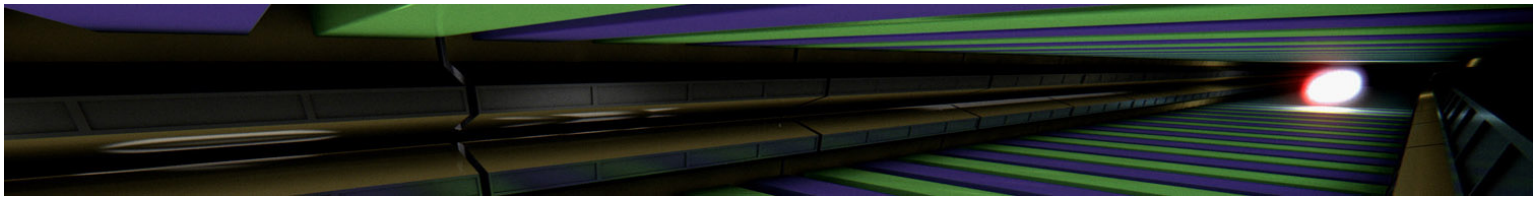
European XFEL



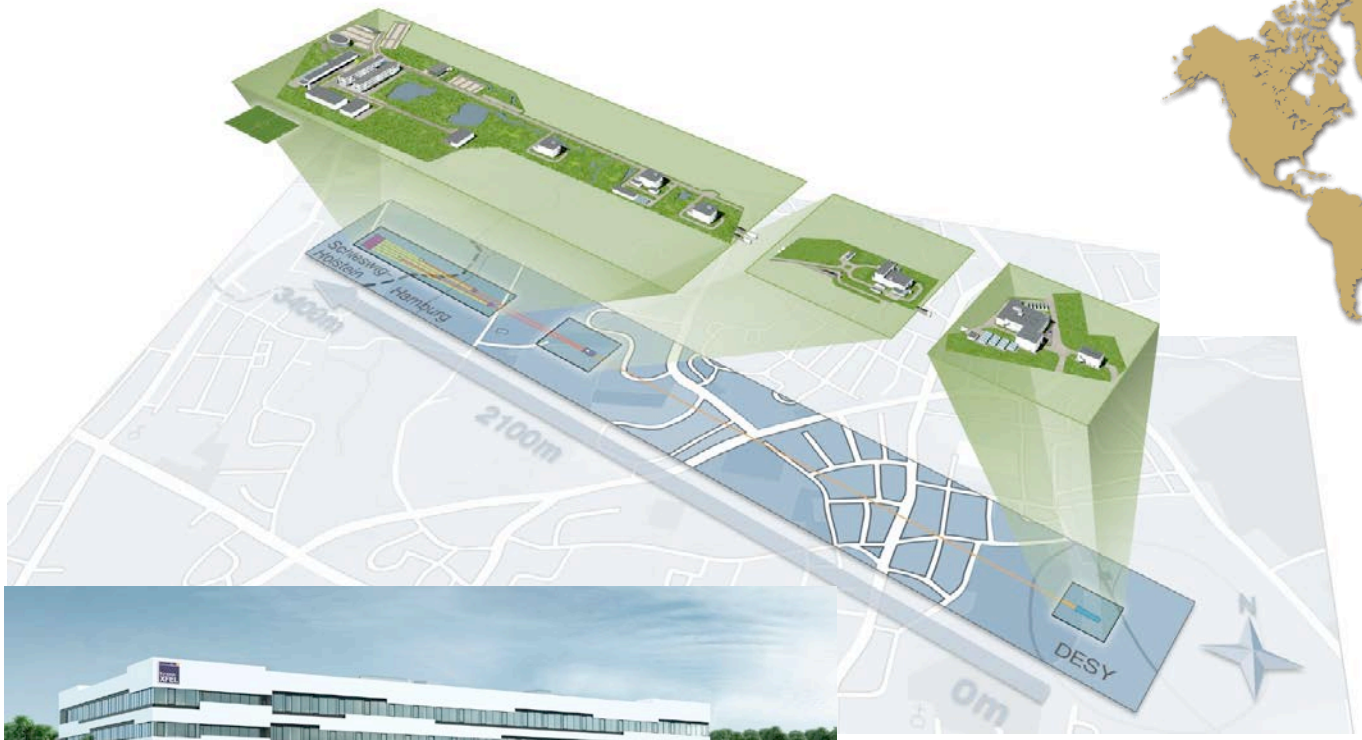
Karabo

An integrated software framework combining control, data management, and scientific computing tasks

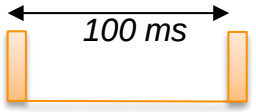
Burkhard Heisen

October 11, ICALEPCS 2013

Karabo will be used at the European XFEL

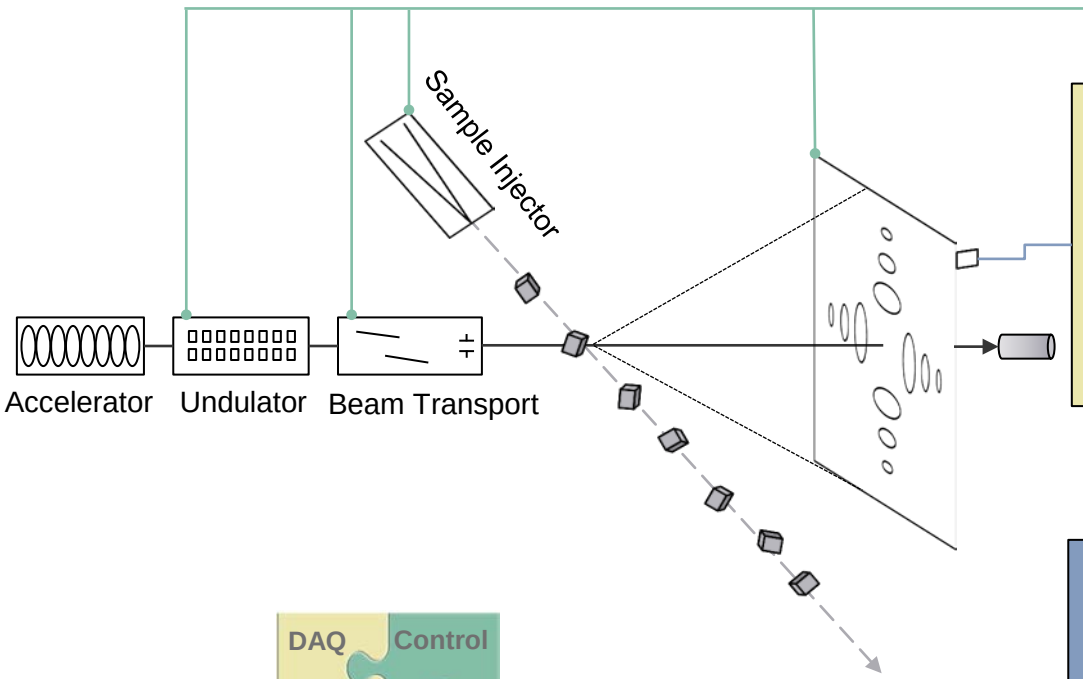


- Data rate (10 GB/s/2d-detector)
- Bunch pattern (4.5 MHz pulses @ 10Hz)



Functional requirements

A typical use case:



Control
drive hardware and complex experiments
monitor variables & trigger alarms

DAQ
data readout
online processing
quality monitoring (vetoing)

allow some control & show hardware status

show online data whilst running

DM
storage of experiment & control data
data access, authentication authorization etc.

setup computation & show scientific results

SC
processing pipelines
distributed and GPU computing
specific algorithms (e.g. reconstruction)

Tight integration of applications

European XFEL

What is Karabo? - Lets draw an analogy

Available on the Android App Store

Karabo devices

android

Karabo framework

Karabo device-server running devices

What is Karabo? - Lets draw an analogy

Karabo devices

Karabo framework

Karabo device-server running devices

What is Karabo? - Lets draw an analogy

Karabo devices

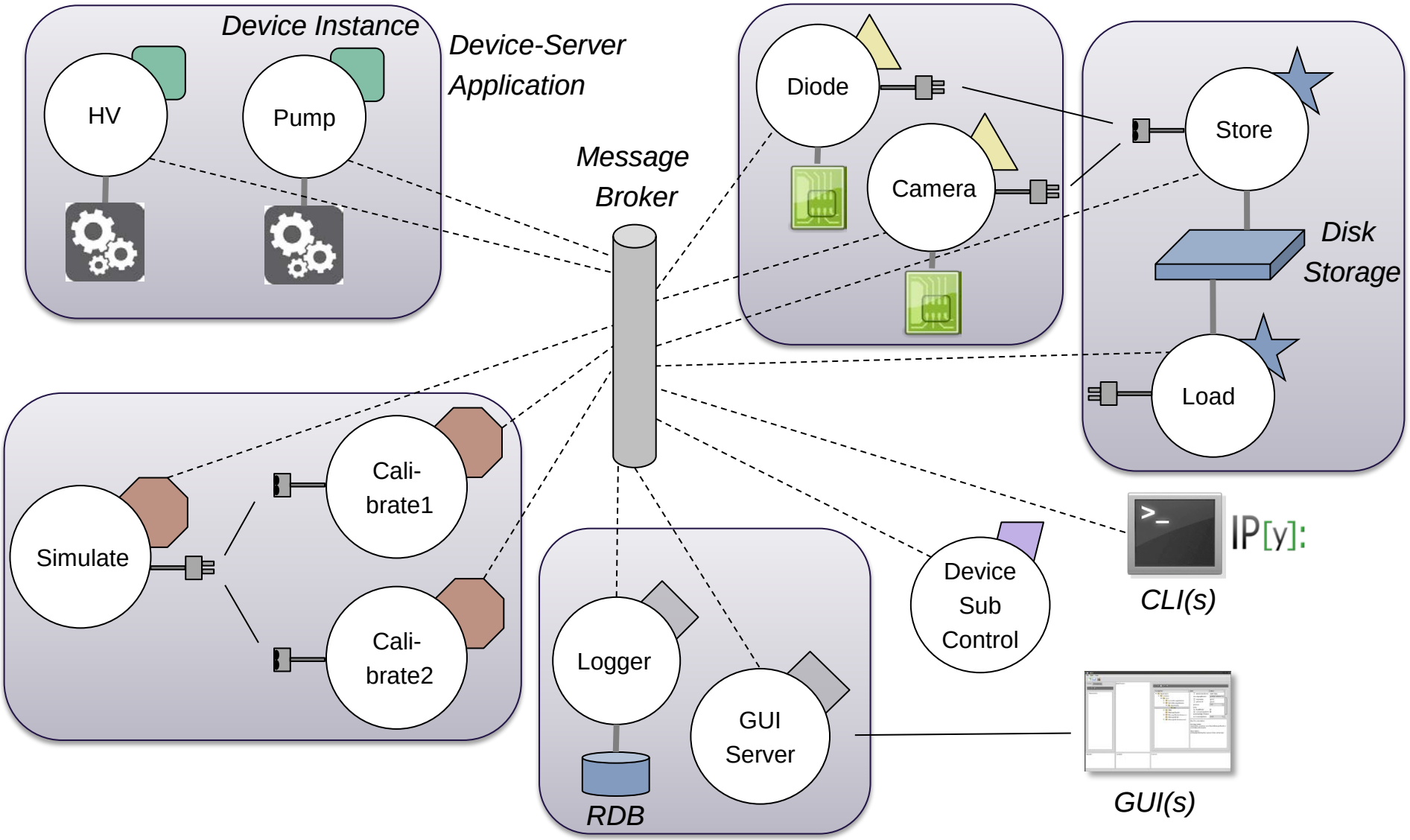
Karabo central services (archive, database)

Karabo device-server running devices

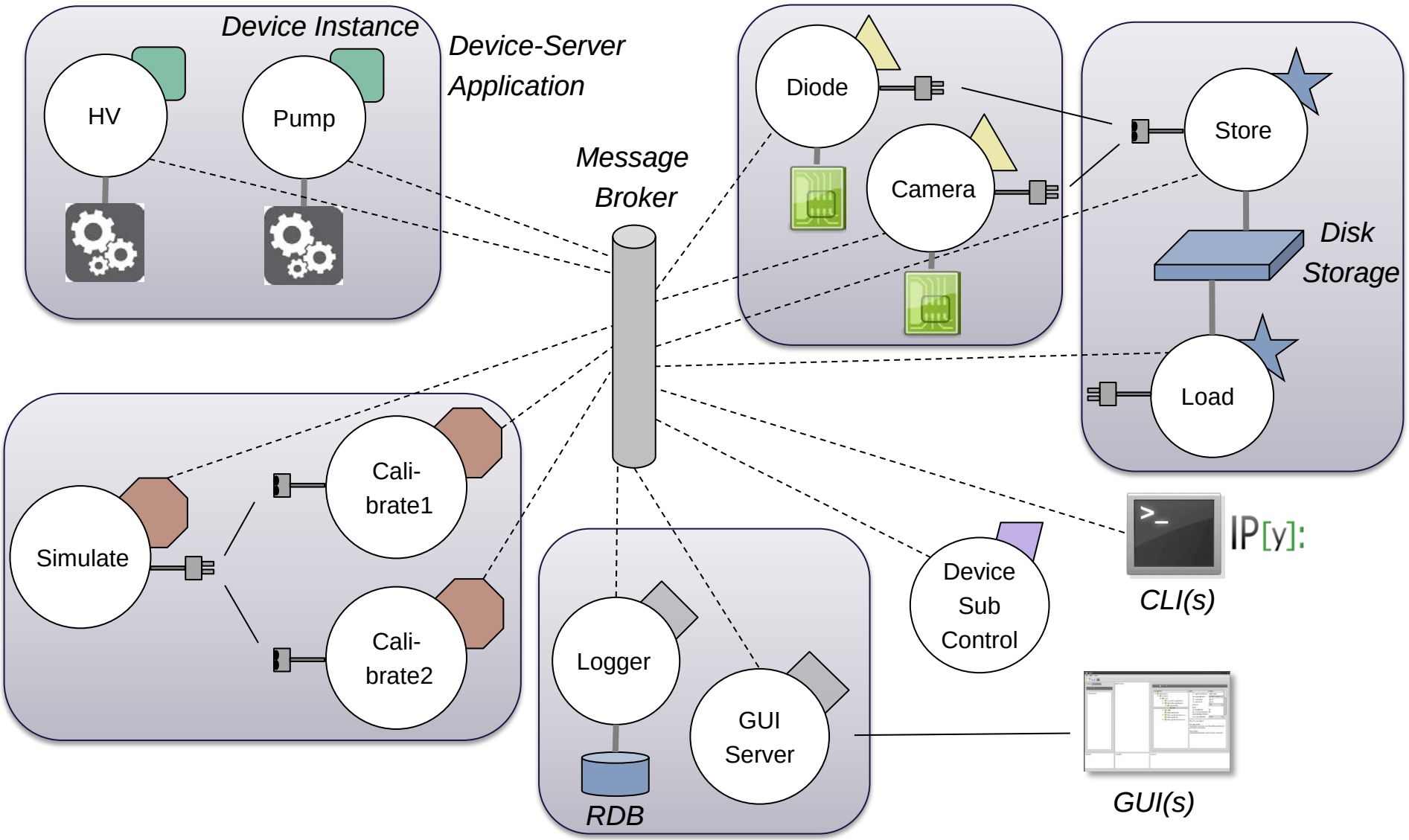
Karabo framework

European XFEL

Essential components

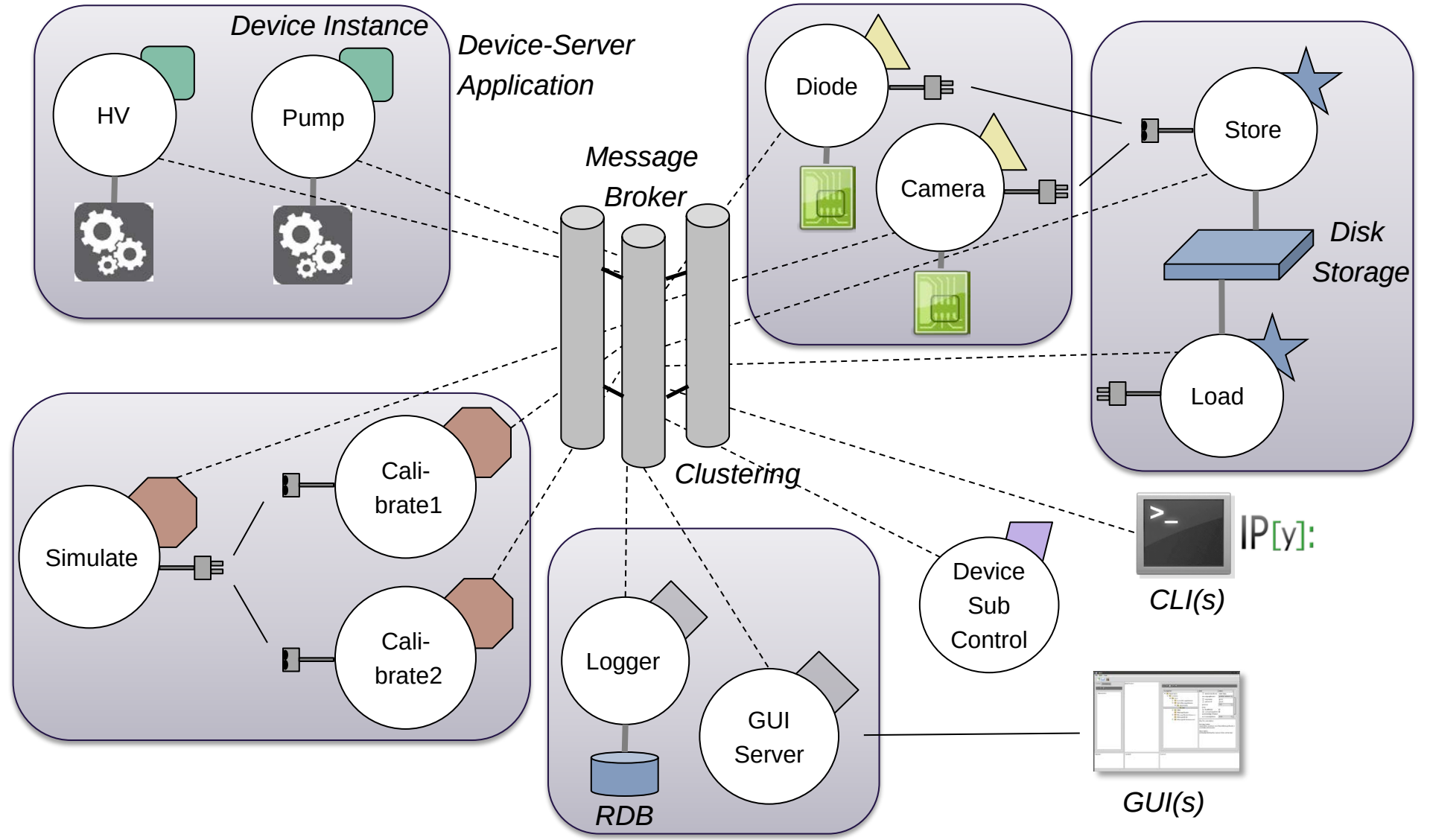


Essential components



European XFEL

Essential components



Device – Properties and commands

- Devices are **controllable objects** managed by a device server
- Device classes can be loaded at runtime (**plugin technology**)
- Can be written in **C++** or **Python**
- Devices completely **describe themselves**. Allows automatic GUI creation and auto-completion in IPython
- **Runtime-extension** of properties, commands and attributes is possible
- **No need** for device developers **to validate** any parameters. This is internally done taking the expectedParameters as a white-list
- Properties and commands can be nested, such that **hierarchical groupings** are possible

Class: MotorDevice

```
static expectedParameters( Schema& s ) {
  FLOAT_ELEMENT(s).key("velocity")
    .description("Velocity of the motor")
    .unitSymbol("m/s")
    .assignmentOptional().defaultValue(0.3)
    .maxInc(10)
    .minInc(0.01)
    .reconfigurable()
    .allowedStates("Idle")
    .commit();

  INT32_ELEMENT(s).key("currentPosition")
    .description = "Current position of the motor"
    .readOnly()
    .warnLow(10)
    […]

  SLOT_ELEMENT(s).key("move")
    .description = "Will move motor to target position"
    .allowedStates("Idle")
    […]
}

// Constructor with initial configuration
MotorDevice( const Hash& config ) { […] }
```

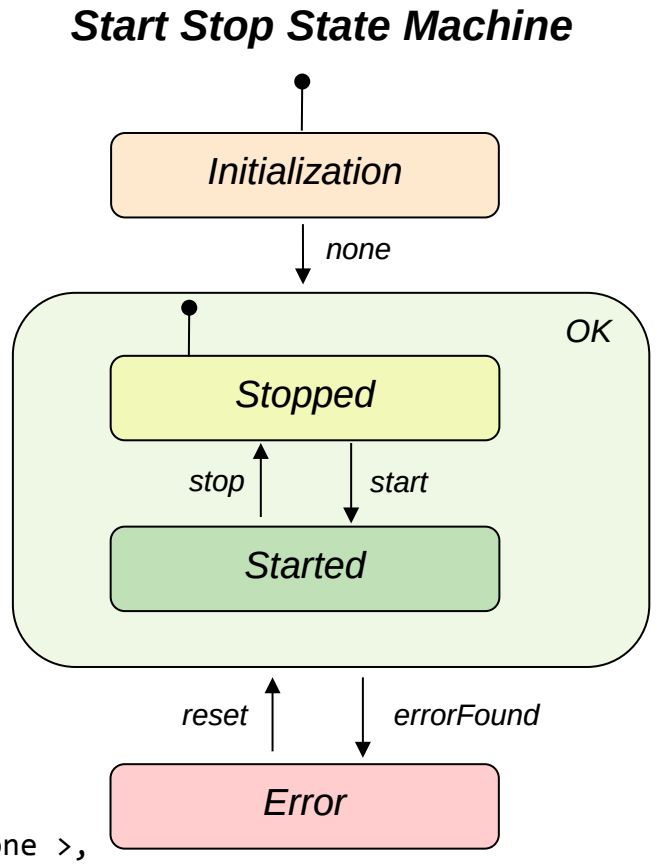
Property

Attribute

Command

Device – Finite state machine (FSM)

