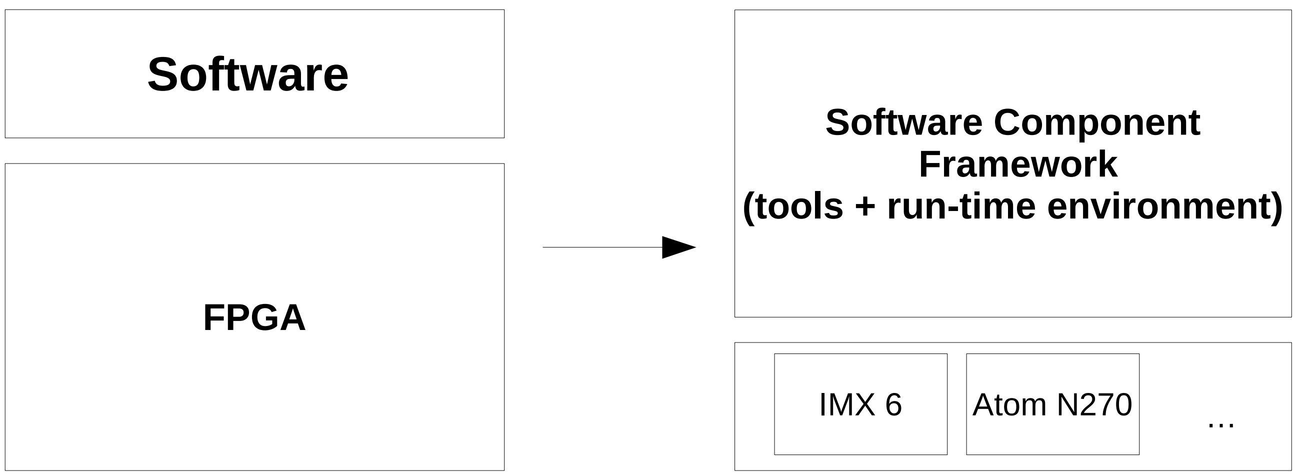
- Replace proprietary FPGAs by a COTS platforms
- Shift to a component-based software architecture



# Envisioned new design

**Main modifications**

- Replace proprietary FPGAs by a COTS platforms
- Shift to a component-based software architecture




Advantages

- Support for product variants
- Time-to-market:
  - Independent development & testing of components

# Problem Description

**Key issues for development**

- Resources
- Performance requirements
- Desired functionality

Manage variations at design time

# Problem Description

**Key issues for development**

- Resources
- Performance requirements
- Desired functionality

Manage variations at design time

**Goals:**

- COTS software framework
- Predictable framework configuration and performance metrics
- Validate predicted performance against run-time performance

**Build a prototype!**

# COTS Software – Logical View

User Interface

Image Processing Pipeline

Custom Algorithms

IMX 6

Atom N270

…

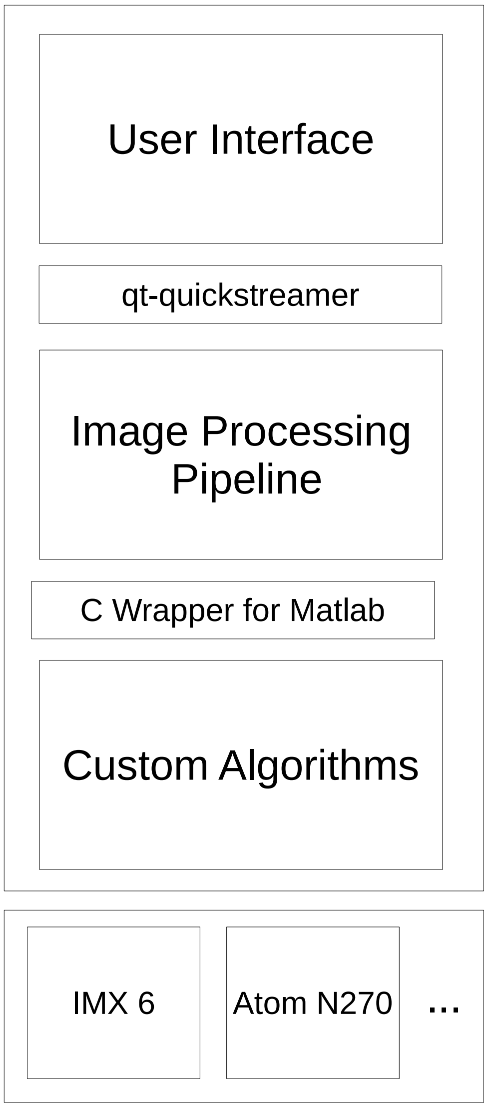
## Software

- Define interface in QML language
- Define pipeline architecture
- Proprietary algorithms for various signal transformations