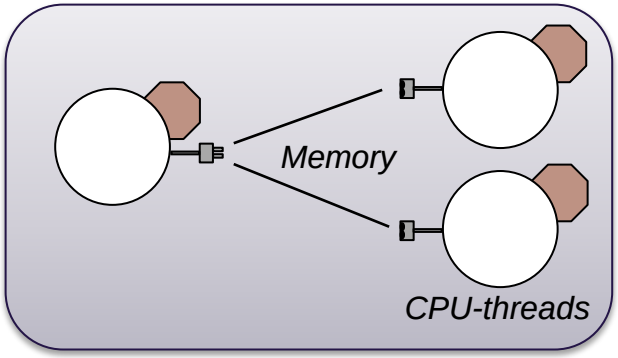
- **Any device uses a standardized way to express its possible program flow**
  - The state machine calls back device functions (guard, onStateExit, action, onStateEntry)
  - The GUI is state-machine aware and enables/disables buttons proactively

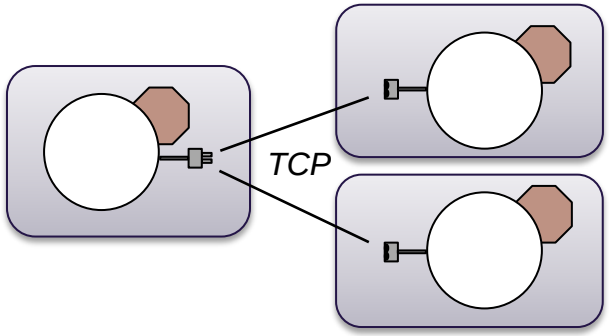



```
// Ok Machine
FSM_TABLE_BEGIN(OkTransitionTable)
//   SrcState   Event      TgtState   Action     Guard
Row< Started, StopEvent,  Stopped, StopAction,  none >,
Row< Stopped, StartEvent, Started, StartAction, none >
FSM_TABLE_END
FSM_STATE_MACHINE(Ok, OkTransitionTable, Stopped, Self)

// Top Machine
FSM_TABLE_BEGIN(TransitionTable)
Row< Initialization, none,           Ok,    none,             none >,
Row< Ok,             ErrorFoundEvent, Error, ErrorFoundAction, none >,
Row< Error,          ResetEvent,      Ok,    ResetAction,      none >
FSM_TABLE_END
KARABO_FSM_STATE_MACHINE(StateMachine, TransitionTable, Initialization, Self)
```

Device – As workflow module

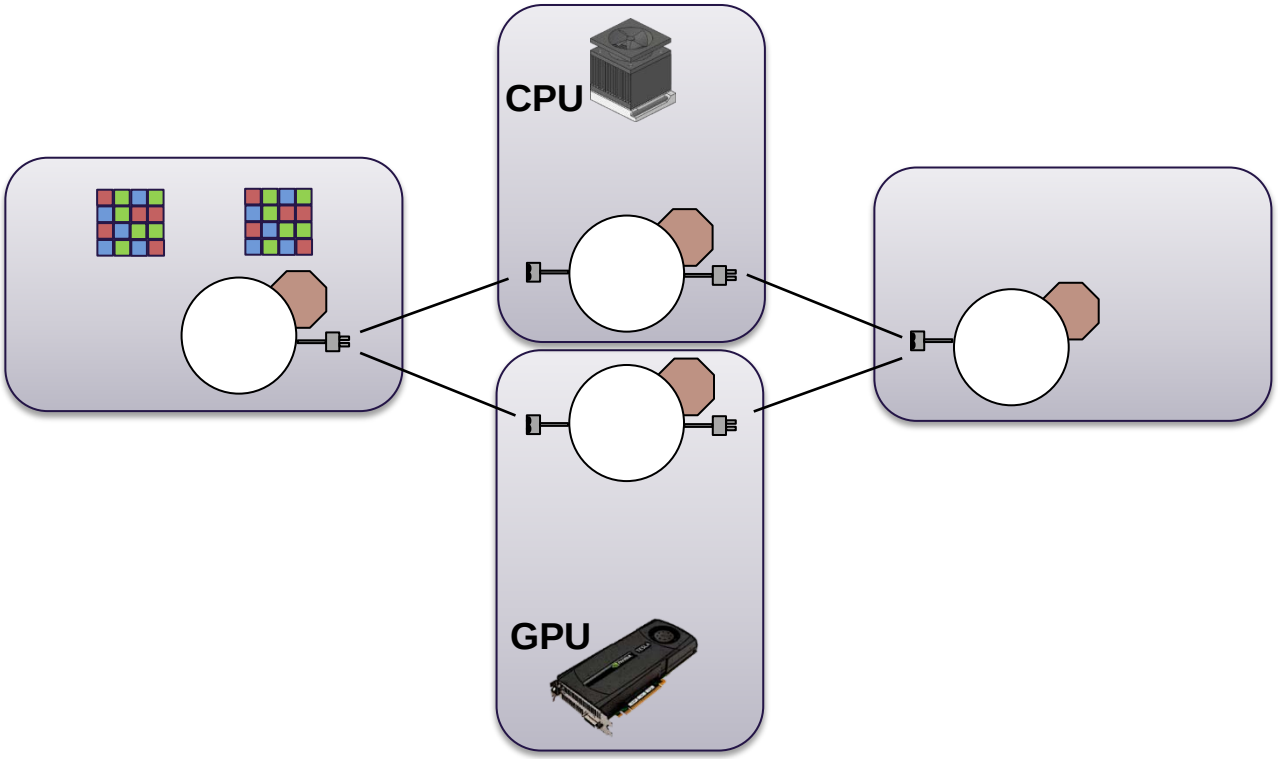
- **Devices can act as modules of a scientific workflow system**
  - Configurable generic input/output channels on devices
  - **One channel** is specific for **one data structure** (e.g. Hash, Image, File)
  - New data structures can be “registered” and are immediately usable
  - Developers just need to code the **compute** and (optionally) the **endOfStream** method
  - **IO** system is **decoupled** from processing system (process whilst transferring data)
  - Broker-based communication transparently establishes point-to-point connection
  - Any workflow device has full **access to** the live **control-system**





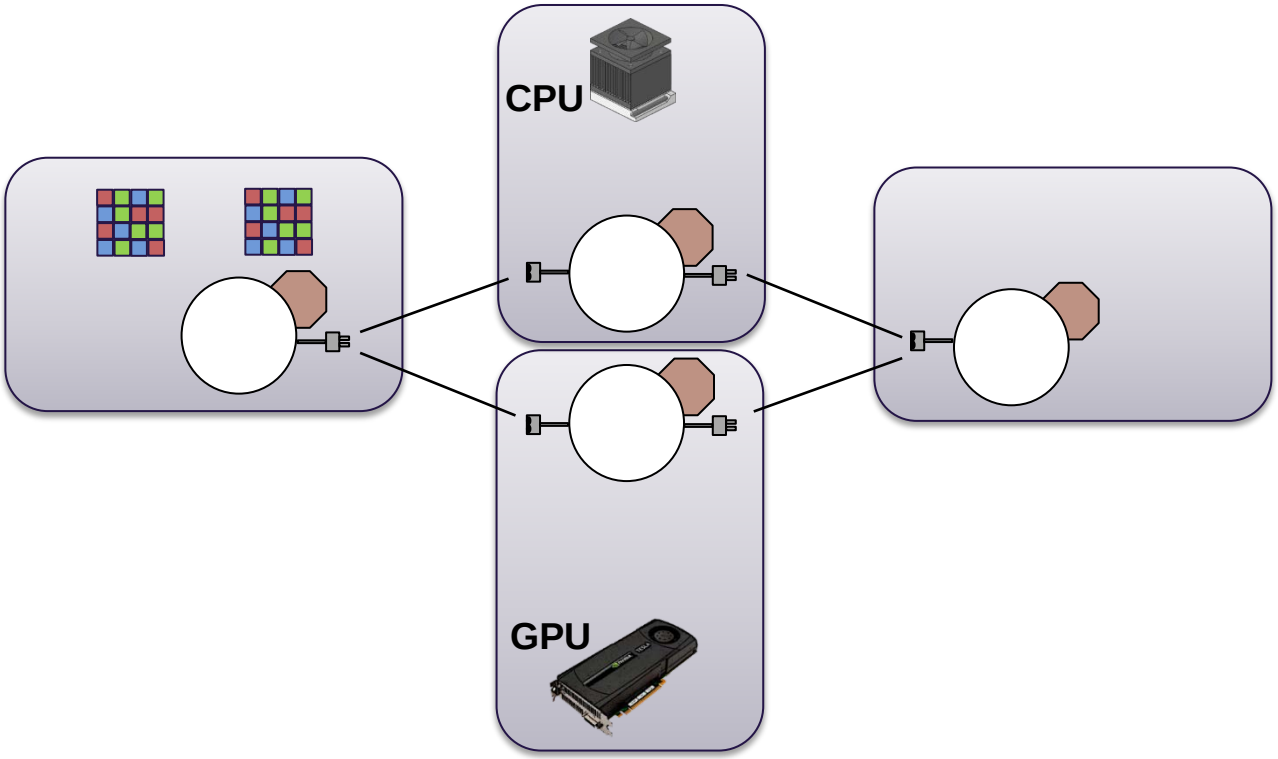
Distributed processing

Workflows support load balancing



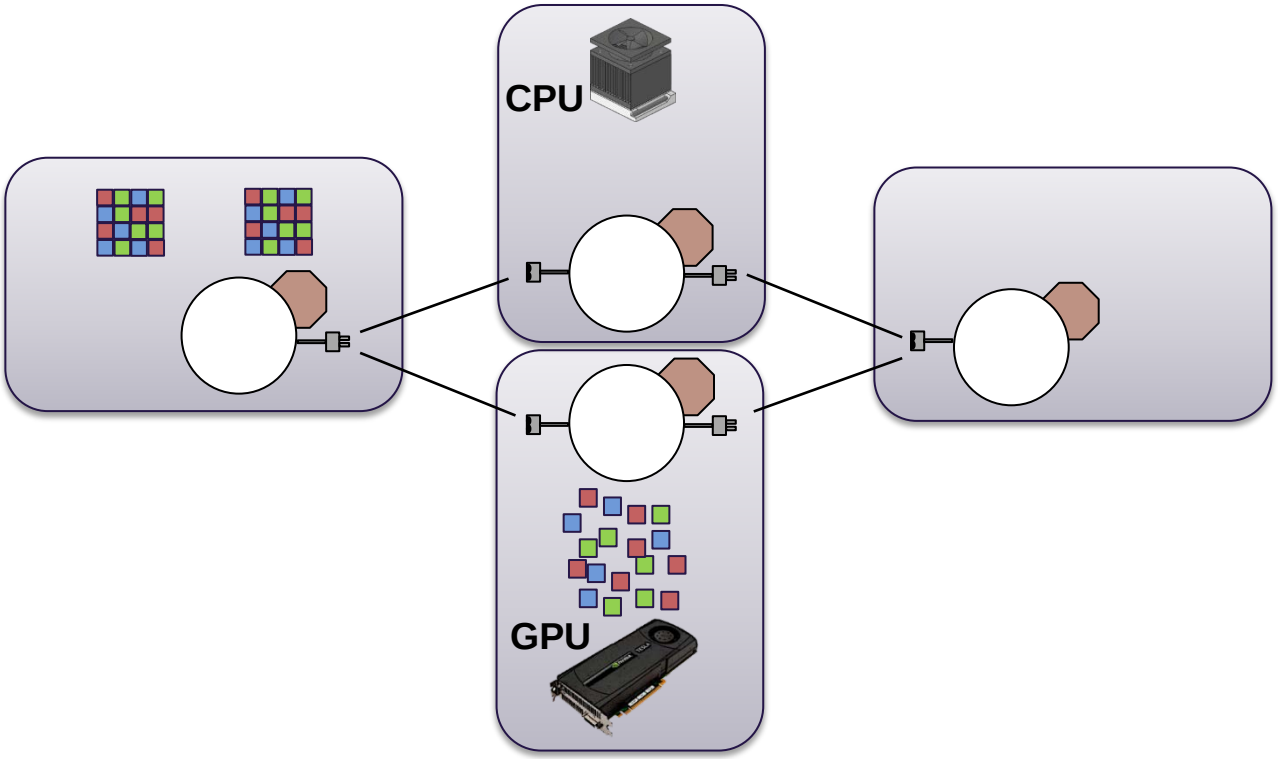
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



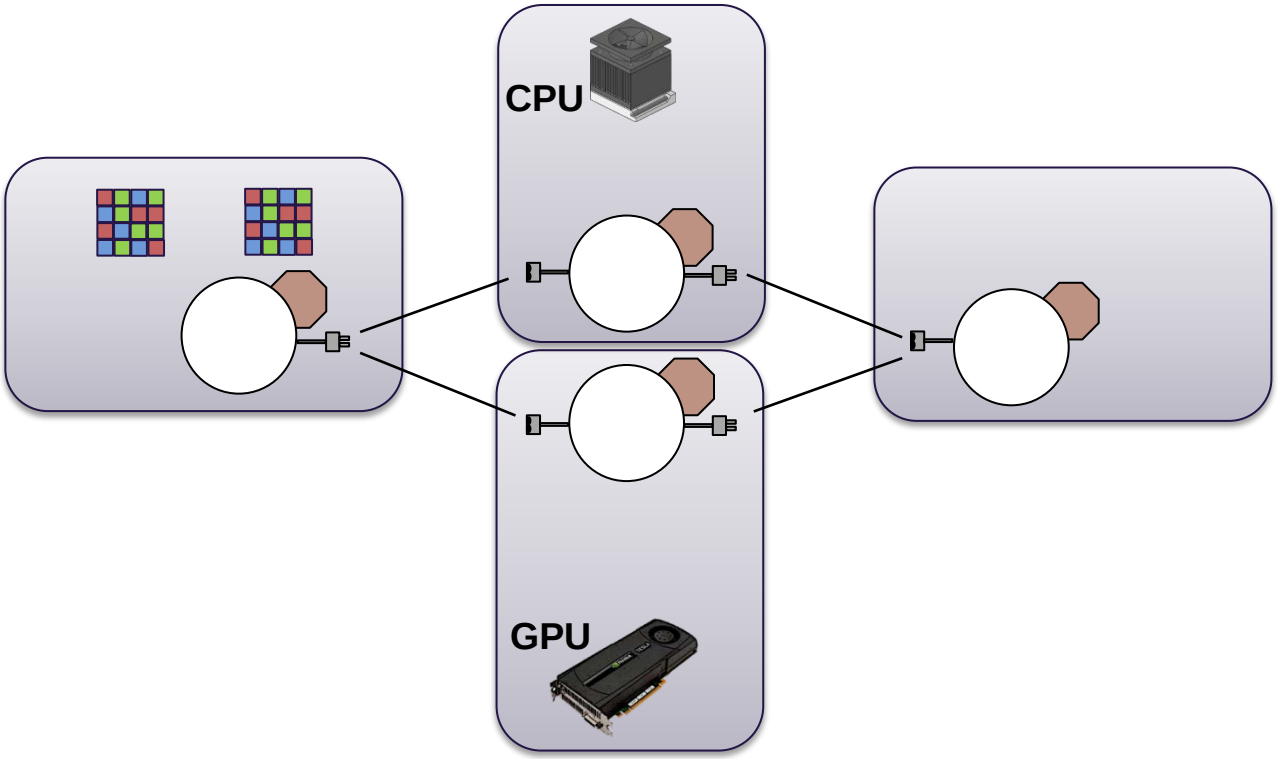
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



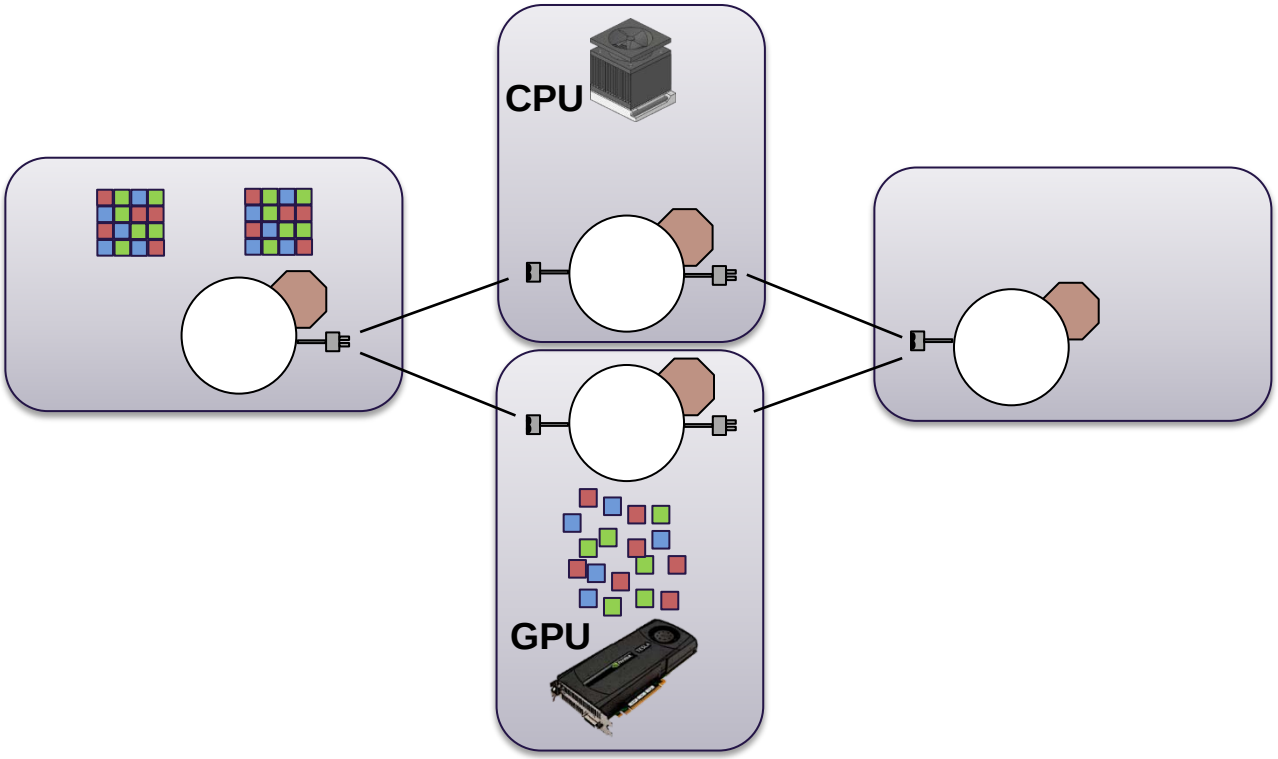
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



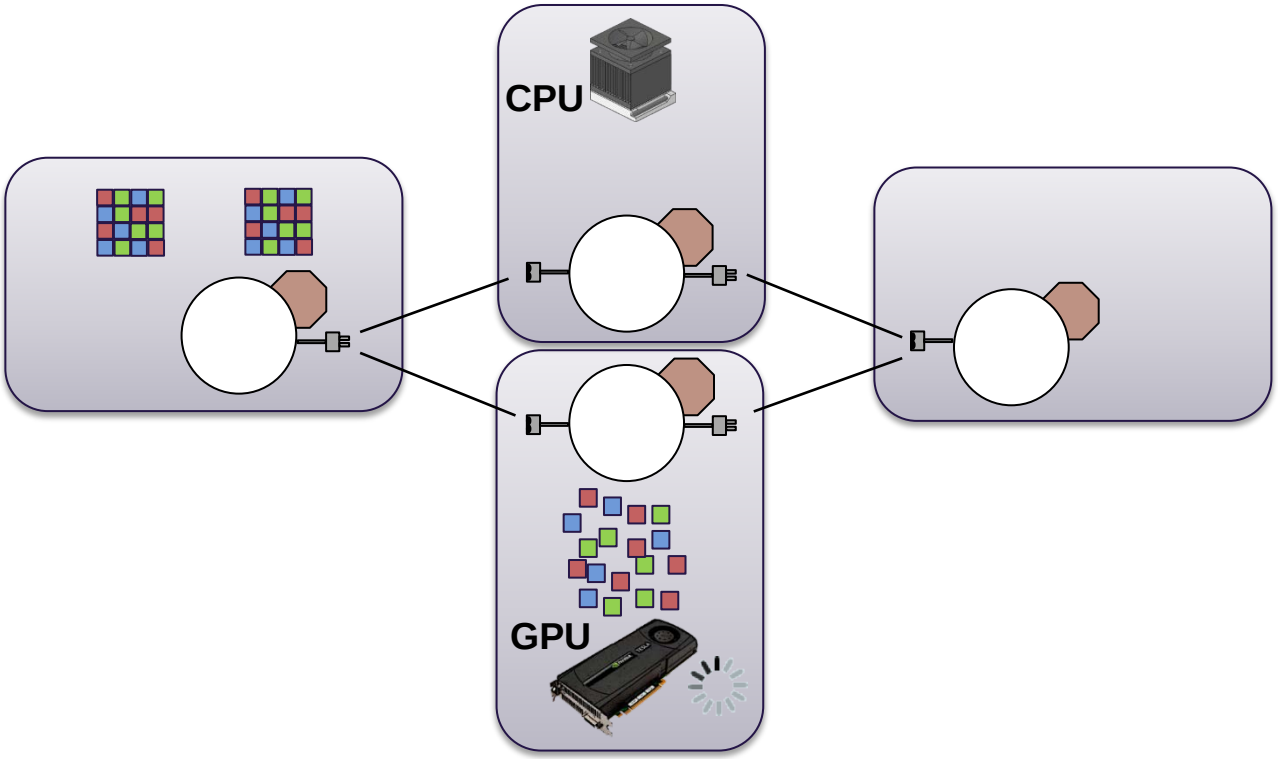
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



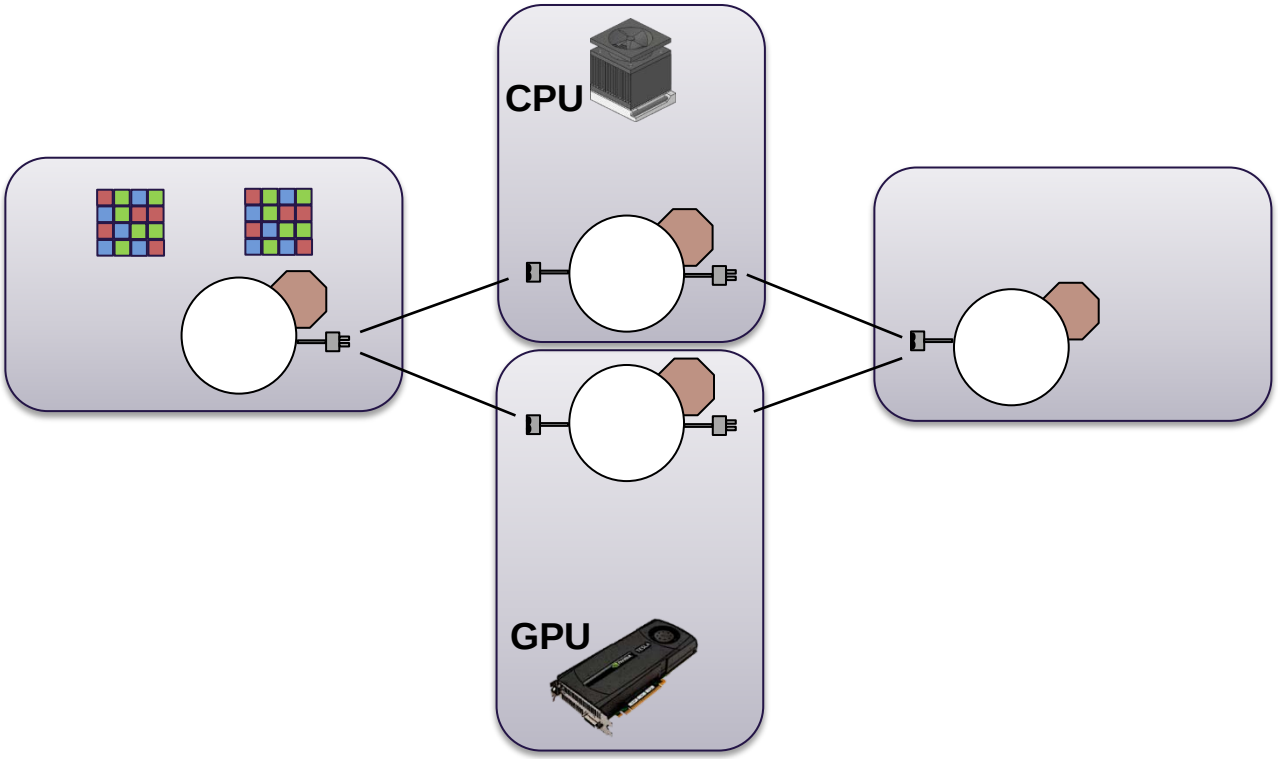
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



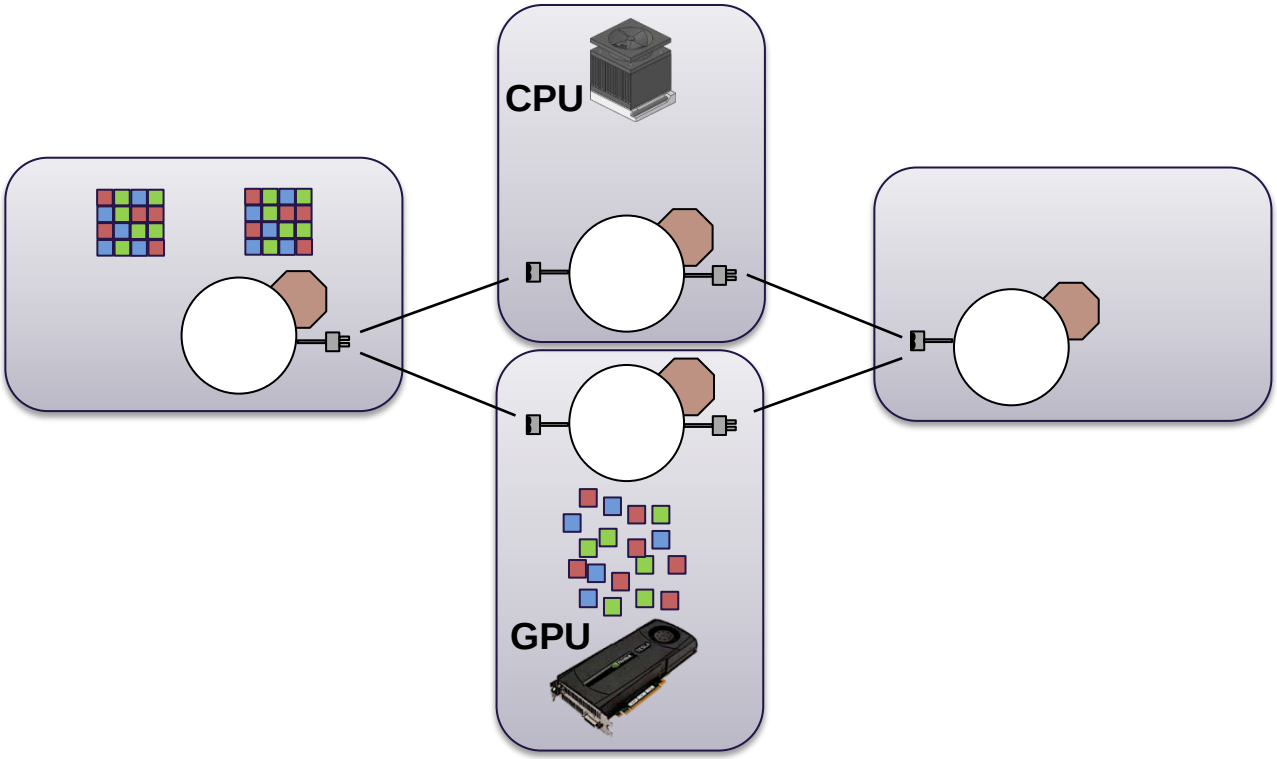
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



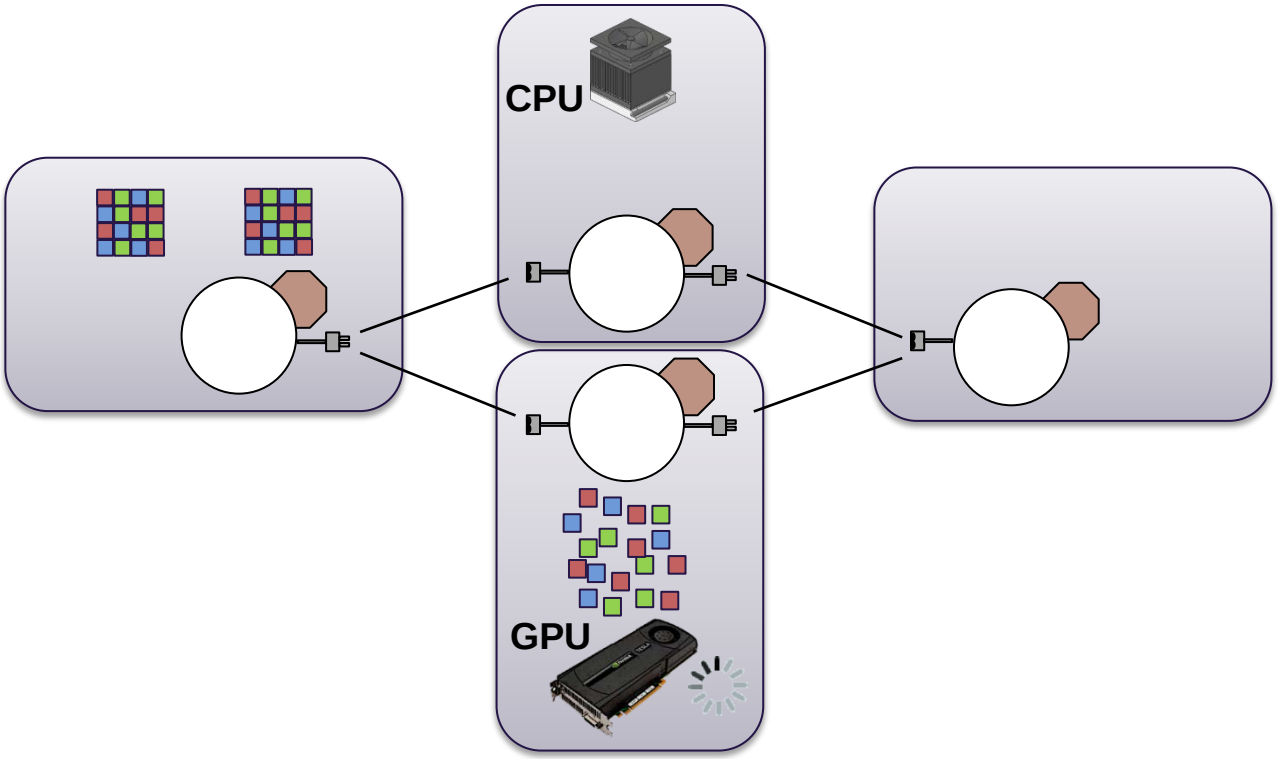
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



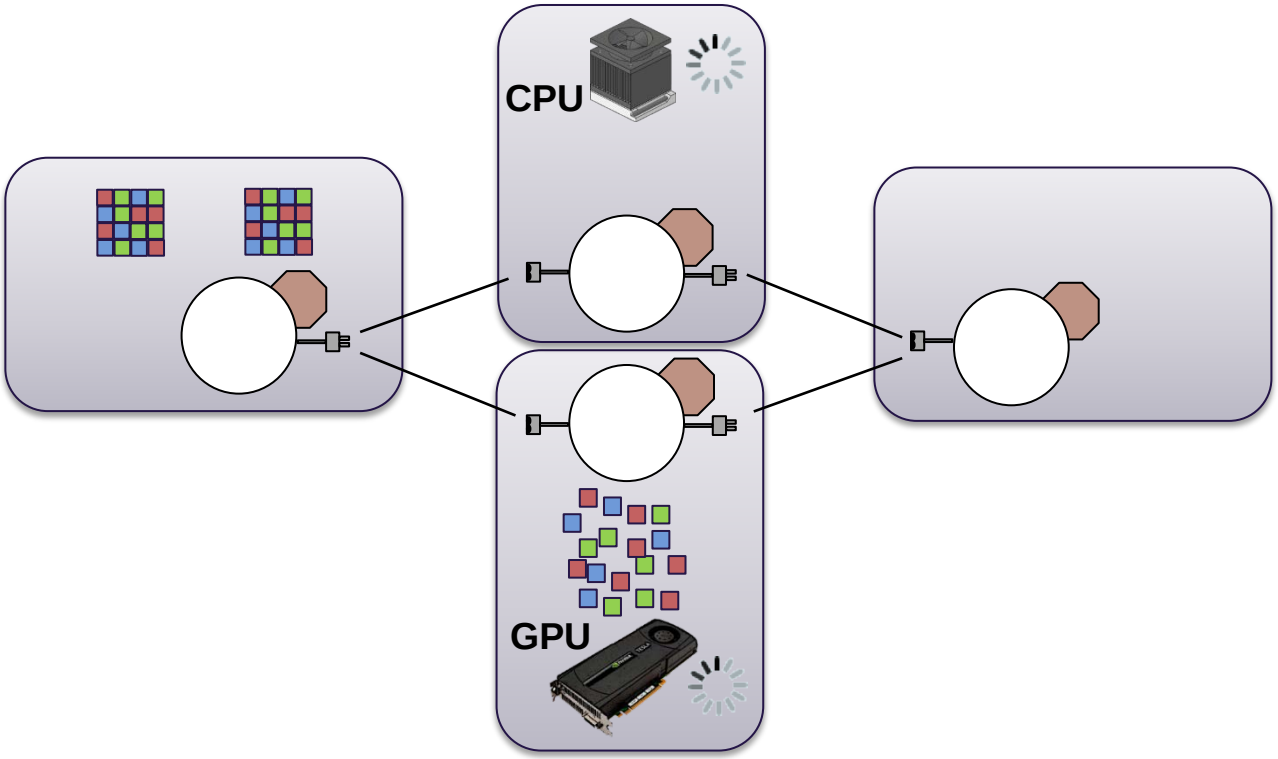
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



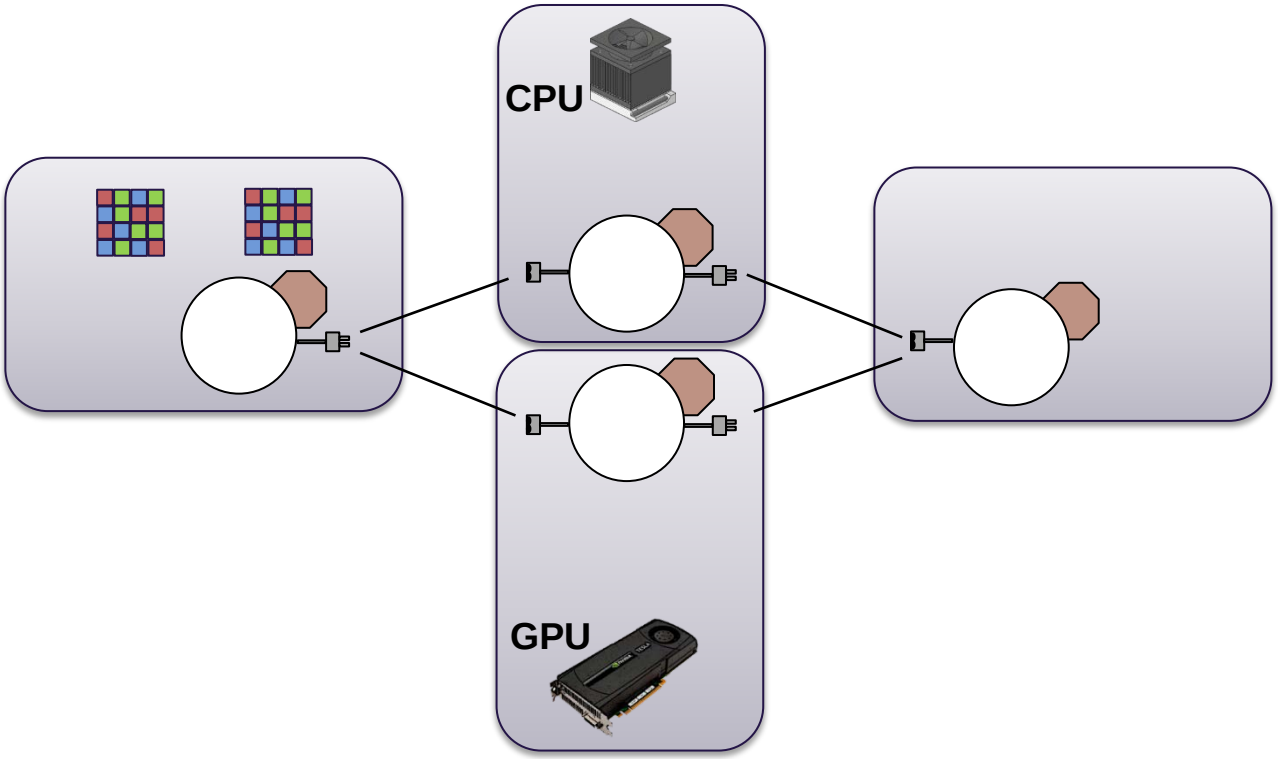
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



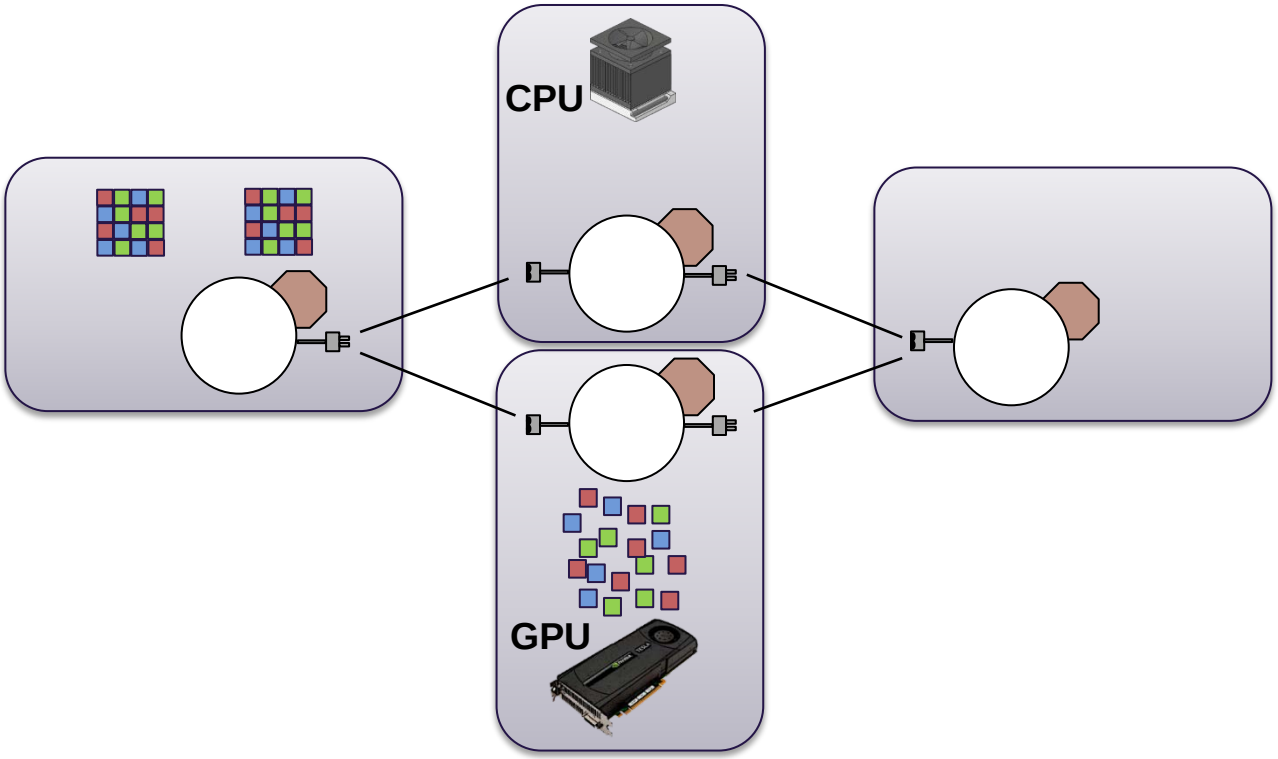
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



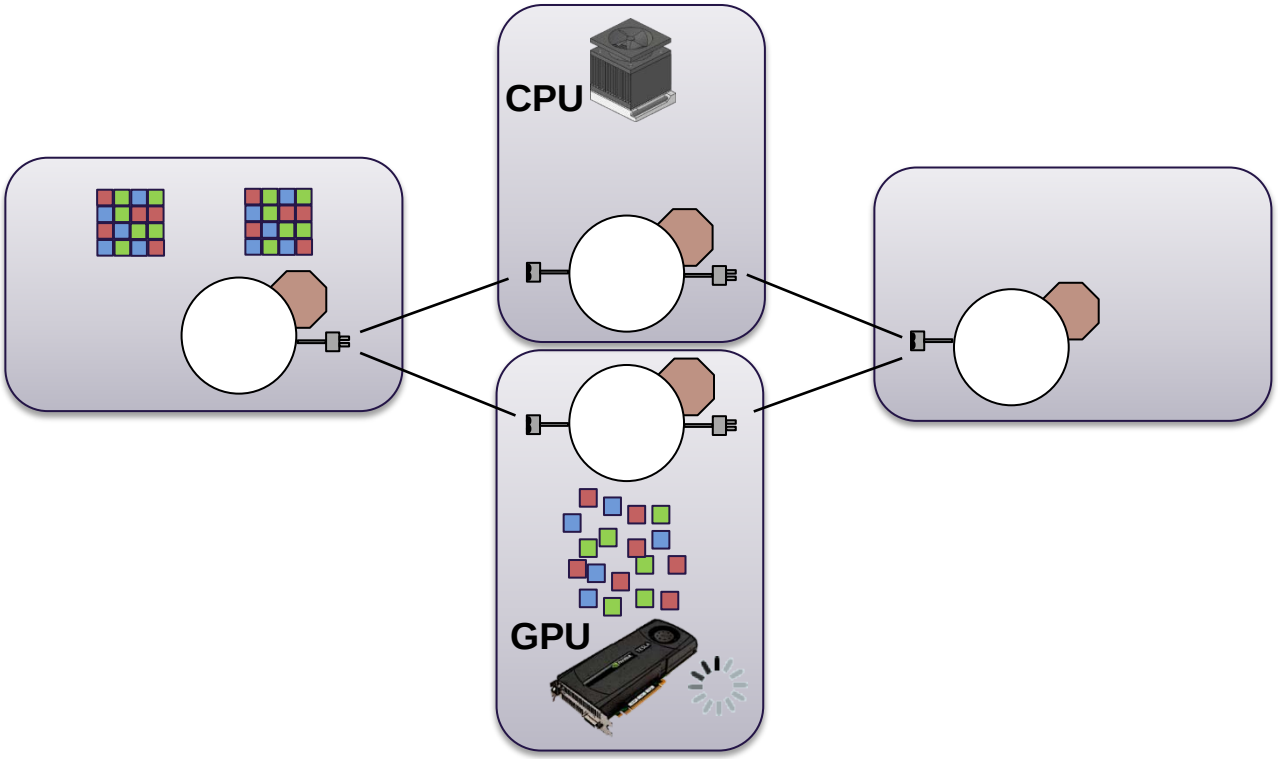
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