## Hardware

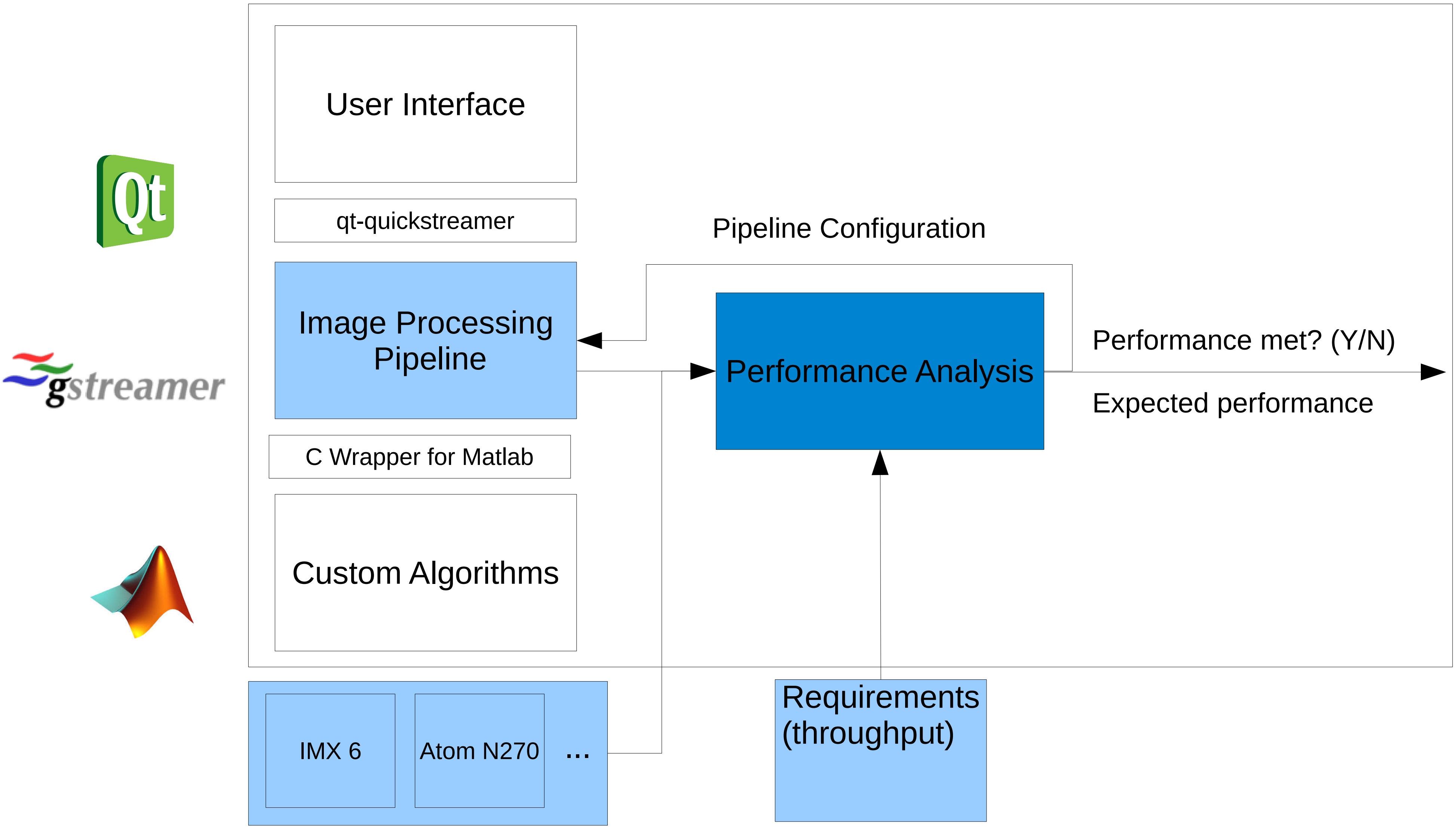
- Various COTS platforms

# COTS Software – new interfaces

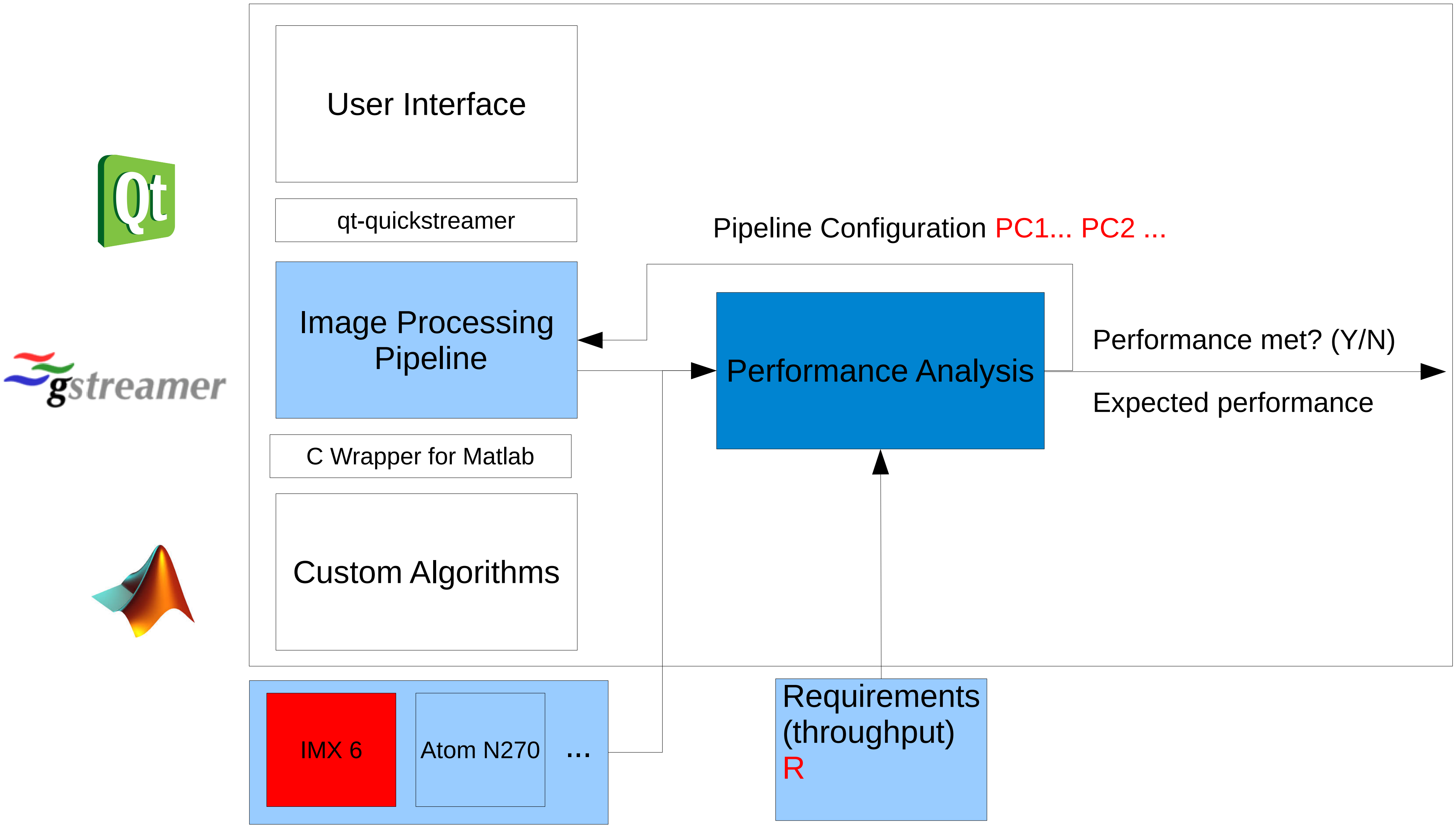



- Plugin for integrating Gstreamer into QML language => high level development
- Integrate Matlab code into Gstreamer => high reusability