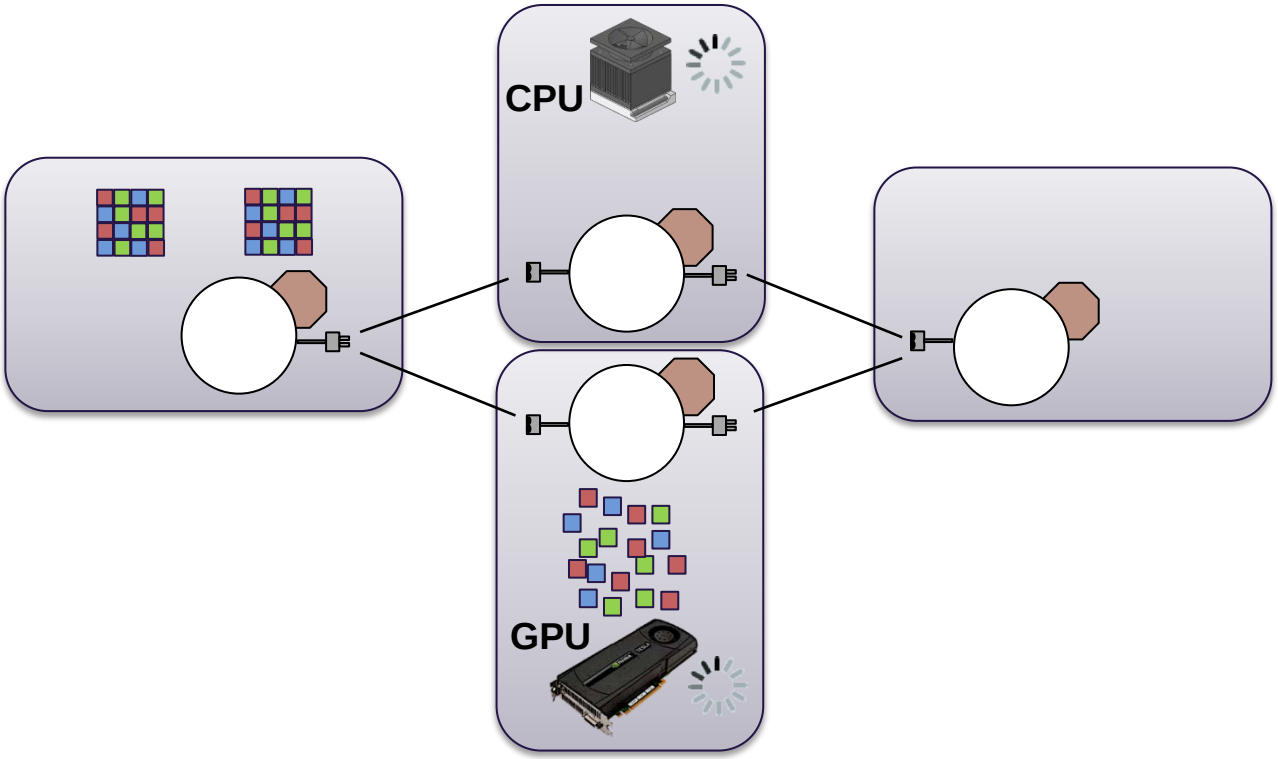
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



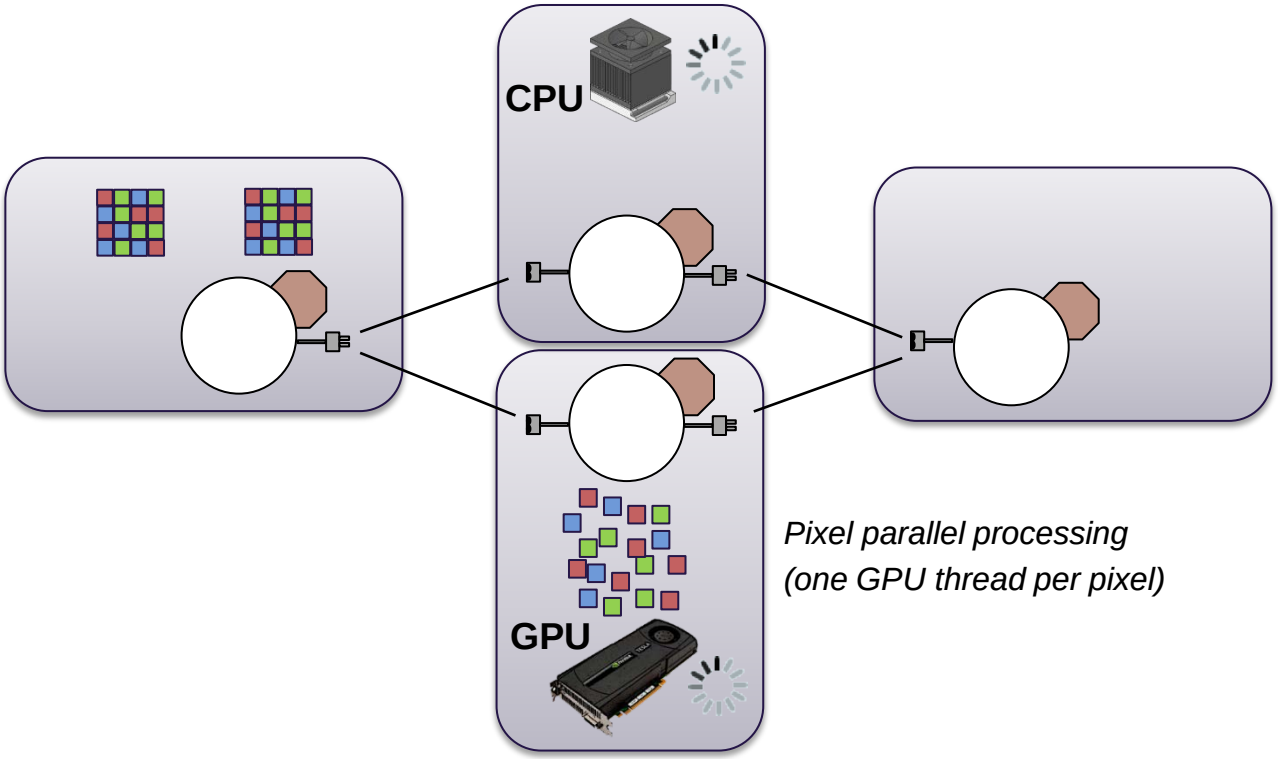
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



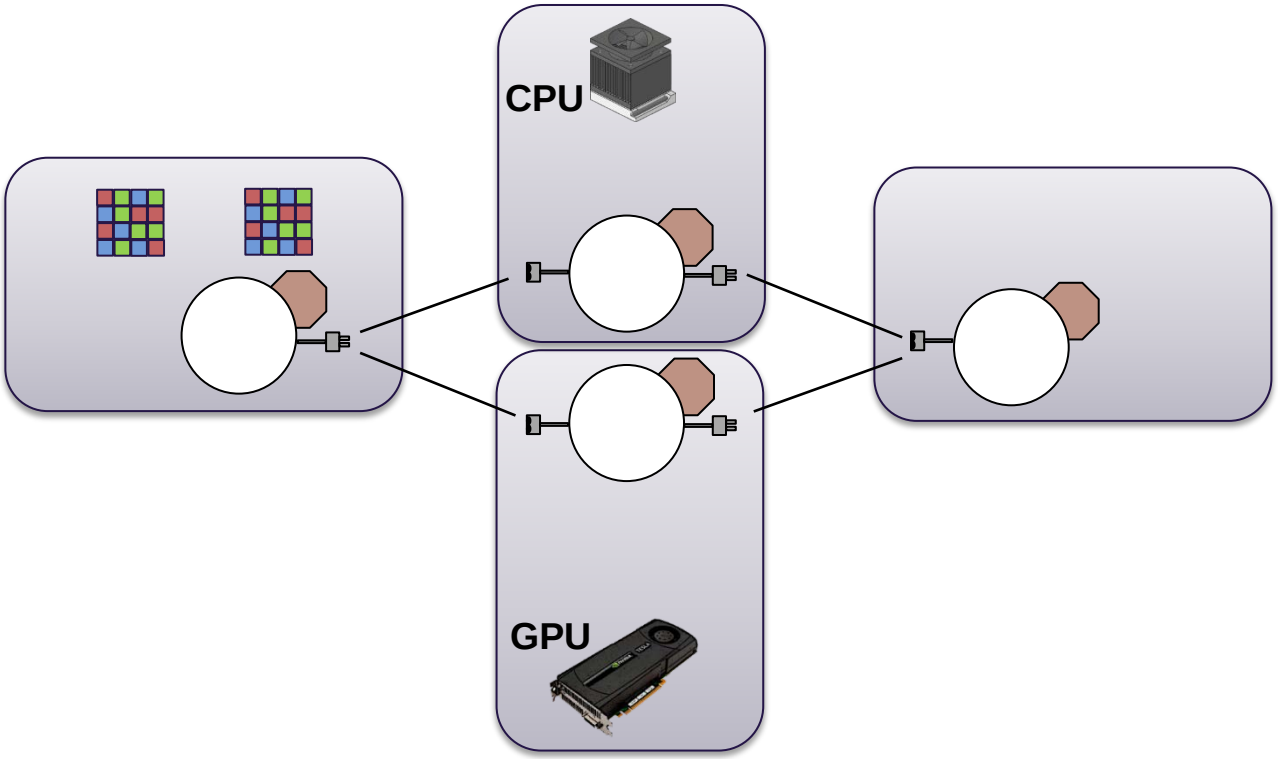
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



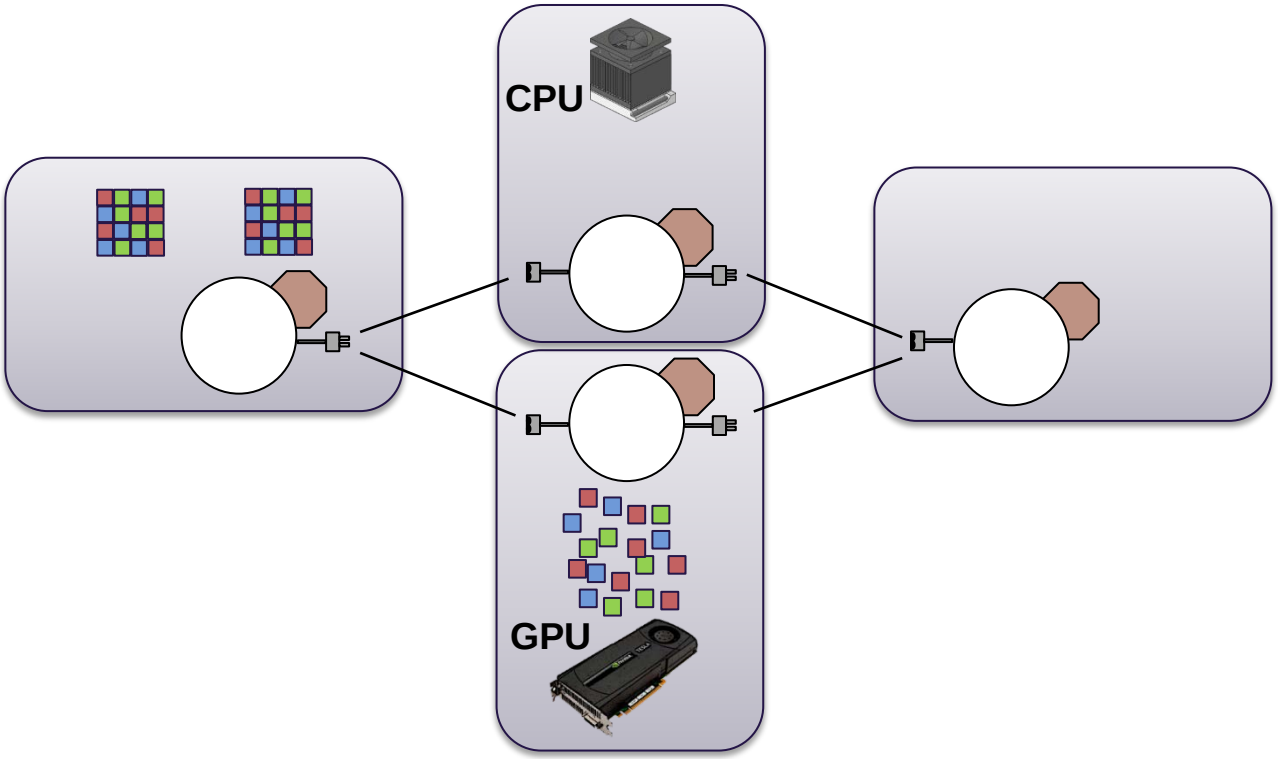
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



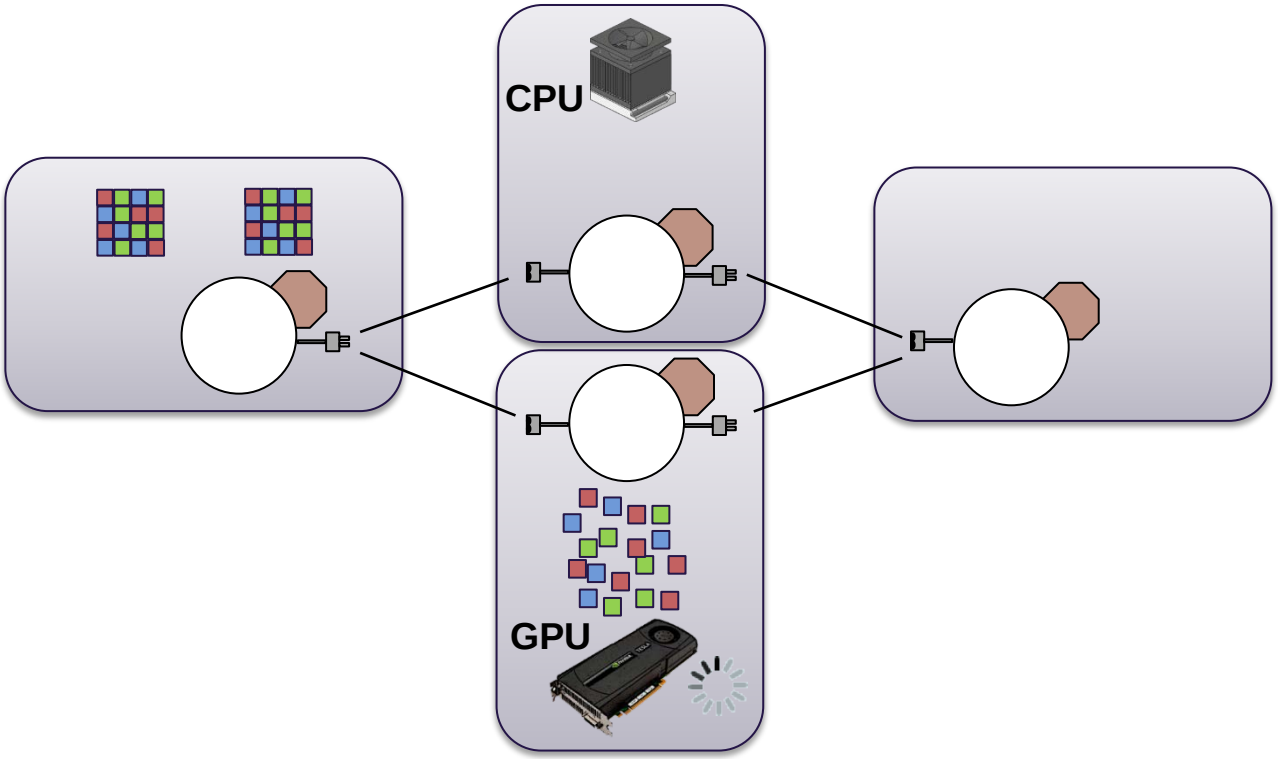
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



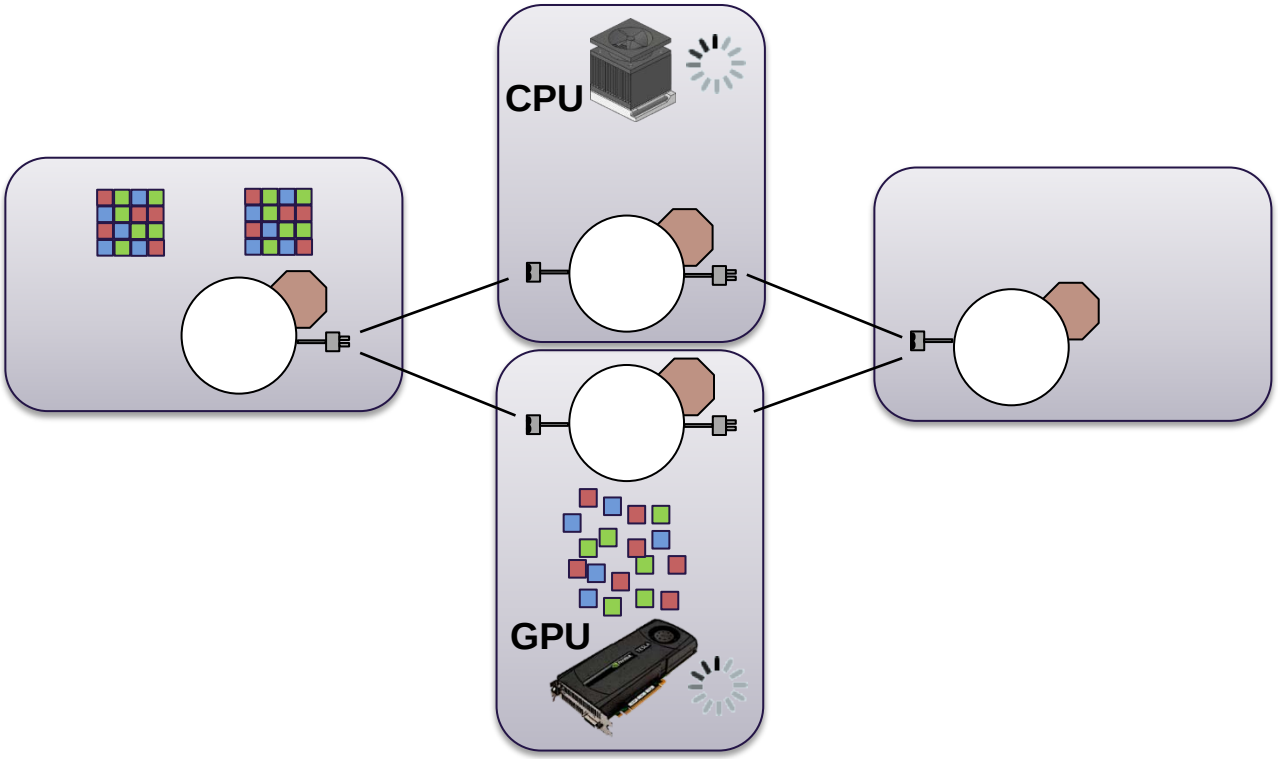
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



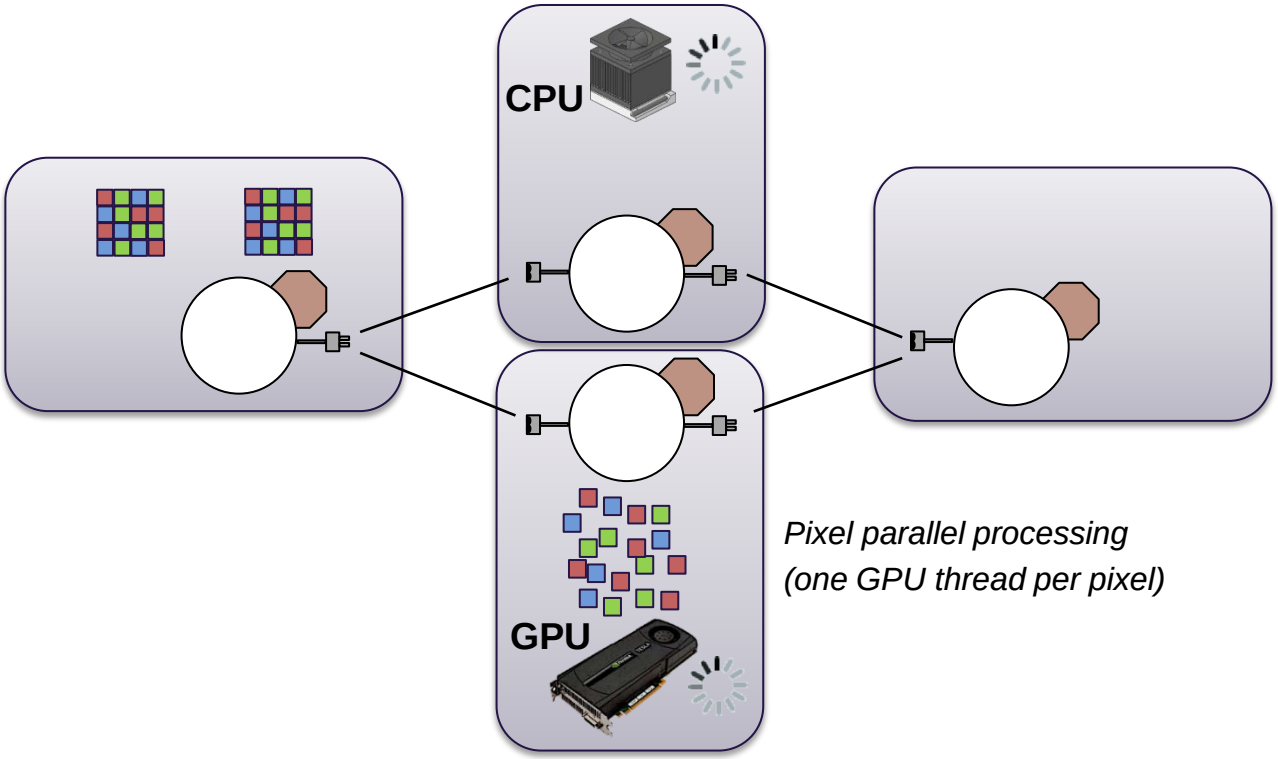
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



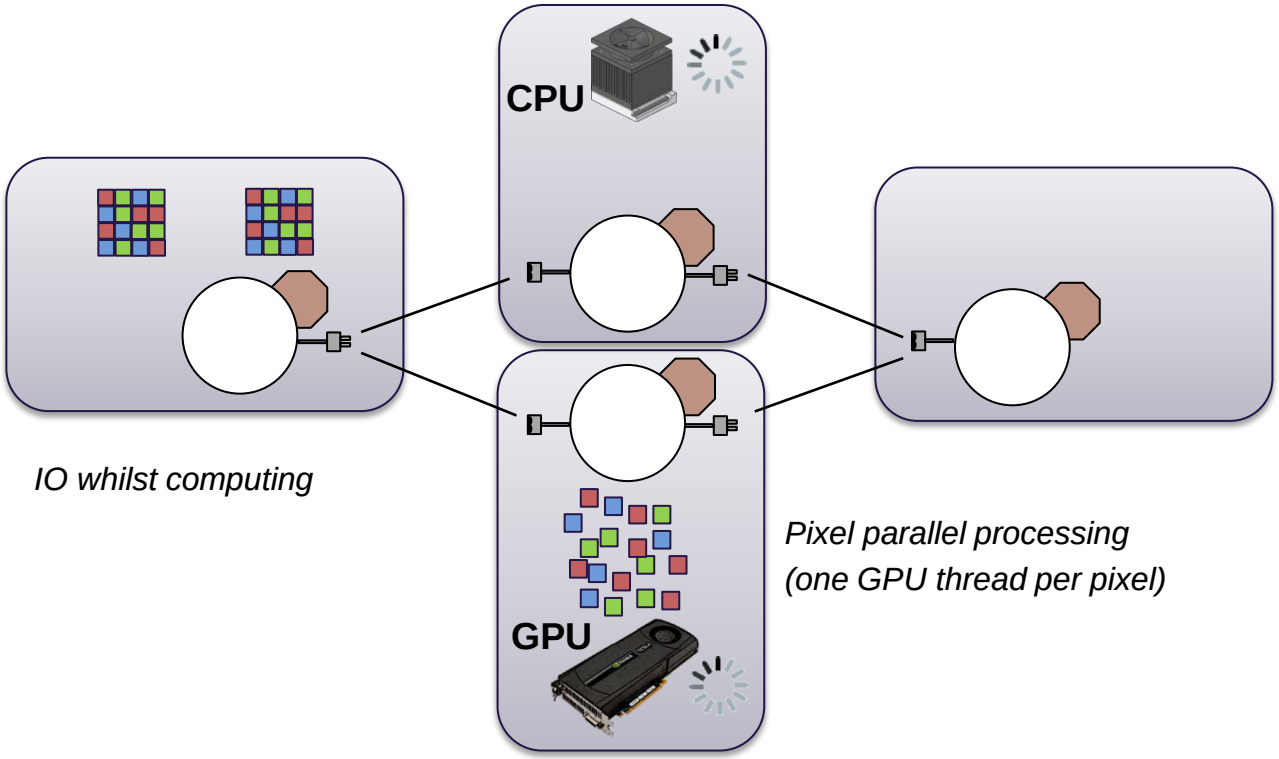
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



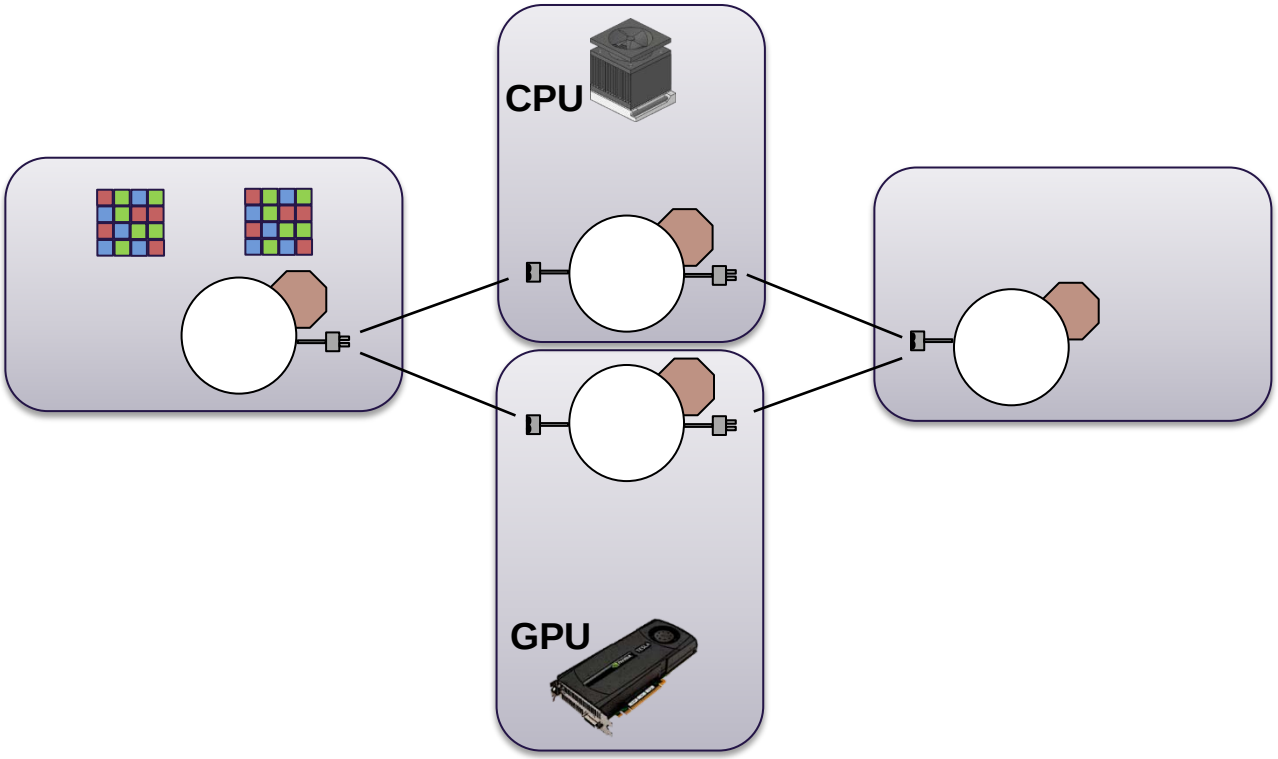
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



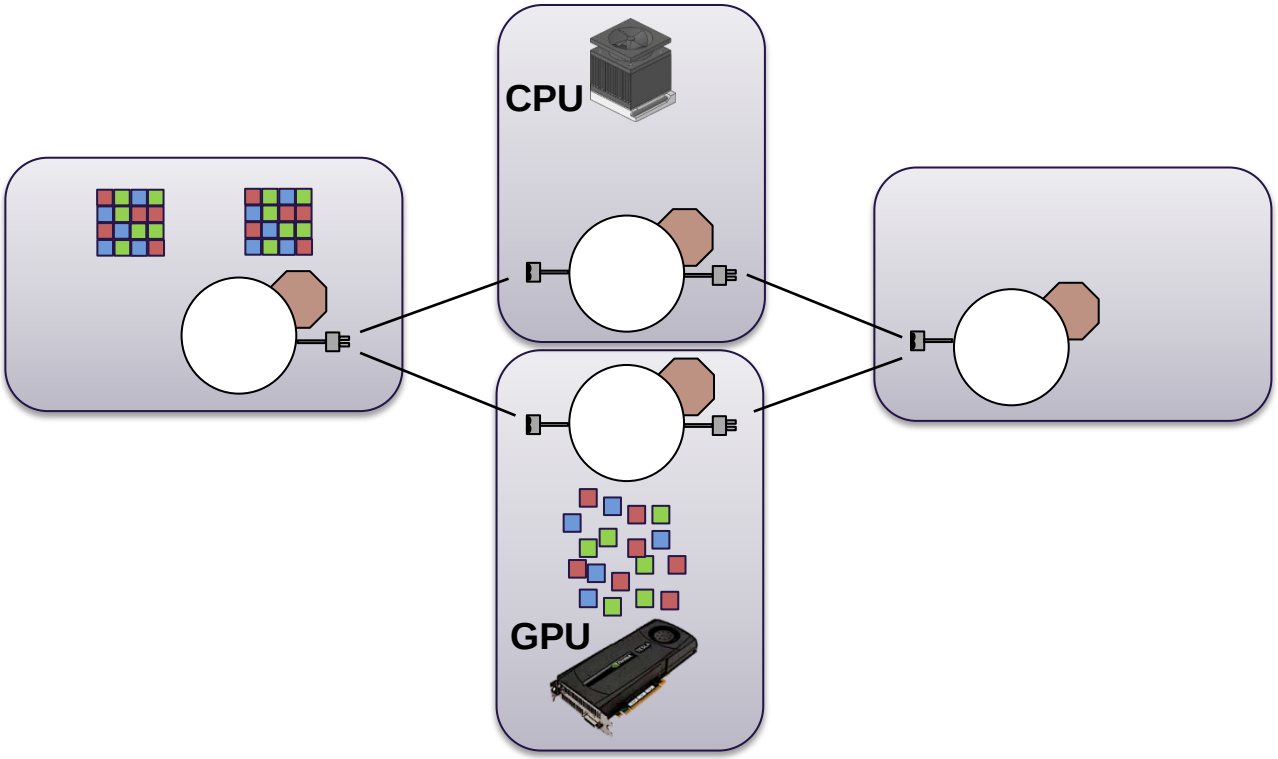
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



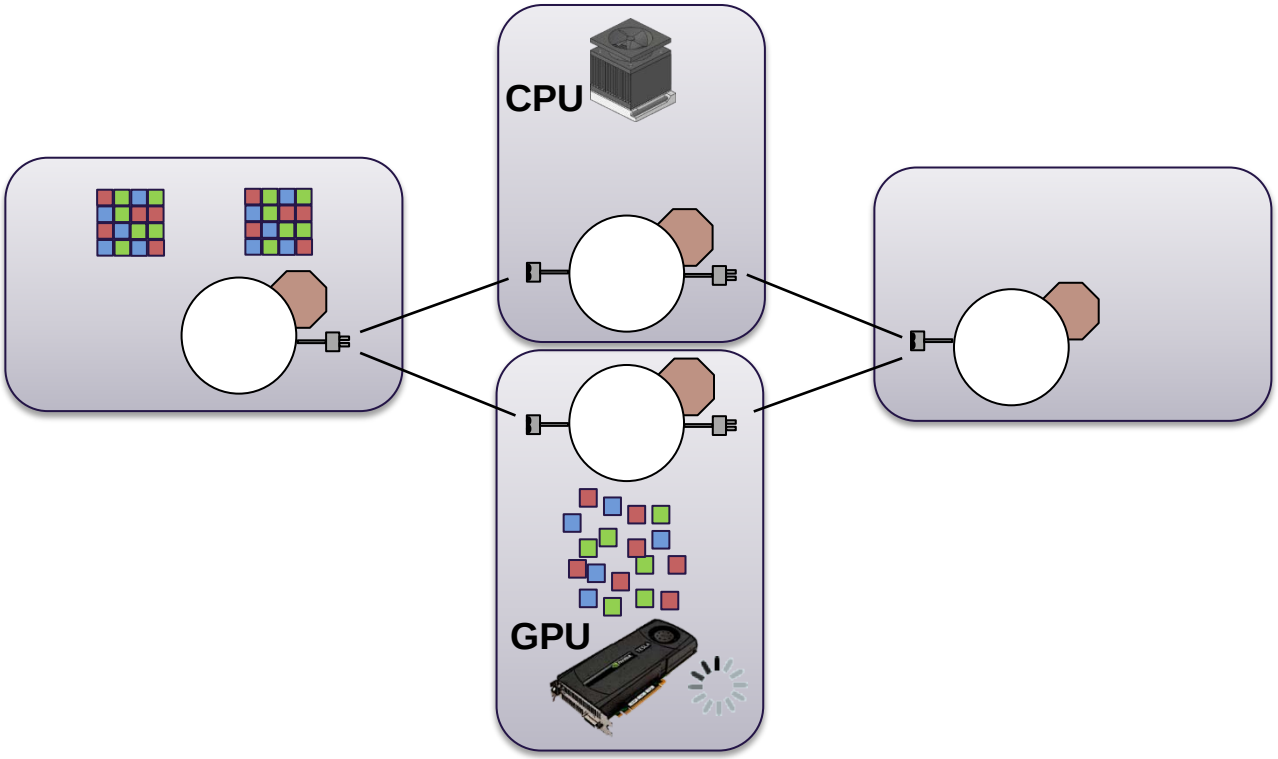
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



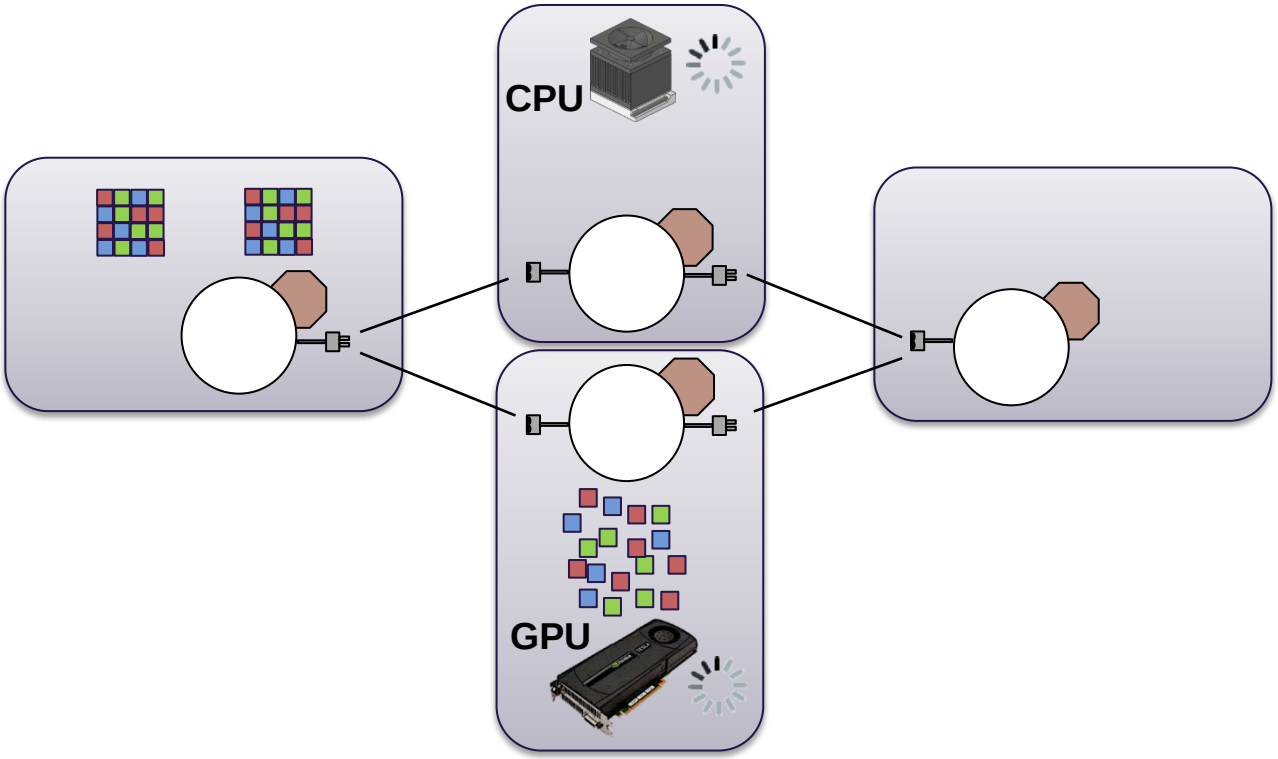
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



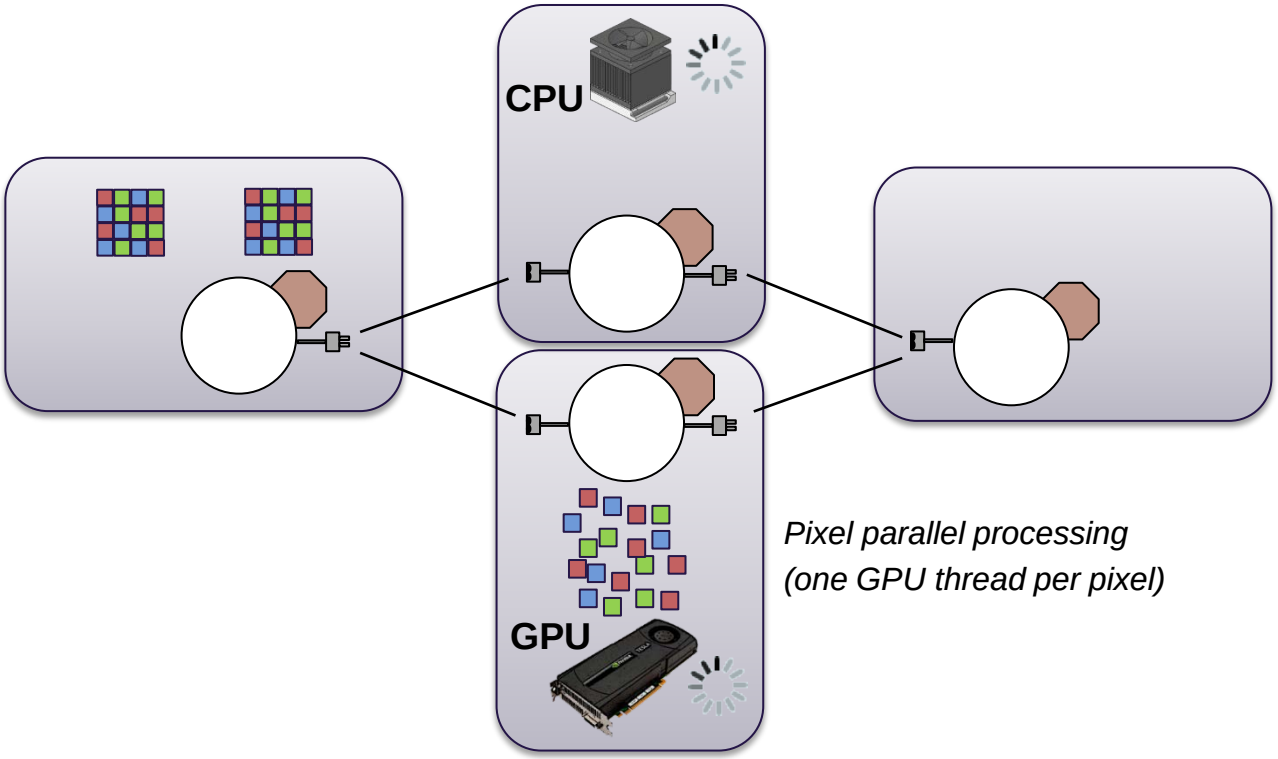
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



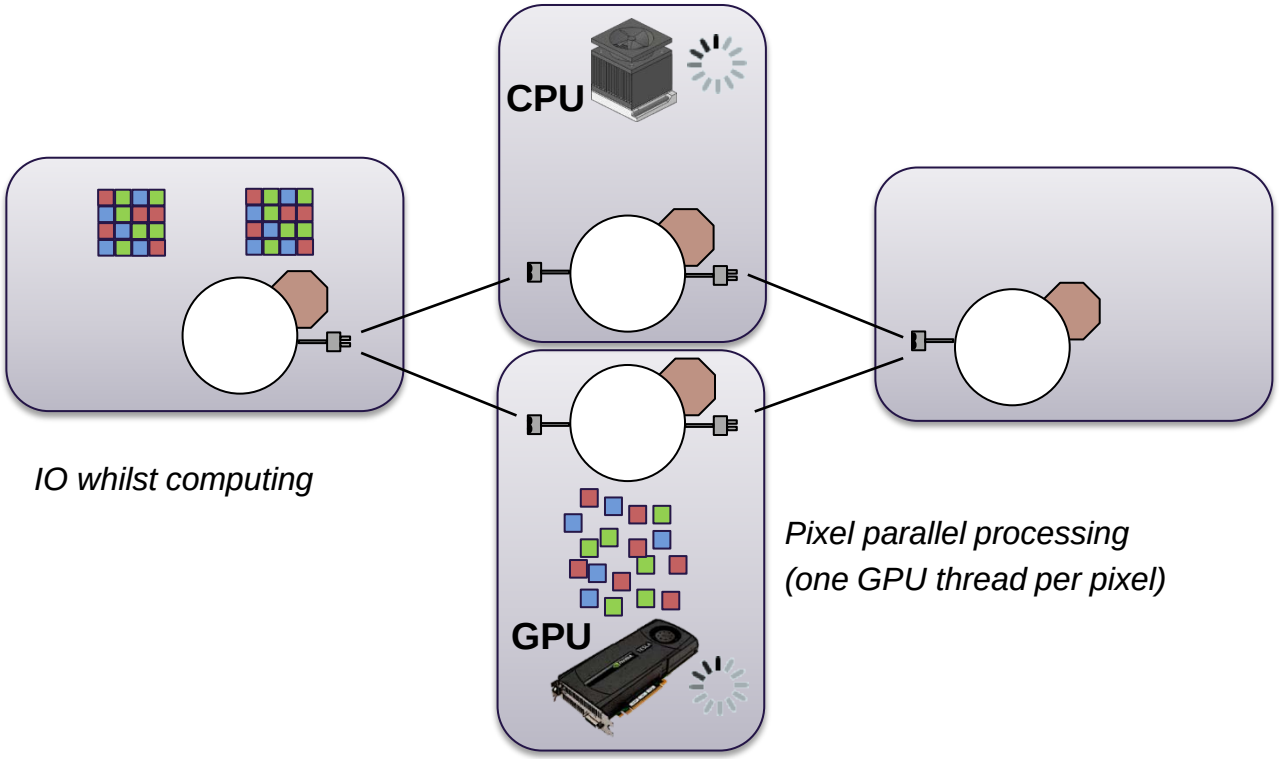
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