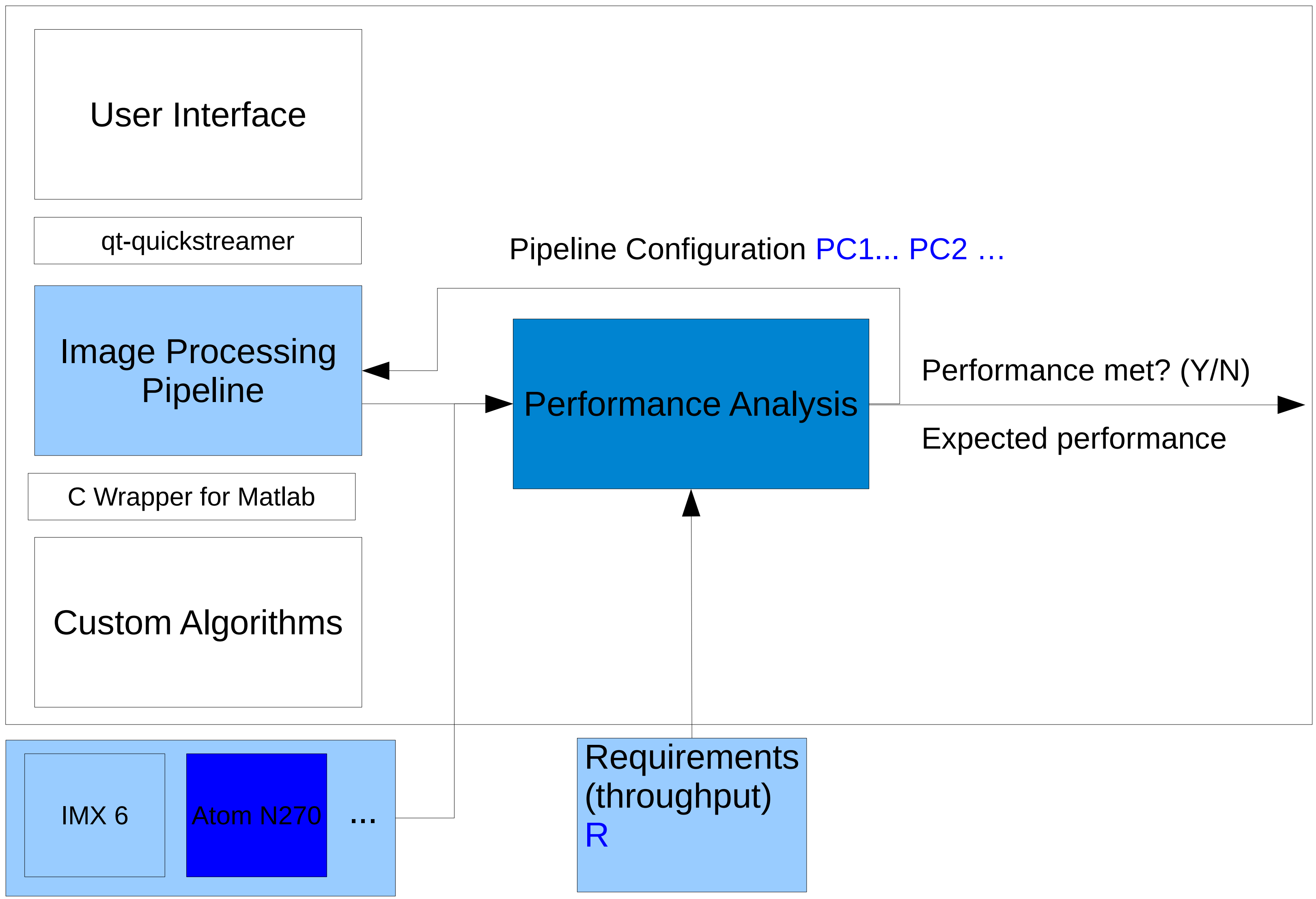
# Adding Performance Analysis



# Adding Performance Analysis and variability management

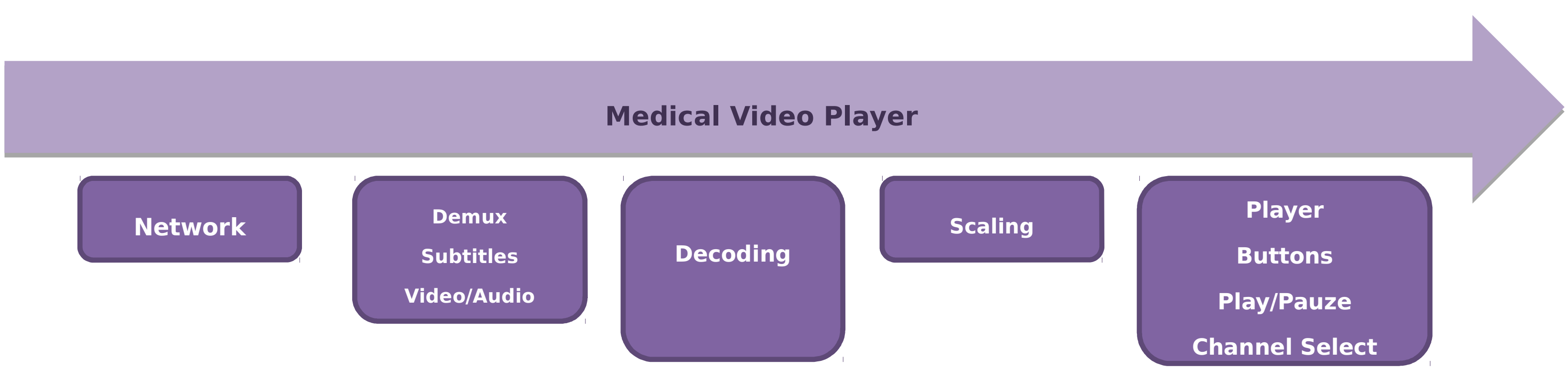


# Adding Performance Analysis and variability management



# Use case: applying the concepts

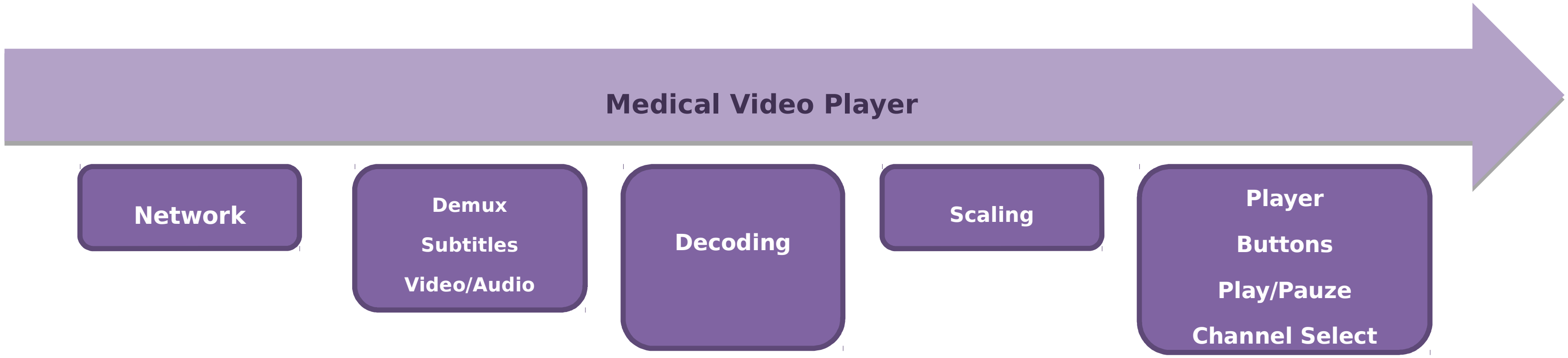
| | |
|---|---|
| Input video stream formats to be supported | SD - 576i: MPEG2 Transport Stream up to 6 Mbit/s<br>HD - 720p: H.264 Transport Stream up to 15 Mbit/s<br>HD - 1080i: H.264 Transport Stream up to 20 Mbit/s |
| Input video transport formats to be supported | Multicast UDP<br>RTP |
| Output screen resolutions | 1366x768<br>1920x1080 |
| Hardware | Intel Atom N270<br>Intel Cedarview D2550<br>Freescale iMX6 dual and quad core<br>Intel Baytrail DN2820 |




[1]