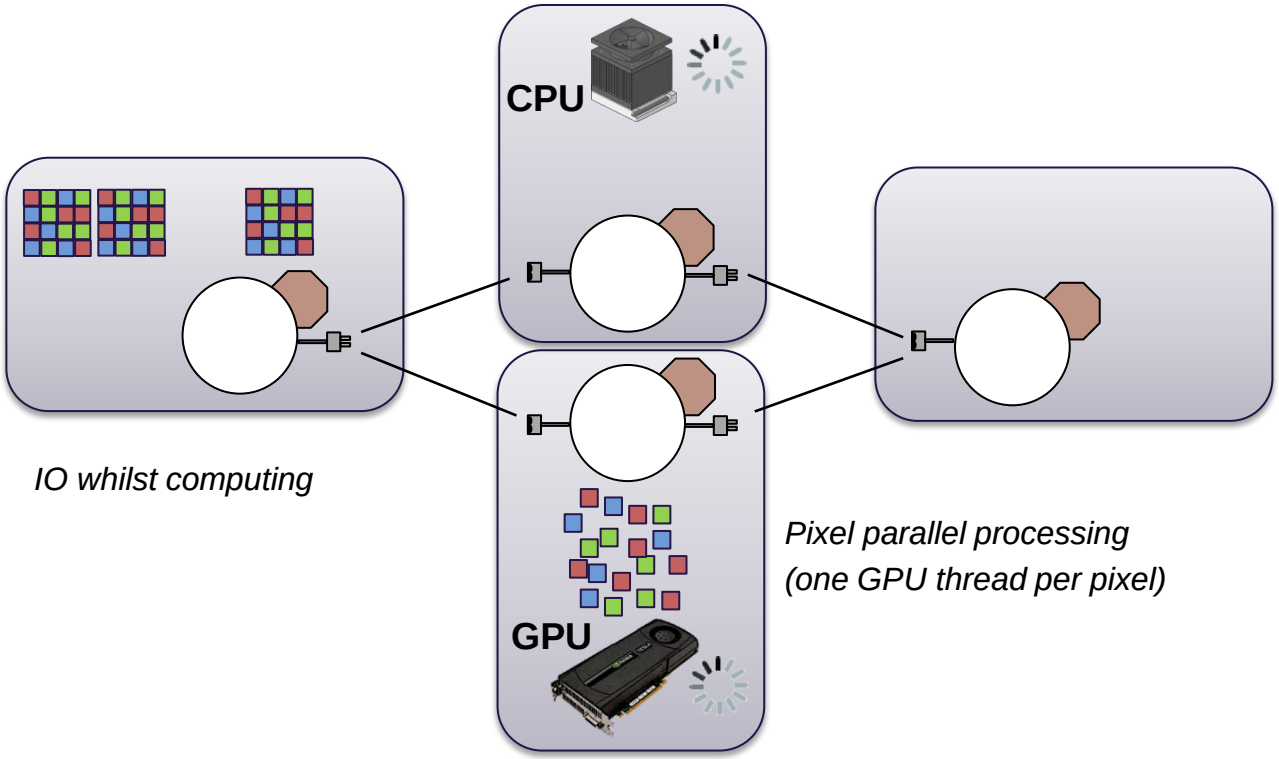
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



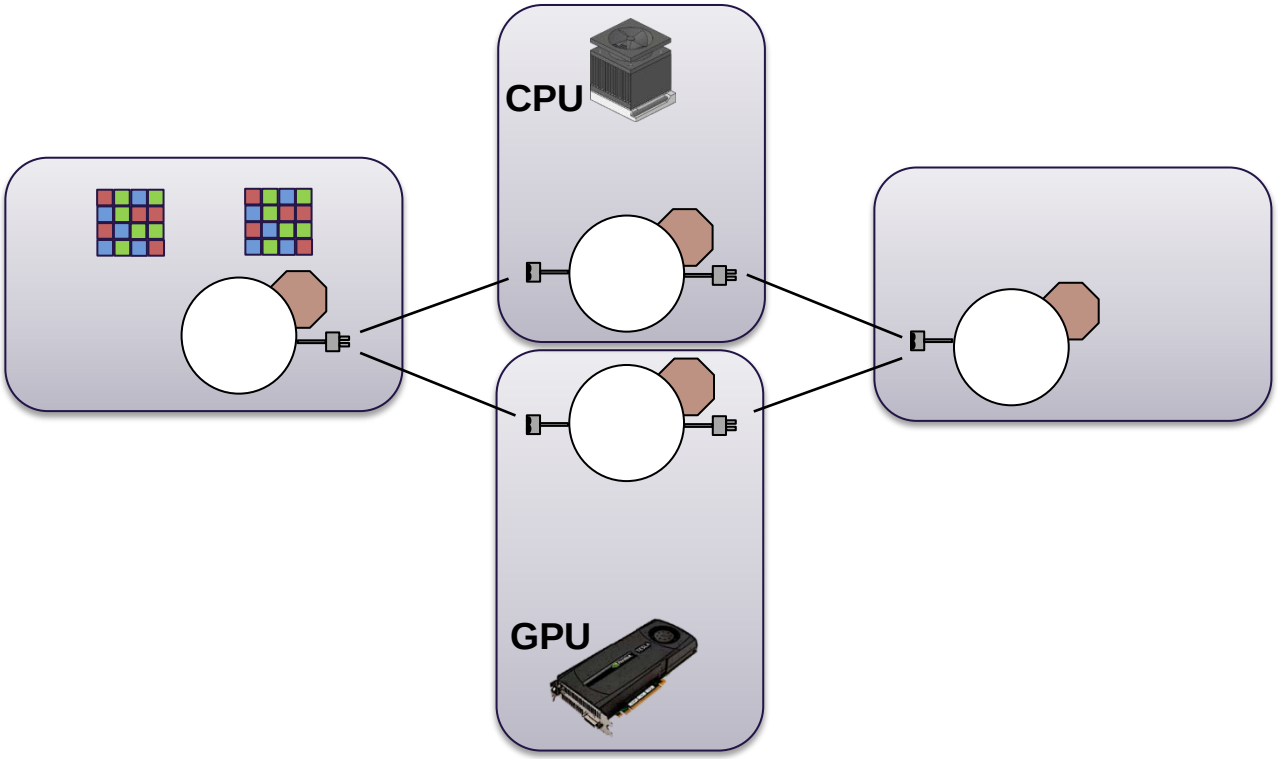
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



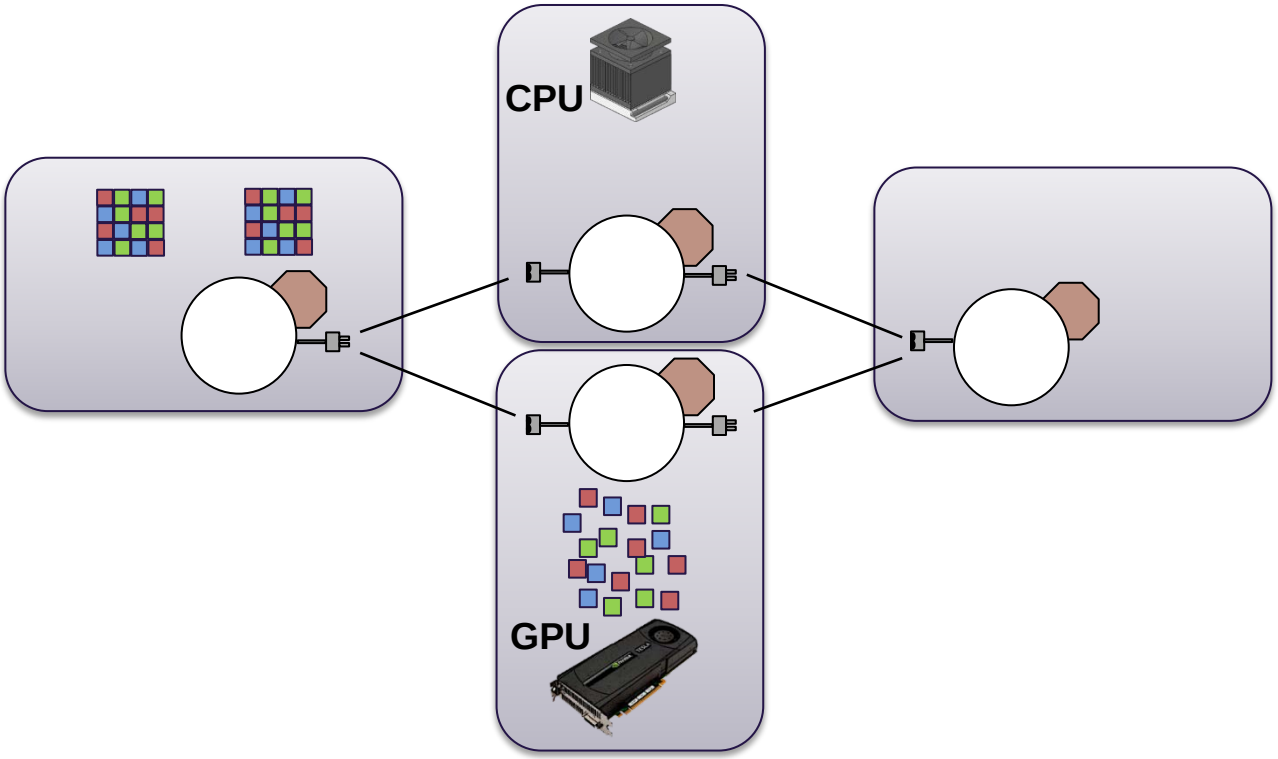
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

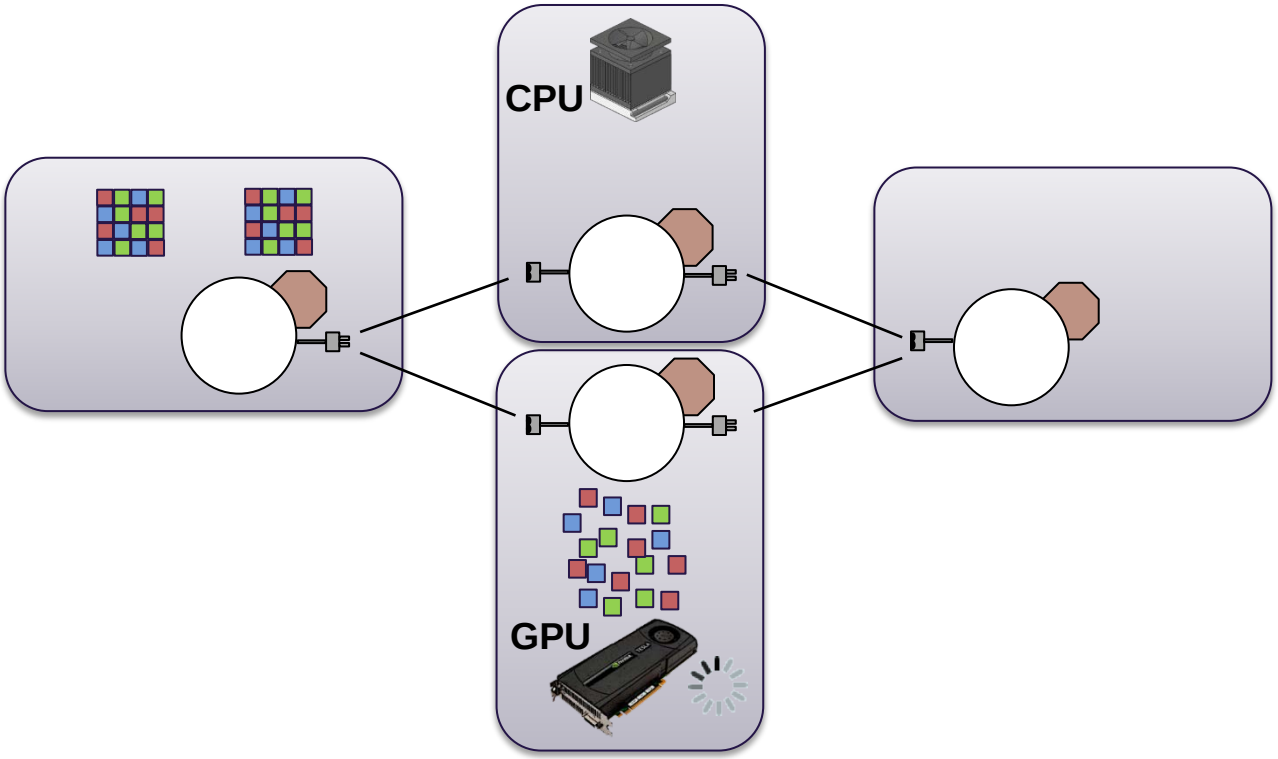
Workflows support load balancing



- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

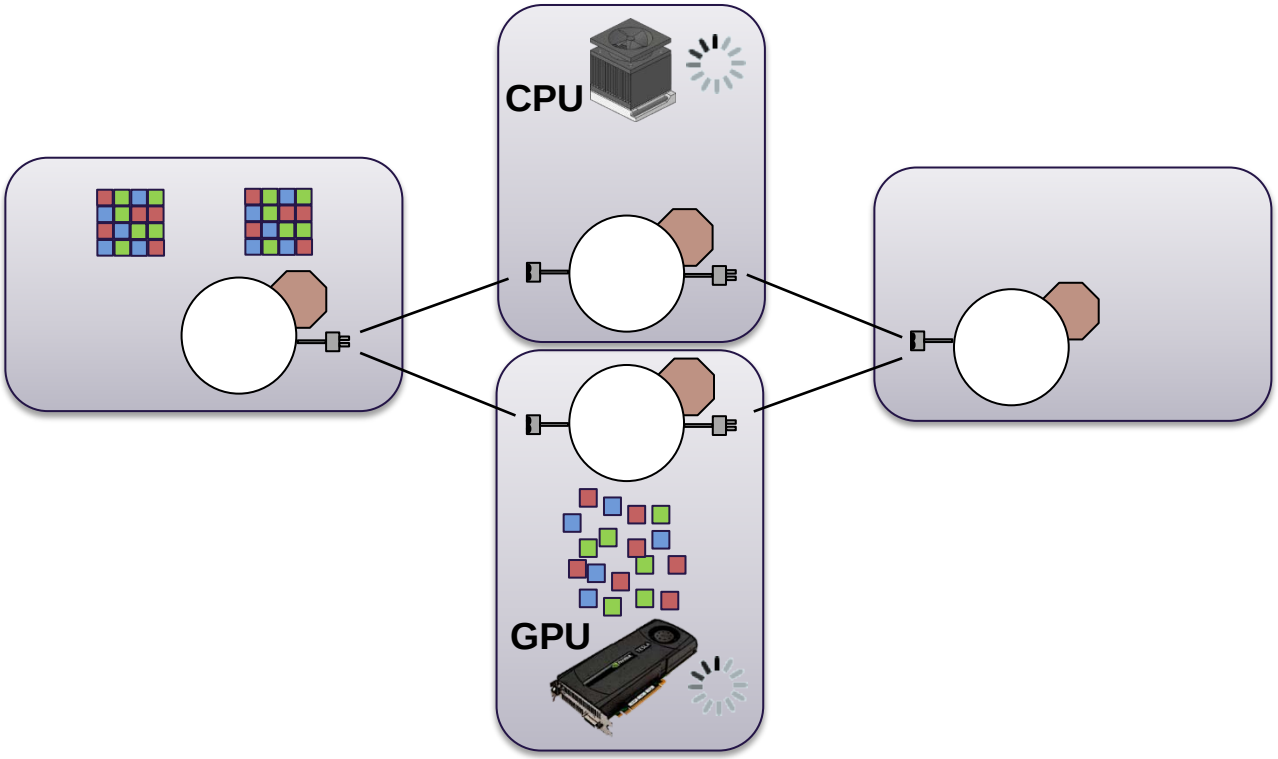
European XFEL

Workflows support load balancing



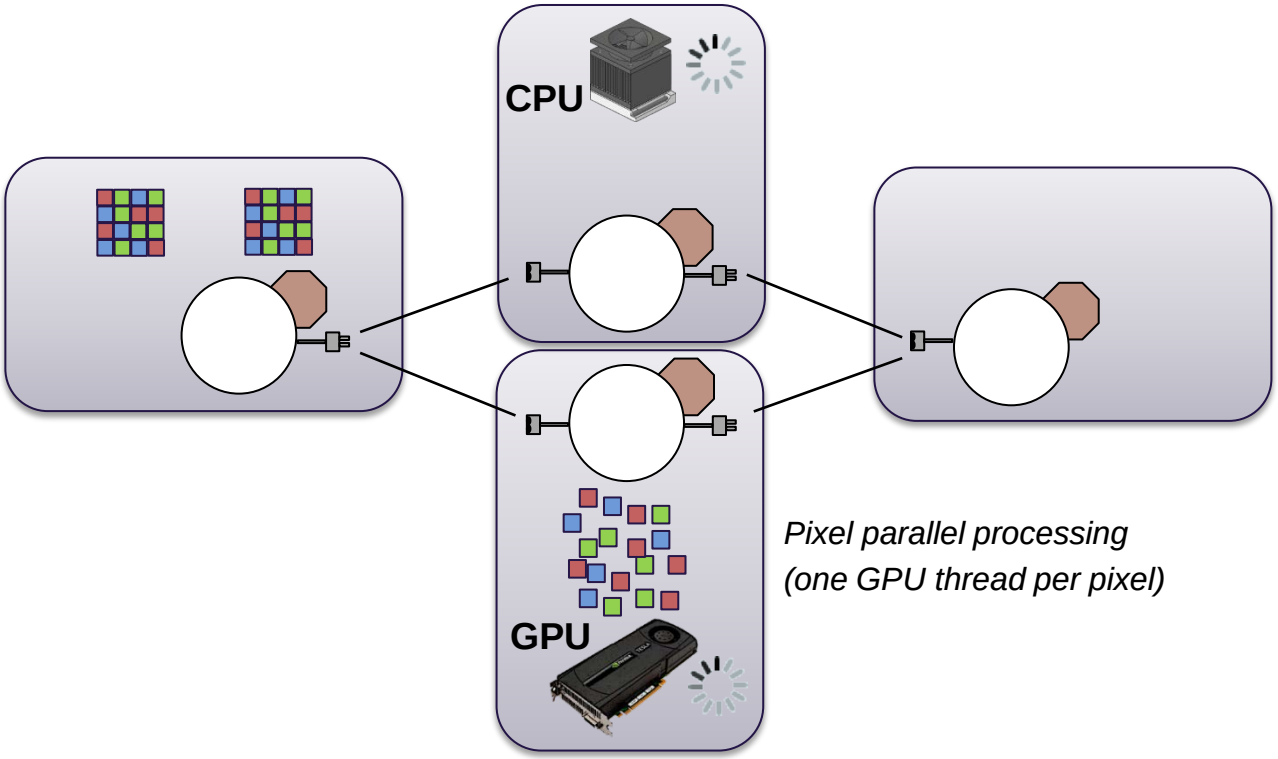
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



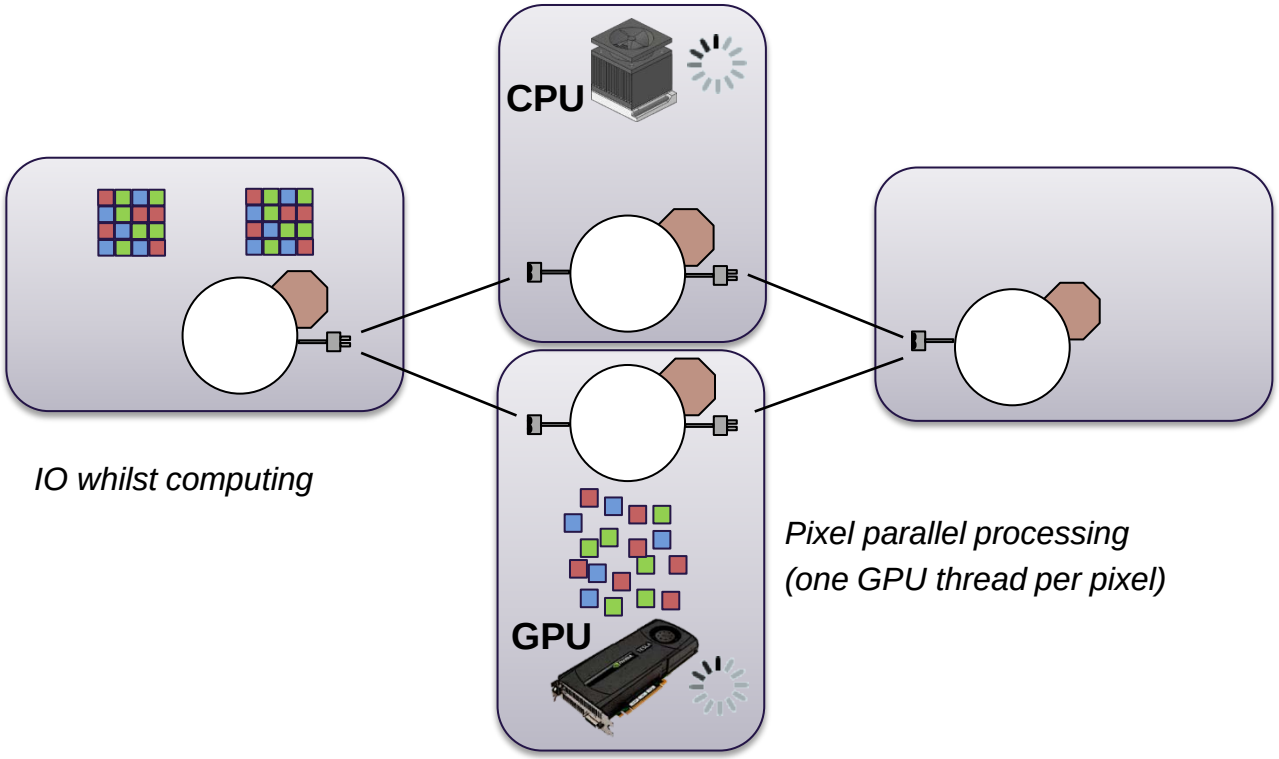
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



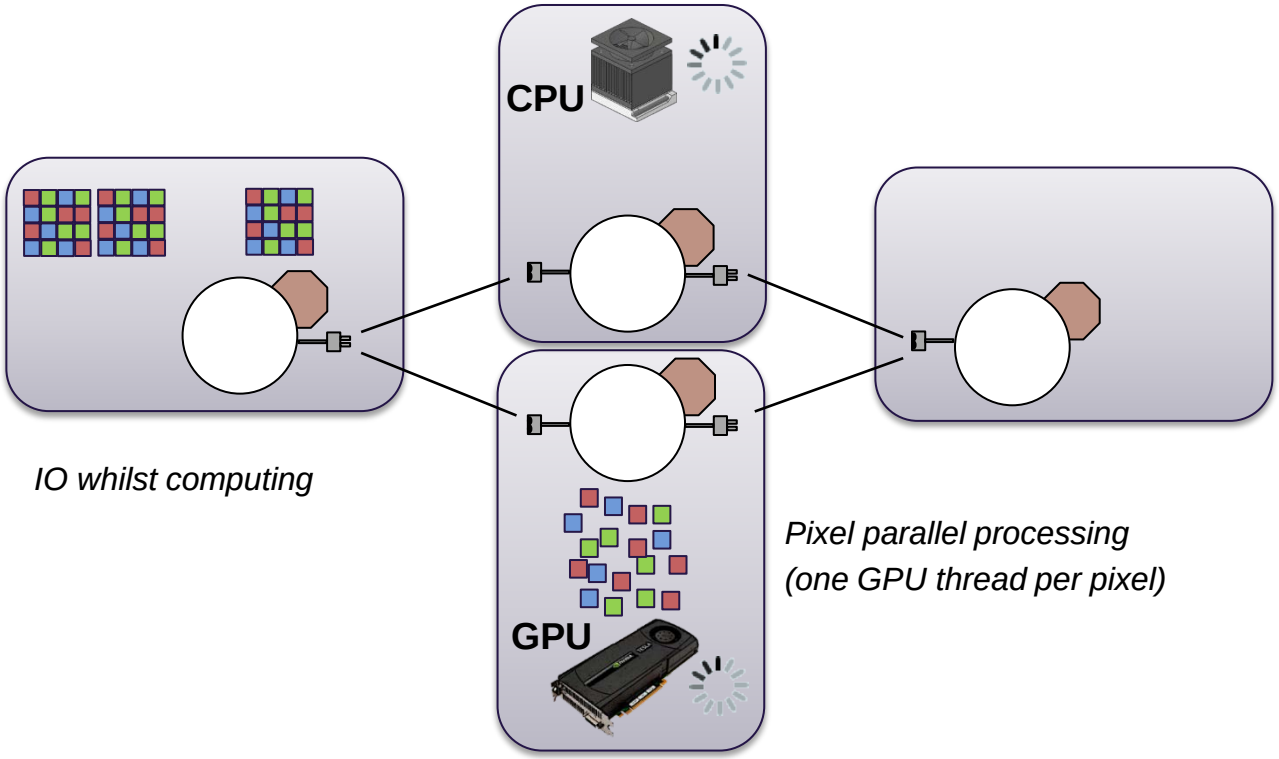
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



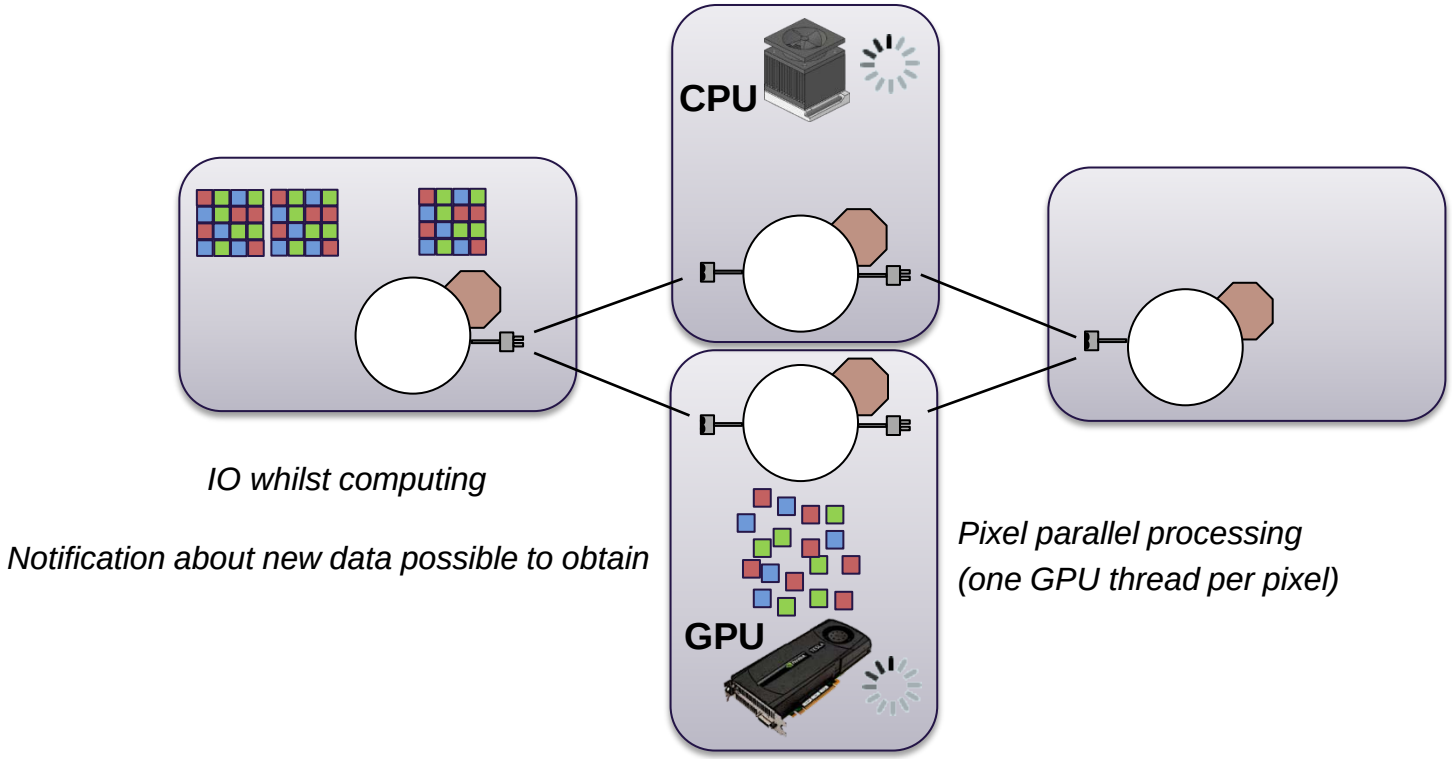
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



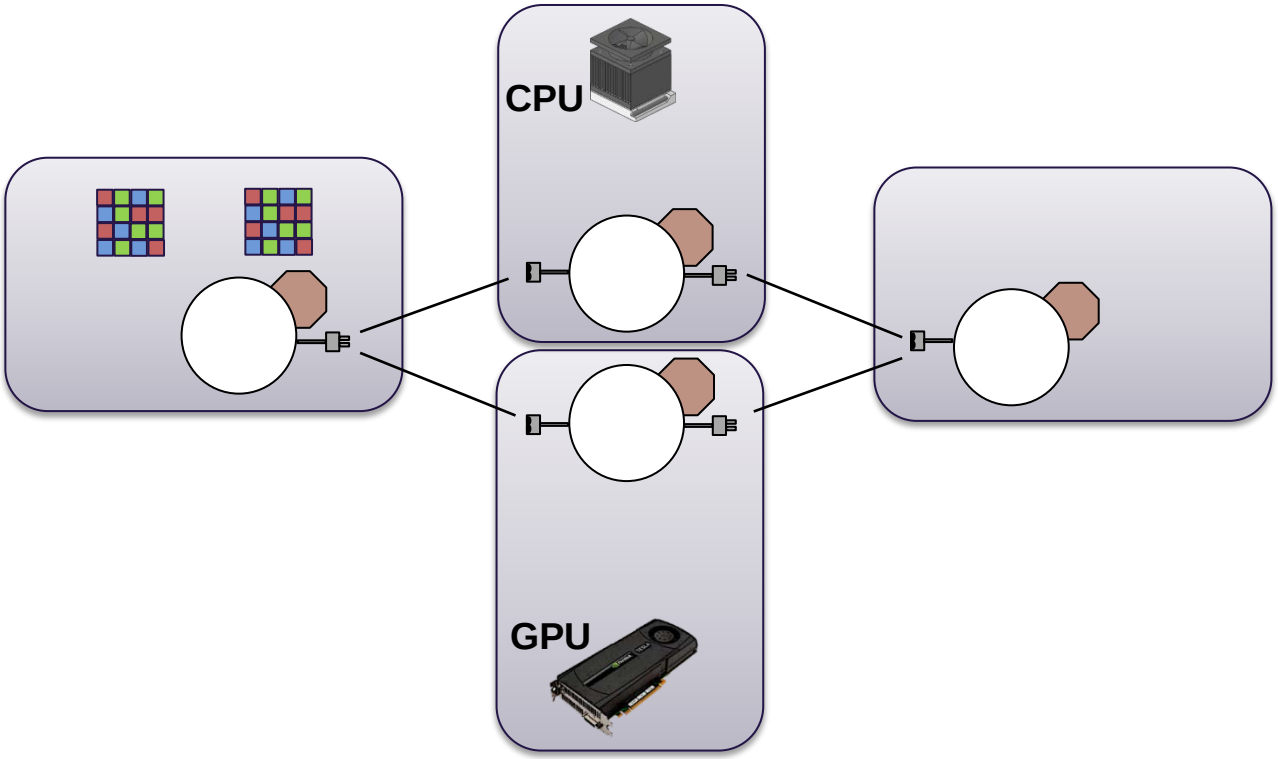
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



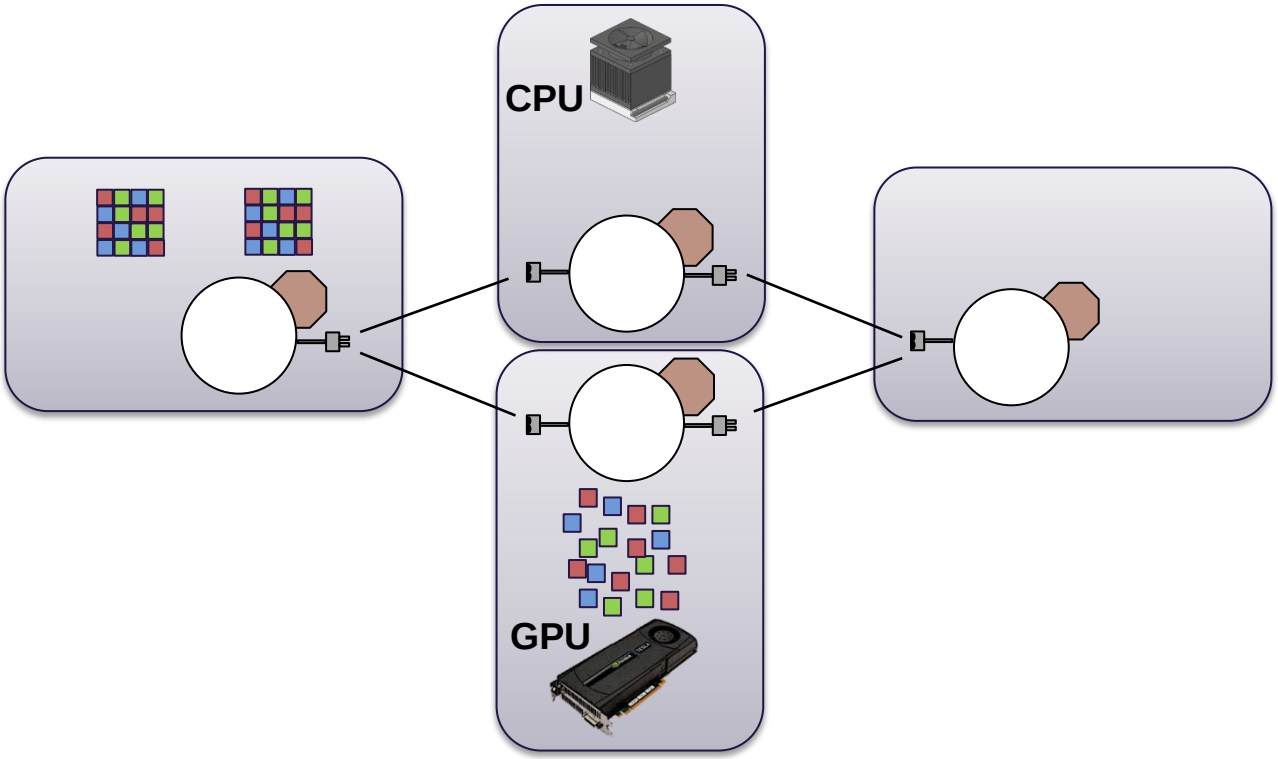
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

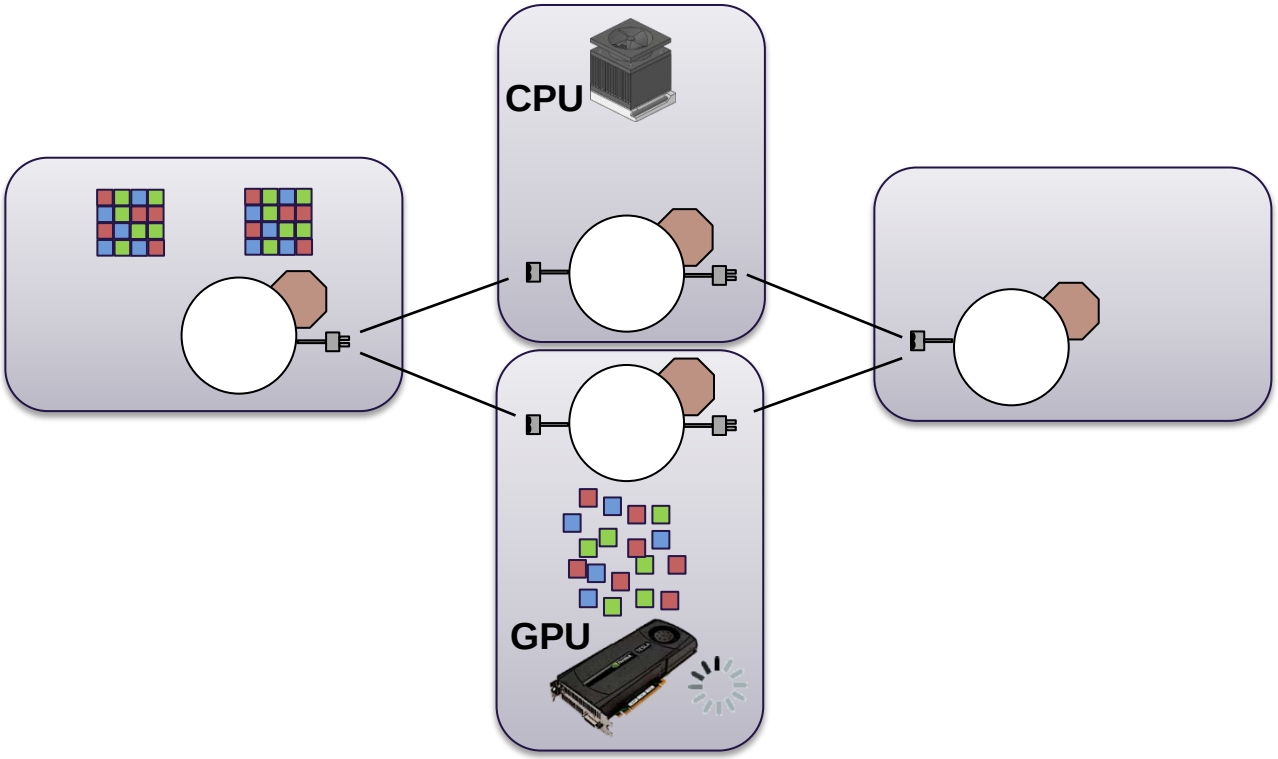
Workflows support load balancing



- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

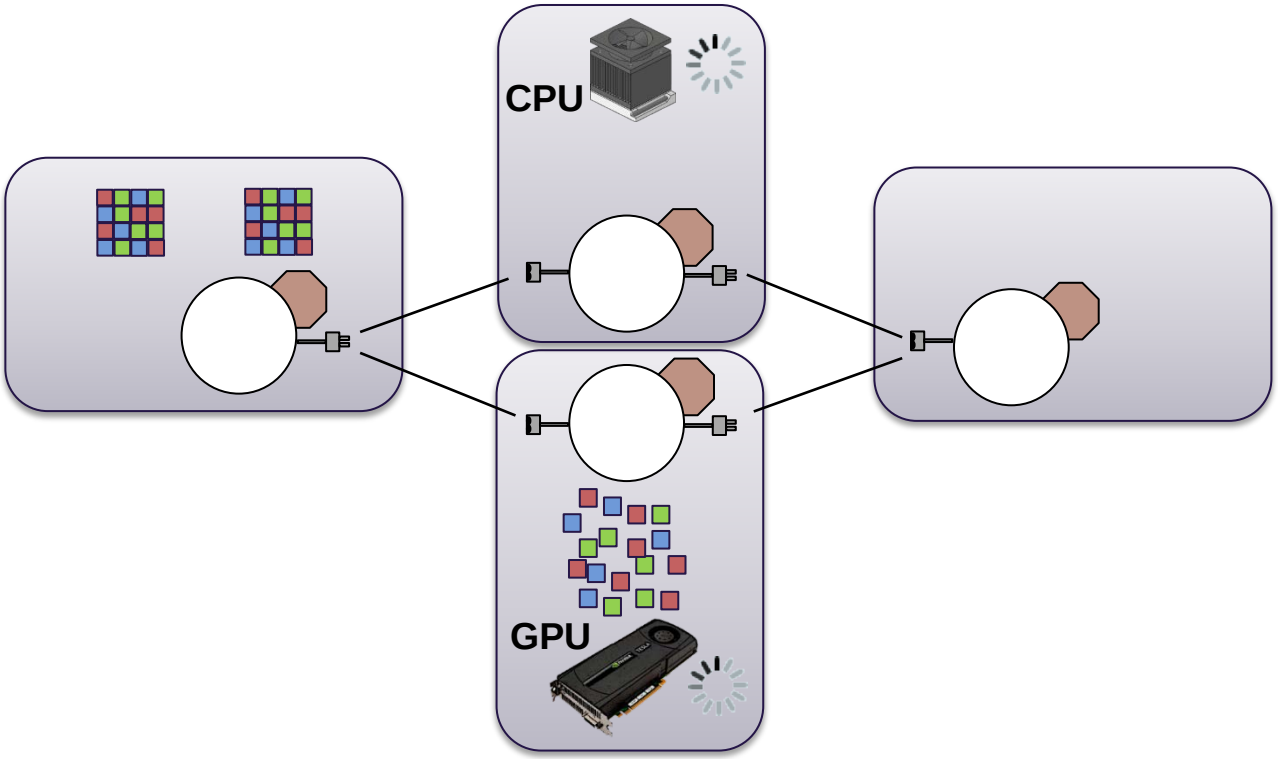
European XFEL

Workflows support load balancing



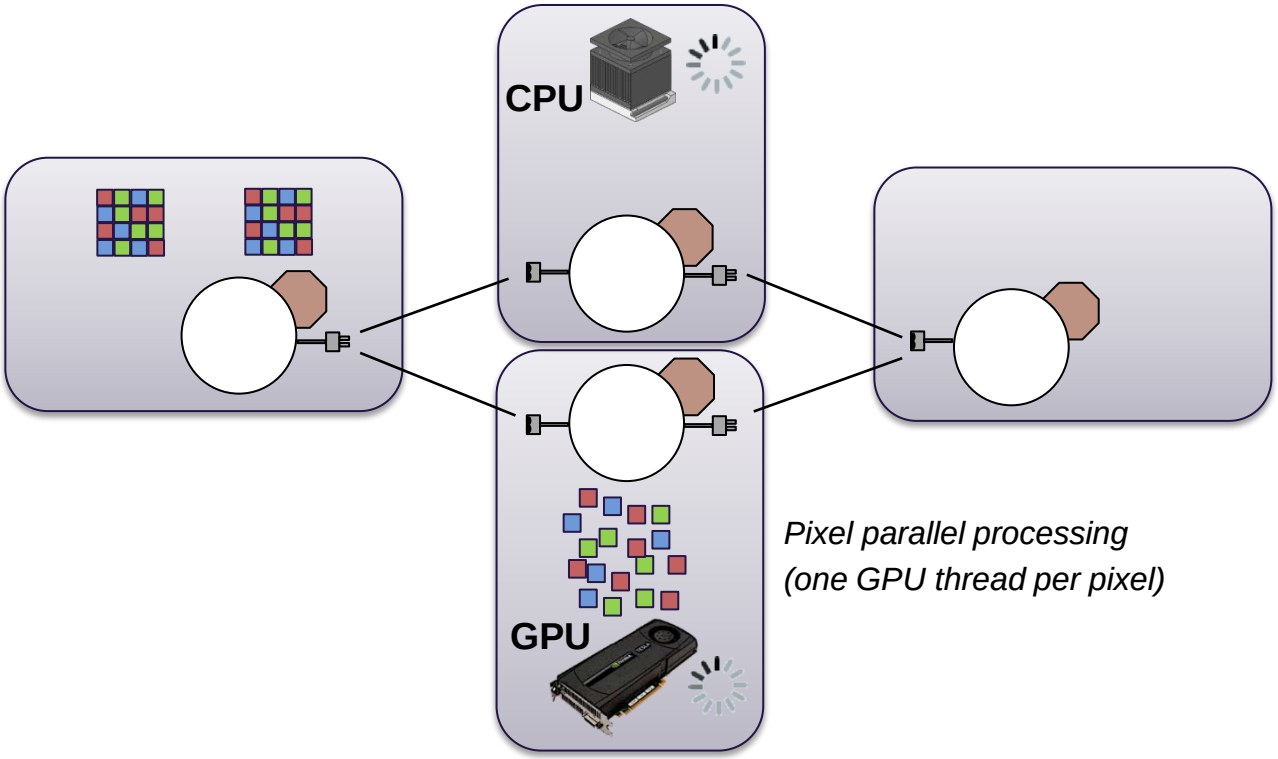
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