# Use case: Requirements analysis

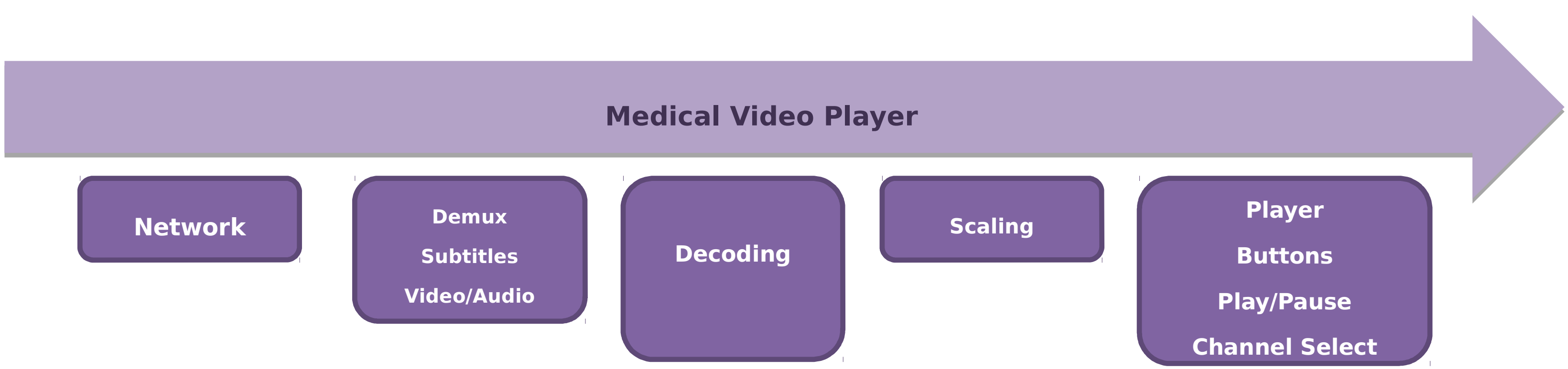
| | |
|---|---|
| Input video stream formats to be supported | SD - 576i: MPEG2 Transport Stream up to 6 Mbit/s<br>HD - 720p: H.264 Transport Stream up to 15 Mbit/s<br>HD - 1080i: H.264 Transport Stream up to 20 Mbit/s |
| Input video transport formats to be supported | Multicast UDP<br>RTP |
| Output screen resolutions | 1366x768<br>1920x1080 |
| Hardware | Intel Atom N270<br>Intel Cedarview D2550<br>Freescale iMX6 dual and quad core<br>Intel Baytrail DN2820 |




[2]

# Use case: variability

| | |
|---|---|
| Input video stream formats to be supported | SD - 576i: MPEG2 Transport Stream up to 6 Mbit/s<br>HD - 720p: H.264 Transport Stream up to 15 Mbit/s<br>HD - 1080i: H.264 Transport Stream up to 20 Mbit/s |
| Input video transport formats to be supported | Multicast UDP<br>RTP |
| Output screen resolutions | 1366x768<br>1920x1080 |
| Hardware | Intel Atom N270<br>Intel Cedarview D2550<br>Freescale iMX6 dual and quad core<br>Intel Baytrail DN2820 |




[2]

# Research questions

1. How to model GStreamer pipelines?

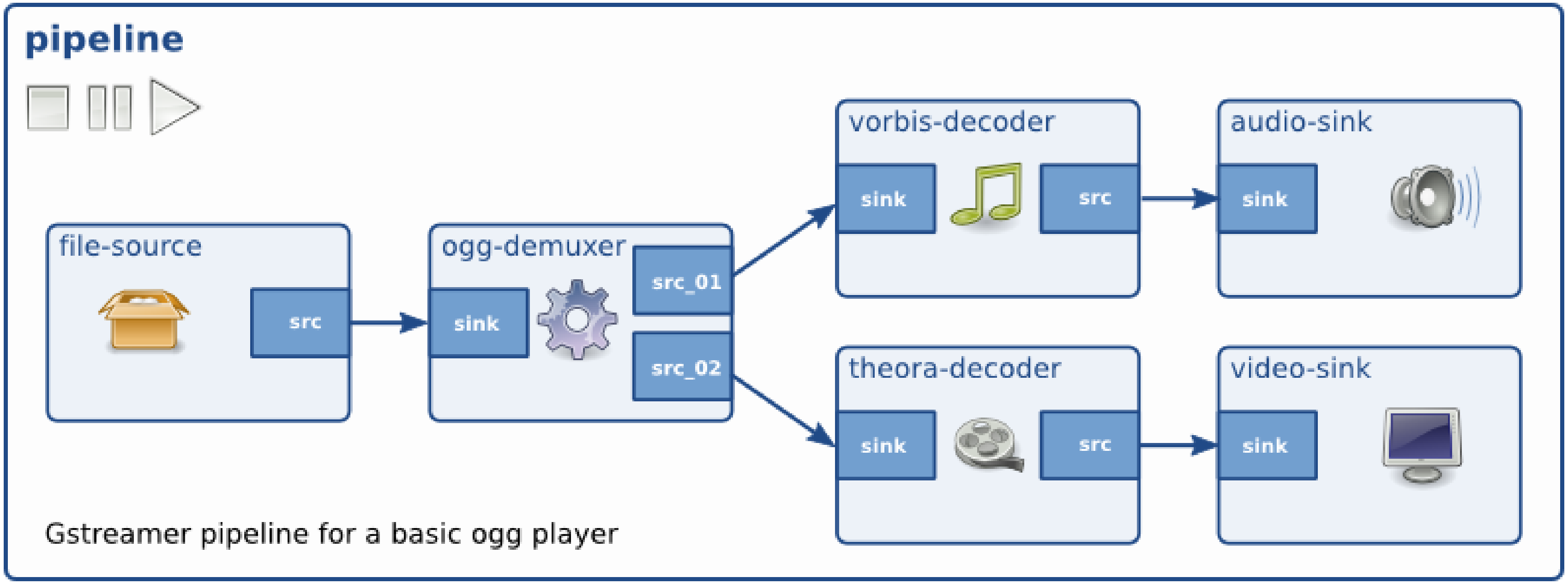
2. What are the required inputs for performance analysis?

3. What is the mapping between an GStreamer pipeline and a performance model?

4. What are the key configuration parameters of a GStreamer pipeline?

# GStreamer Pipeline Architecture

- A number of *plugins* can be connected to attain the requisite media processing
- The processing unit in GStreamer is called a *pipeline*
- It handles the clocking, the synchronizations, scheduling and the control message flow between elements




[3]

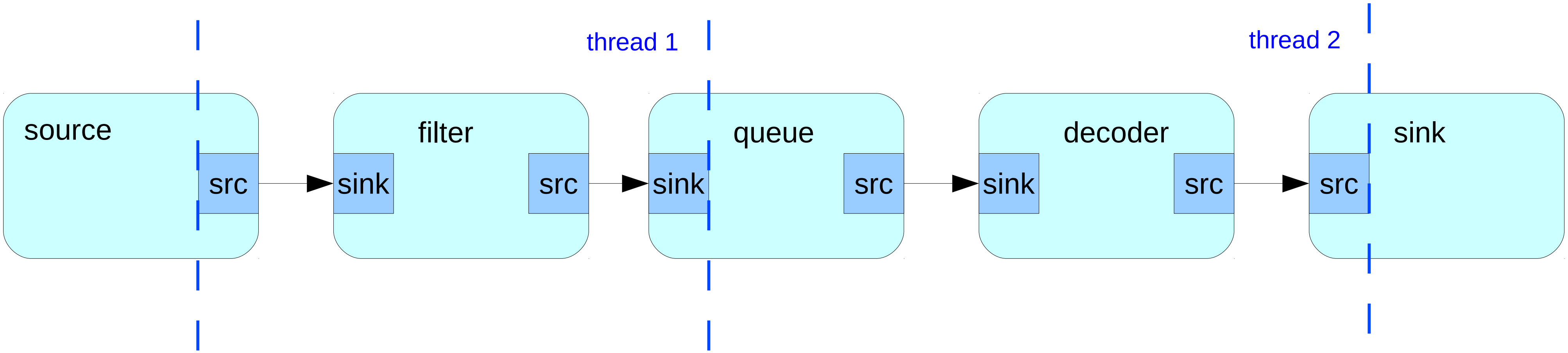
# Important GStreamer elements

Buffers

- Media content passed between elements
- May have different sizes

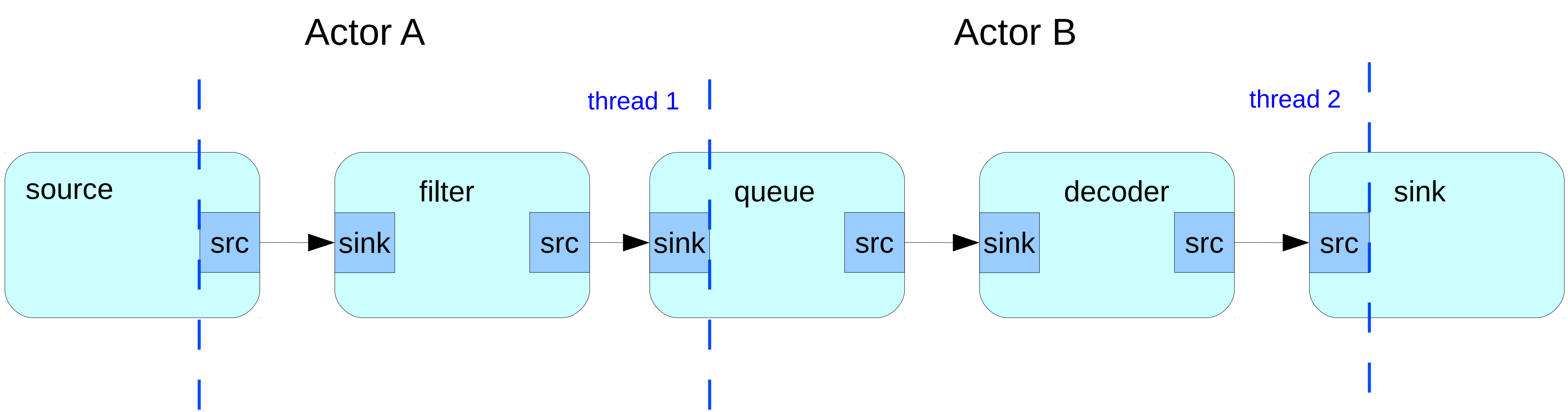
Queues

- Represent thread boundaries
- Enable/disable back pressure (i.e., write protection)

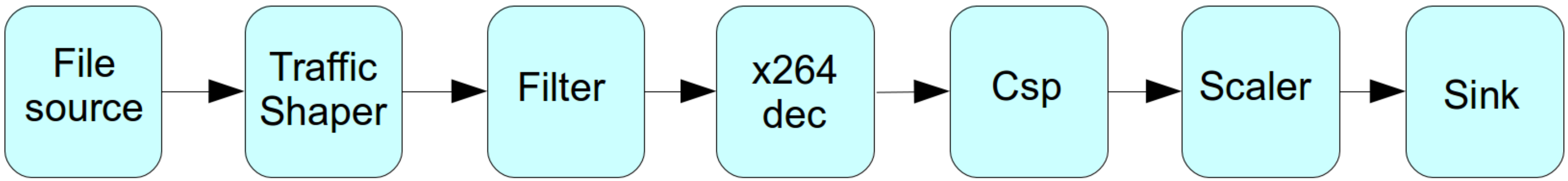


# Mapping of Gstreamer and Synchronous Data-Flow models

| SDF element | Gstreamer equivalent | Description |
|---|---|---|
| Actor | Set of linked elements running on same thread | Functionality, code to be executed |
| Token | Buffer | Data units |
| Channels | Pad links | Data dependencies/execution order |
| Rate | #buffers pushed/popped | Data units consumed/produced |



# Reference pipeline

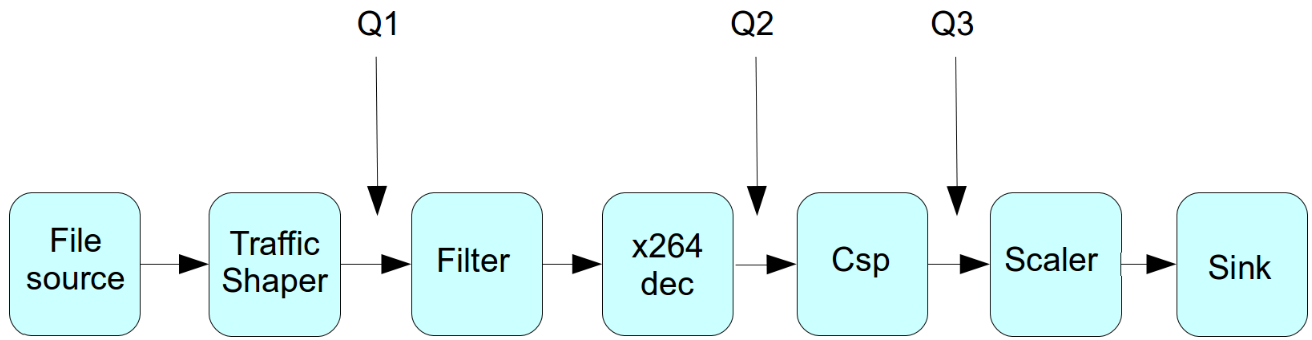



**Traffic shaper:**

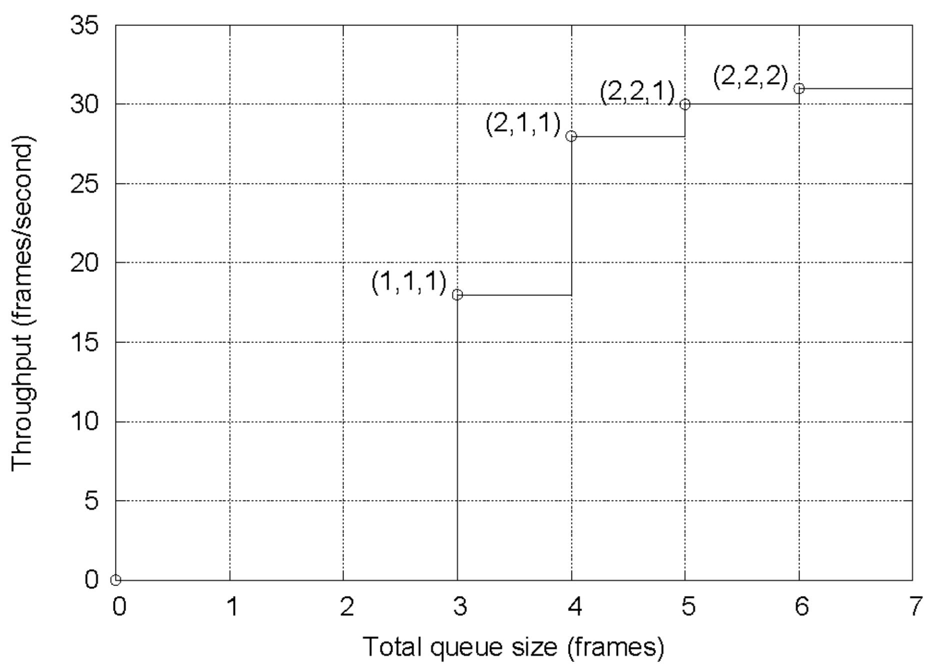
- ensures that a data unit is equal to a video frame;

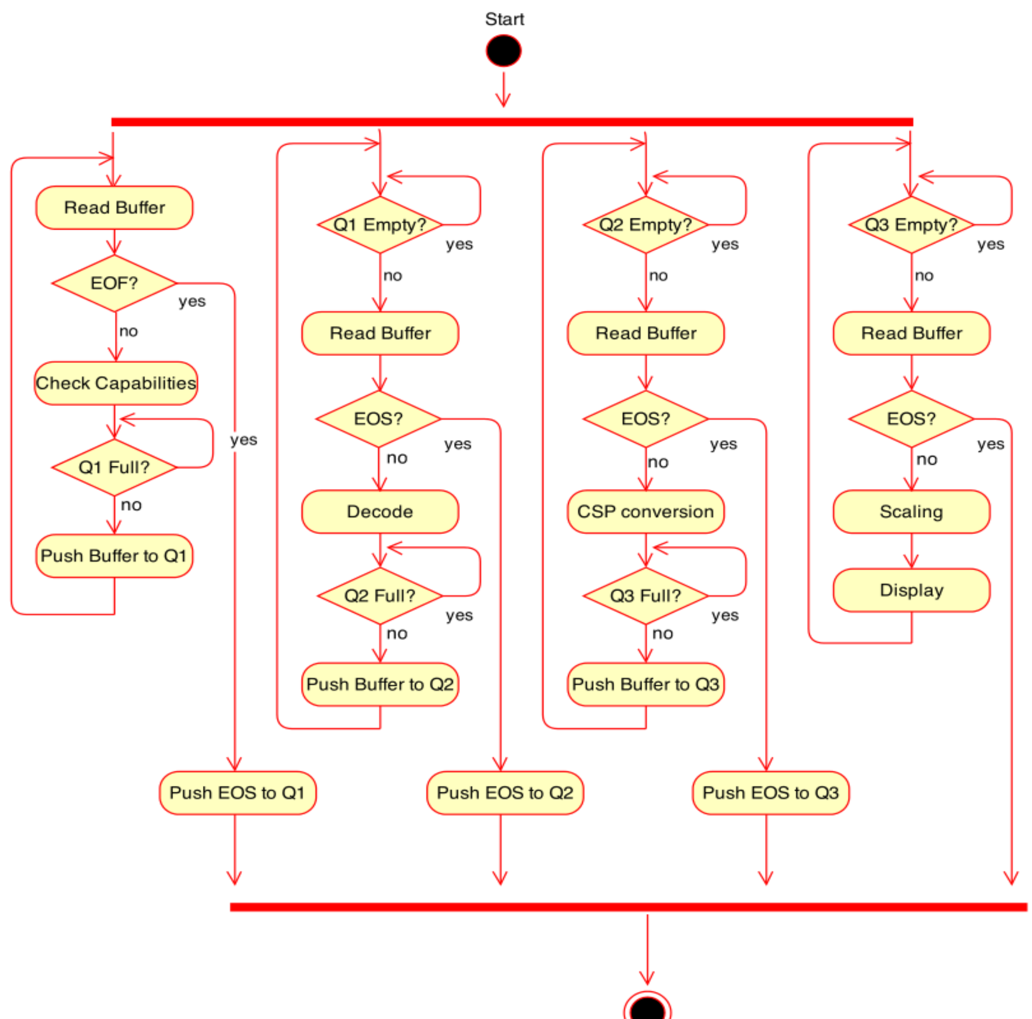
- **If** no back-pressure (data may be overwritten), **then** limit data rate of the source

# Protect overwrites with back pressure

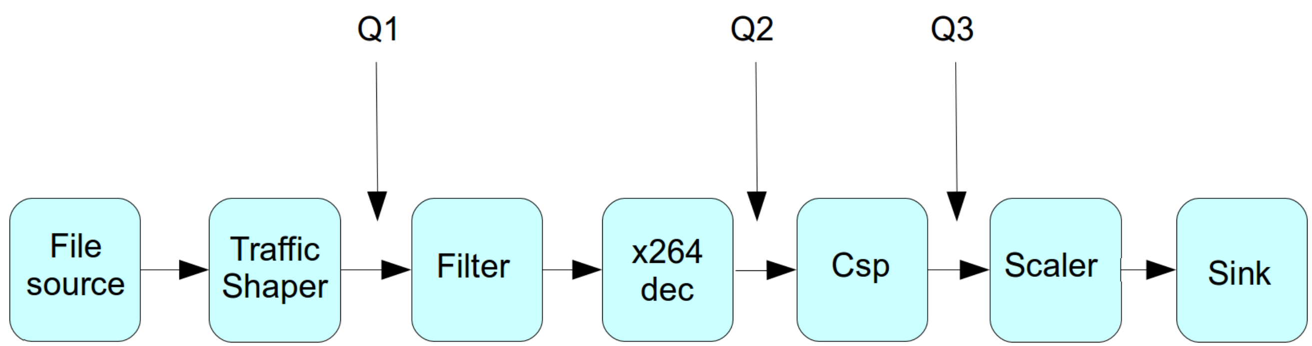



## Buffer sizing vs. performance



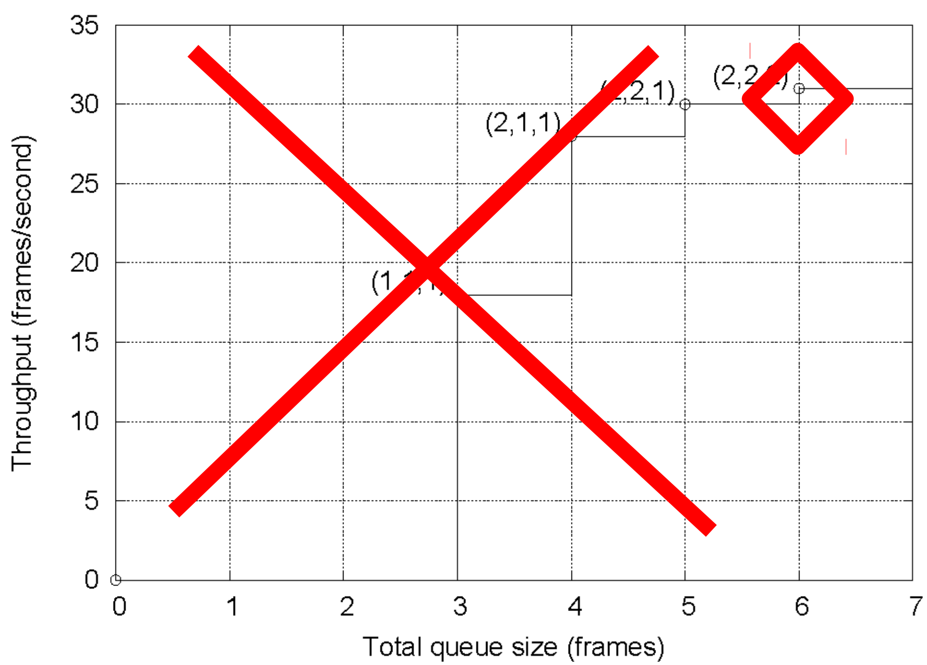



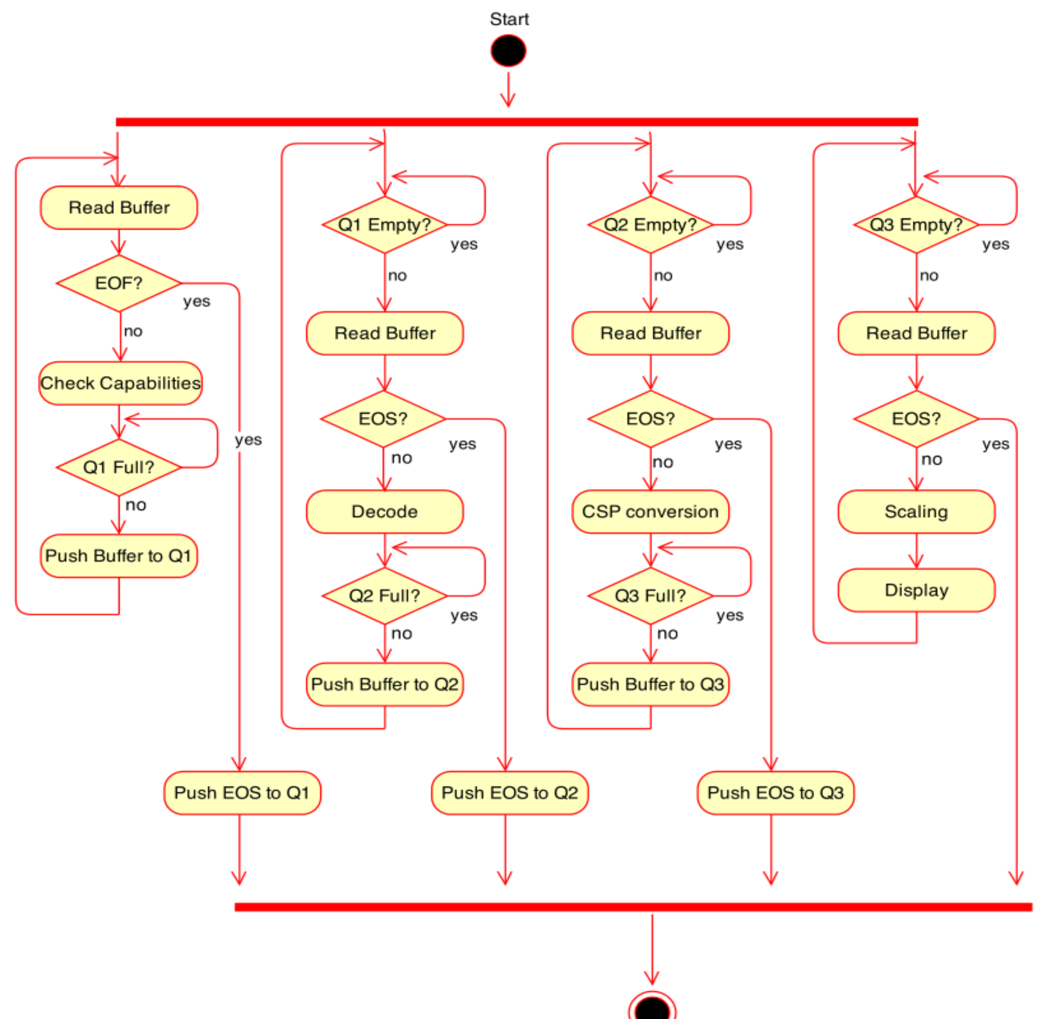
# No back pressure



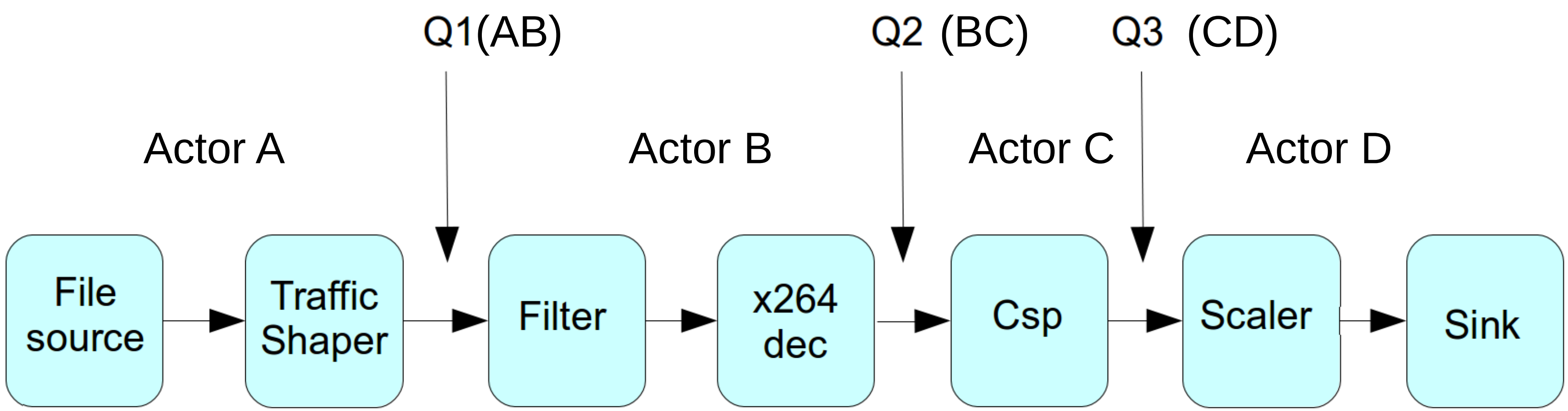

## Largest buffer sizing!

### Data loss vs. highest throughput:



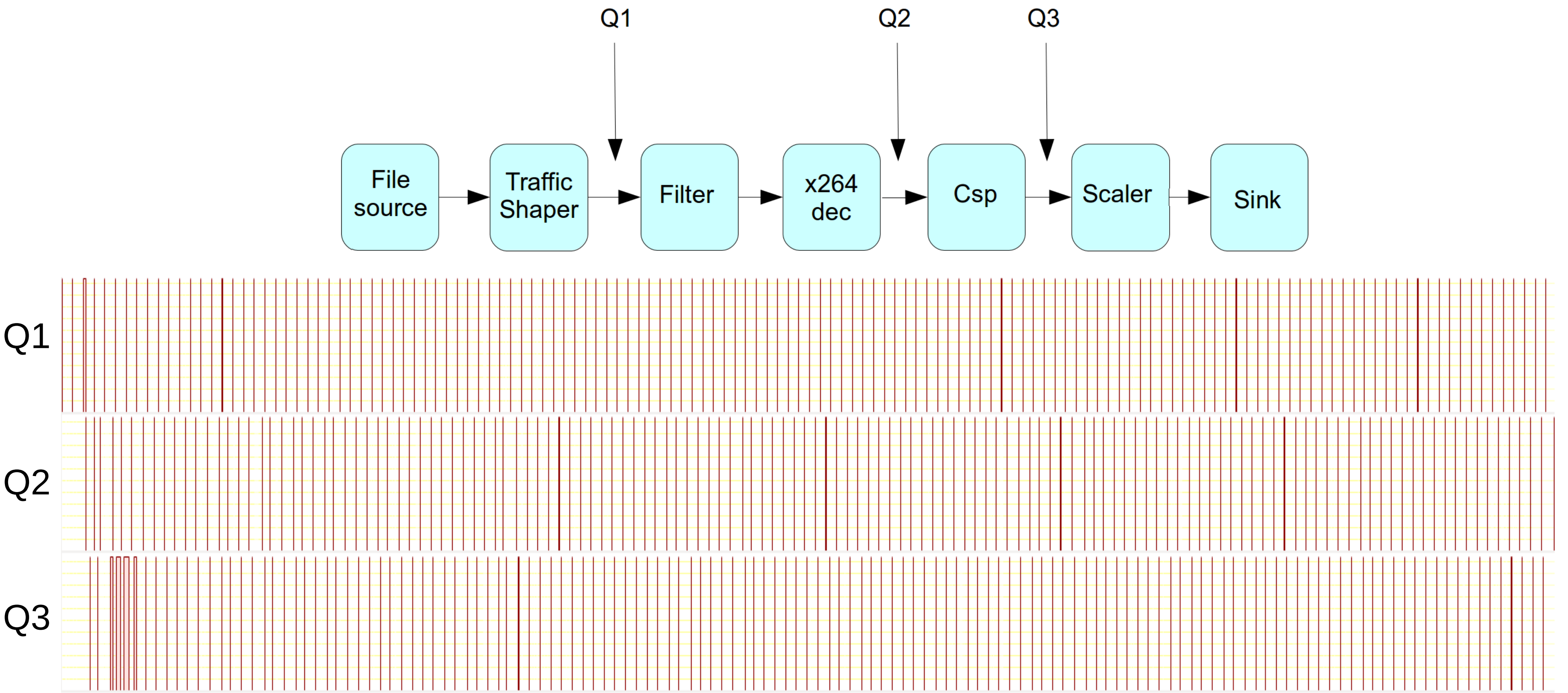
# Performance analysis vs. measures
## (with vs. without back pressure)




| Back Pressure | Distribution | Worst case Run-time Memory usage | Predicted Throughput (fps) | Average Run-time Throughput (fps) |
|---|---|---|---|---|
| Enabled | (2,1,1) | (1,1,1) | 28 | 31 |
| Disabled | (2,2,2) | (1,1,2) | 31 | 31 |

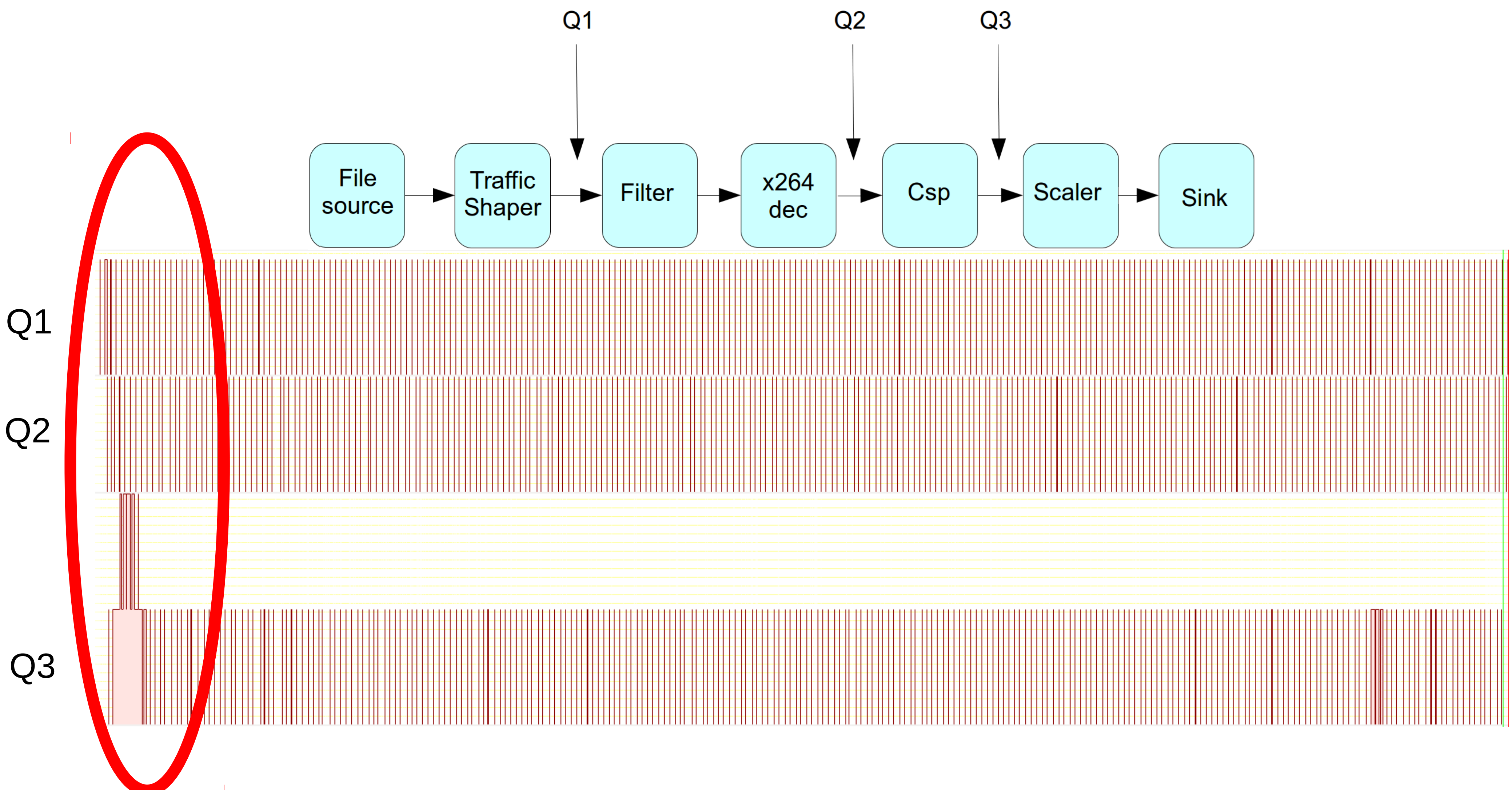
# Run-time analysis of memory usage (back-pressure enabled)

- Run-time monitoring of push/pop events on buffers
- Visualization using Time Doctor (http://sourceforge.net/projects/timedoctor/)



# Run-time analysis of memory usage (back-pressure disabled)

- Run-time monitoring of push/pop events on buffers
- Visualization using Time Doctor (http://sourceforge.net/projects/timedoctor/)



# Conclusions

Current work

- Investigate
  - COTS software framework
  - Predictable framework configuration and performance metrics

- Prototyping:
  - predicted performance against run-time performance

Future work

- Complex pipelines (split and joins)
- GStreamer scalability
- Advanced platform models (processor mappings, caches, etc.)