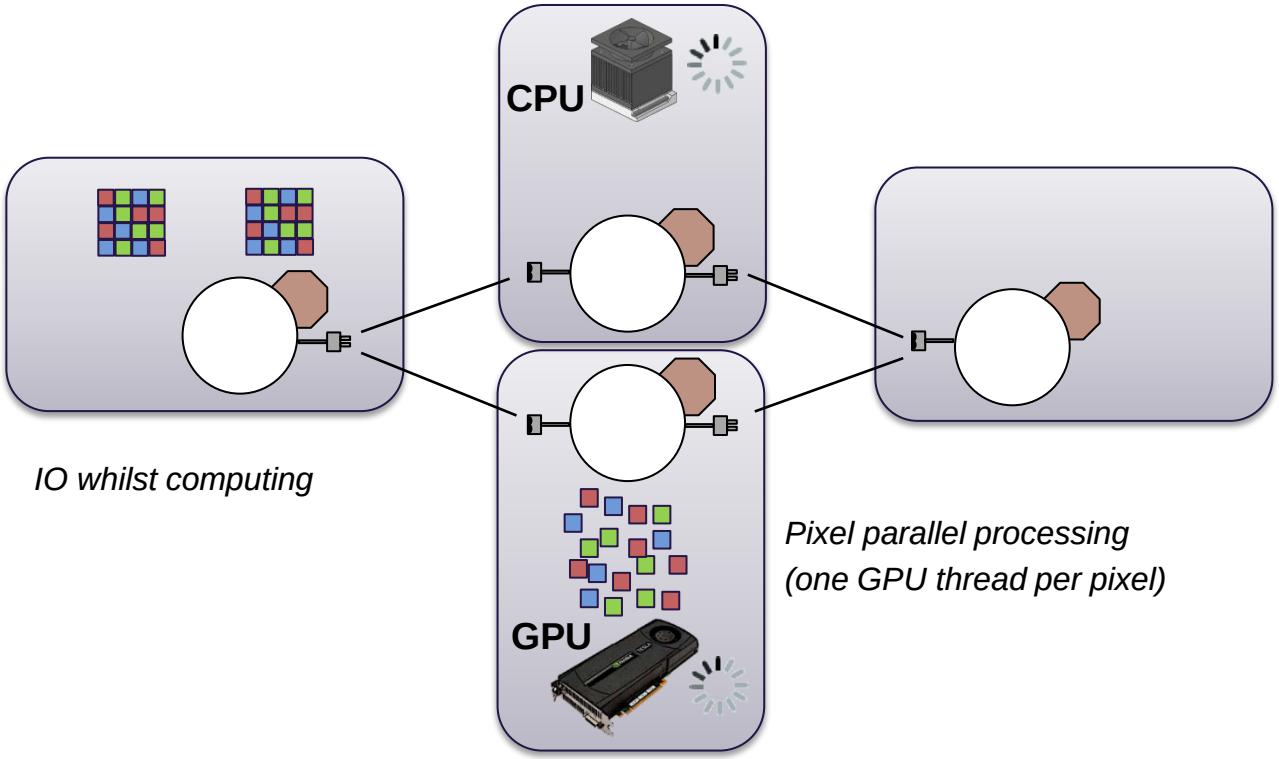
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



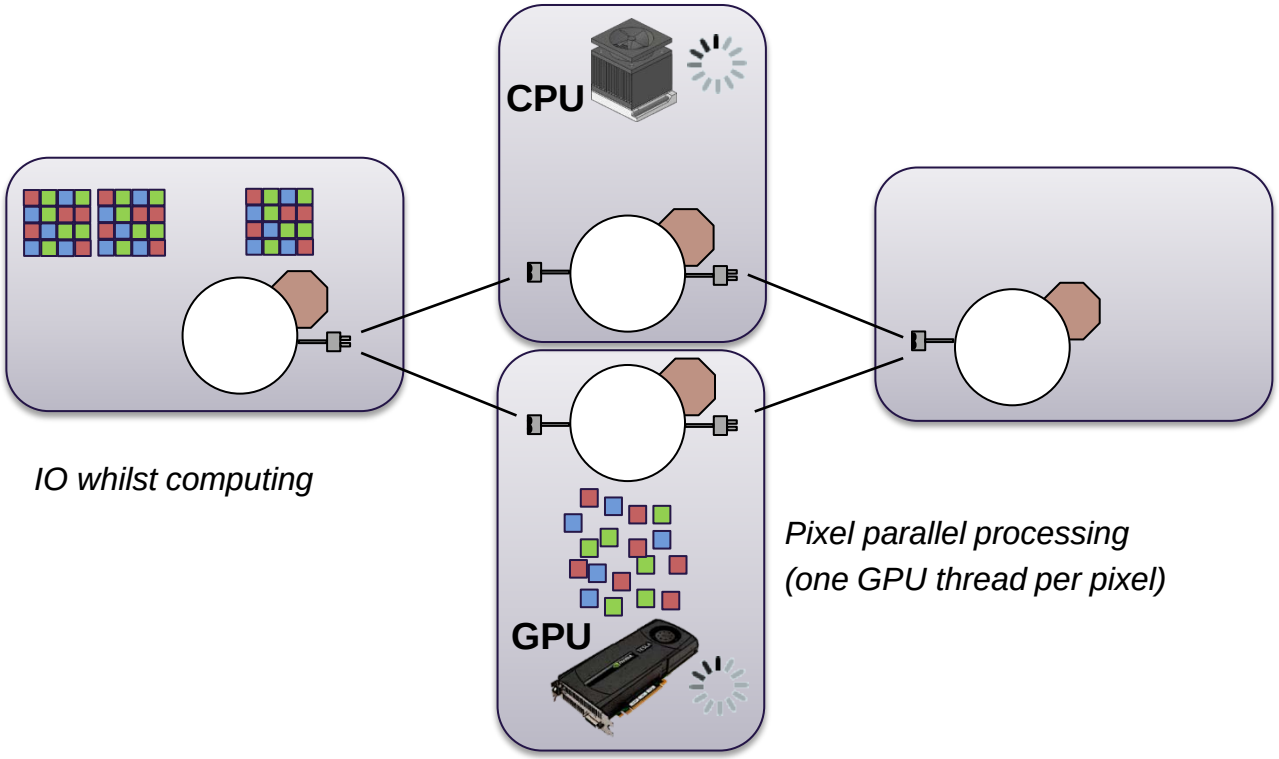
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



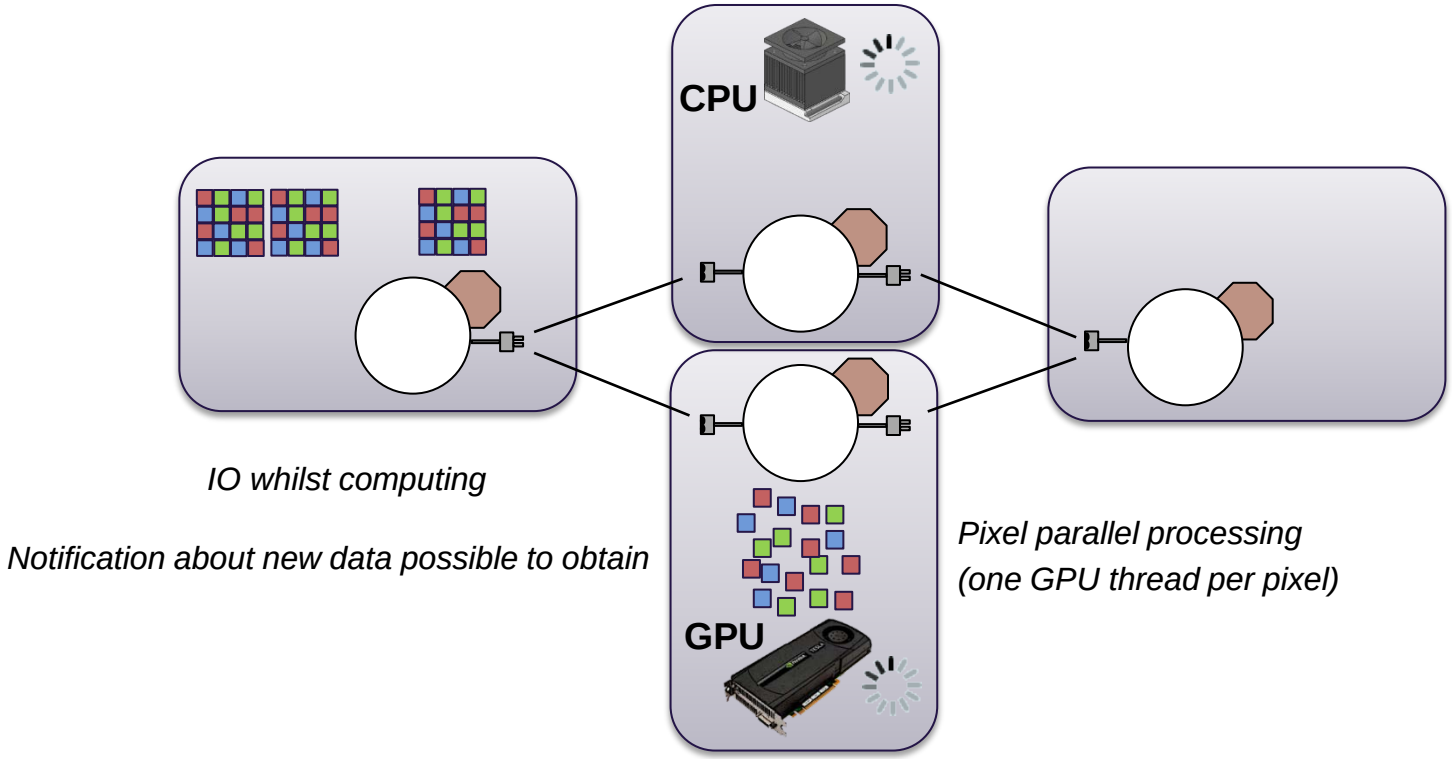
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



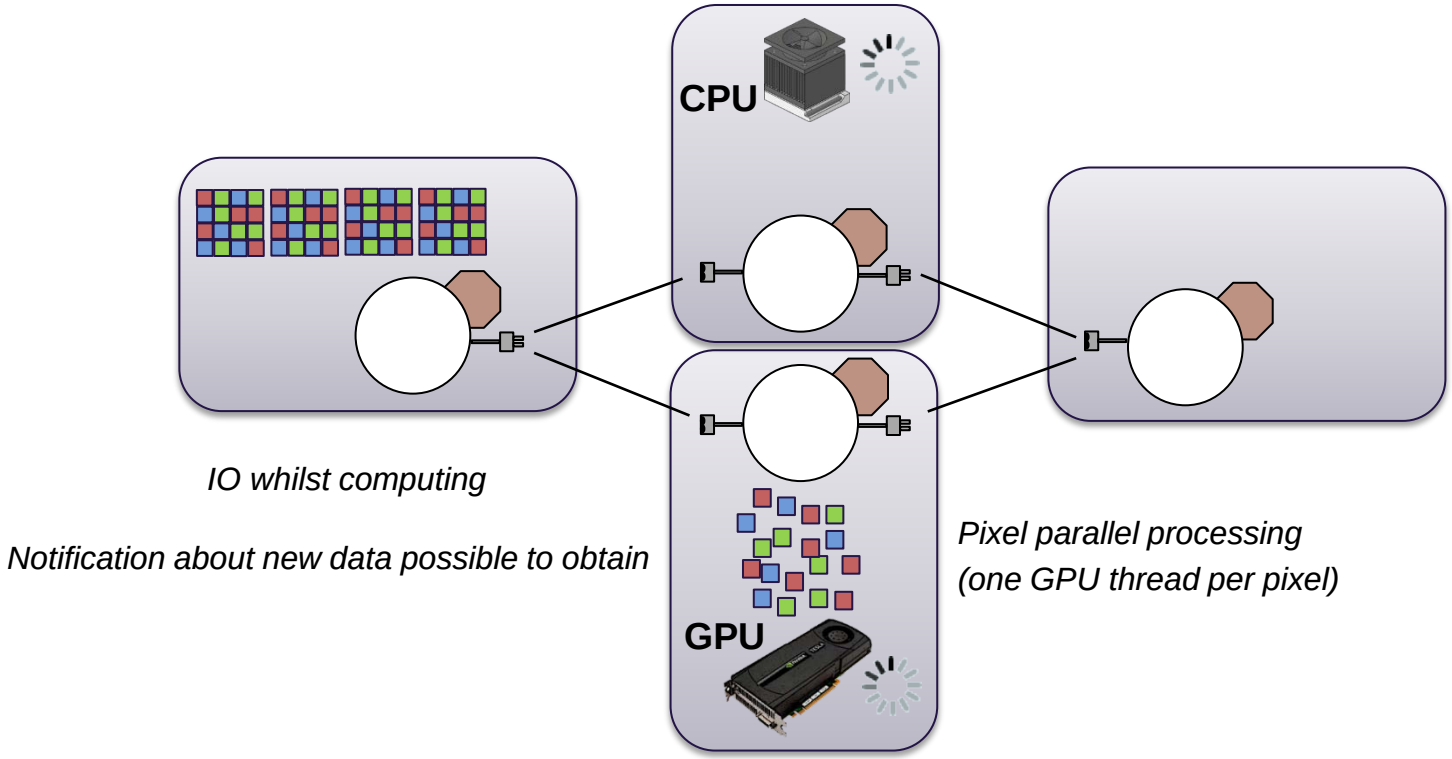
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



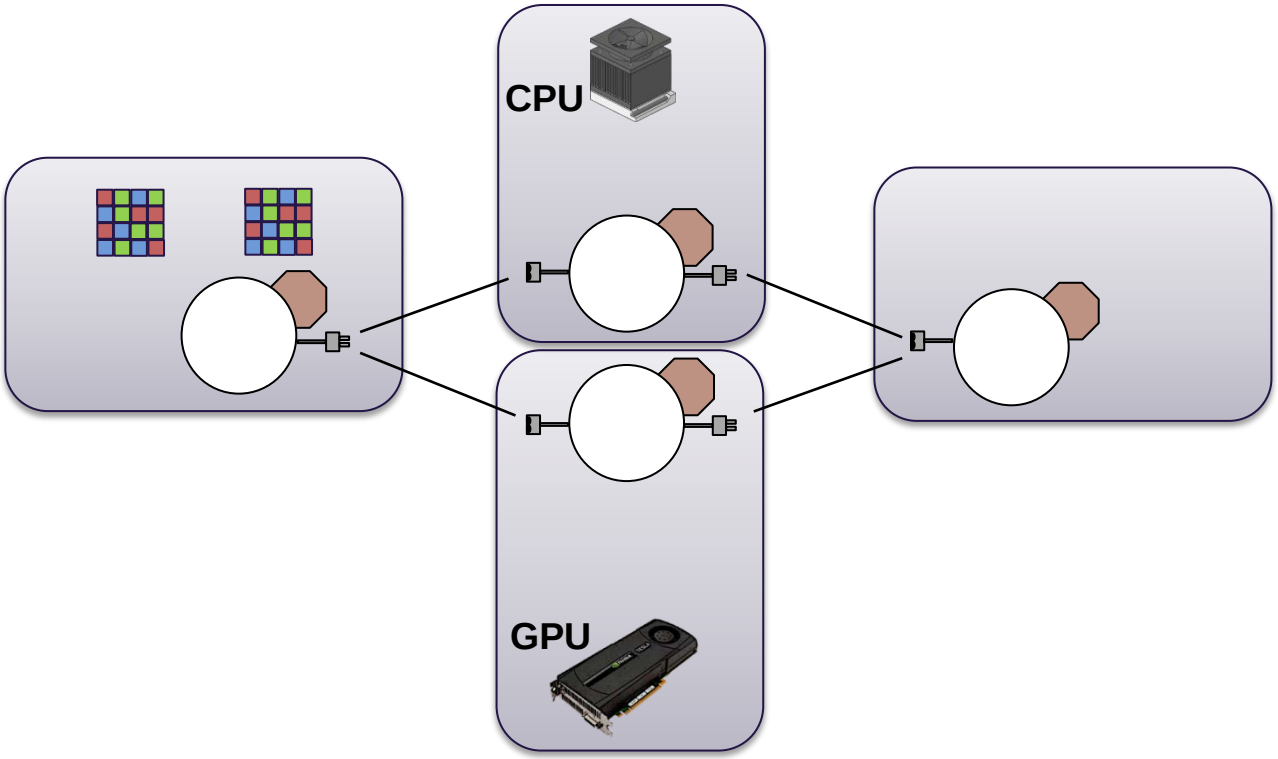
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

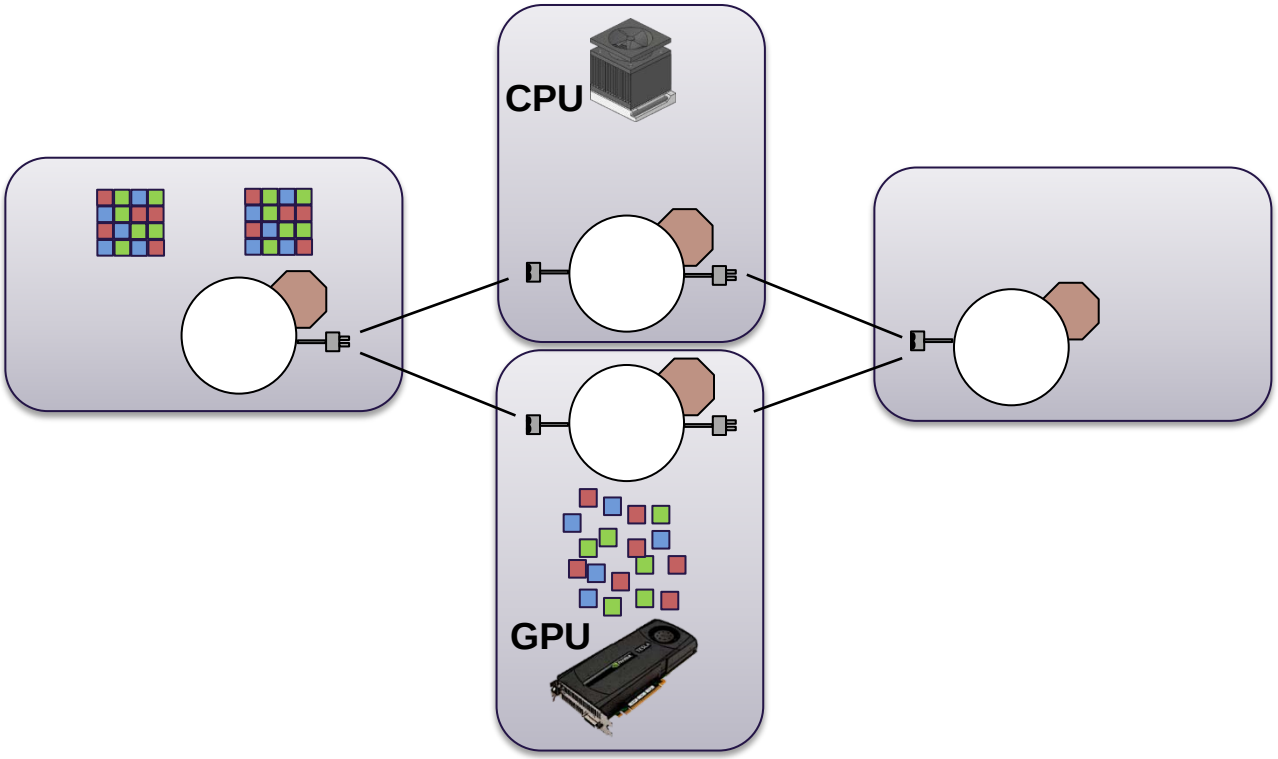
Workflows support load balancing



- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

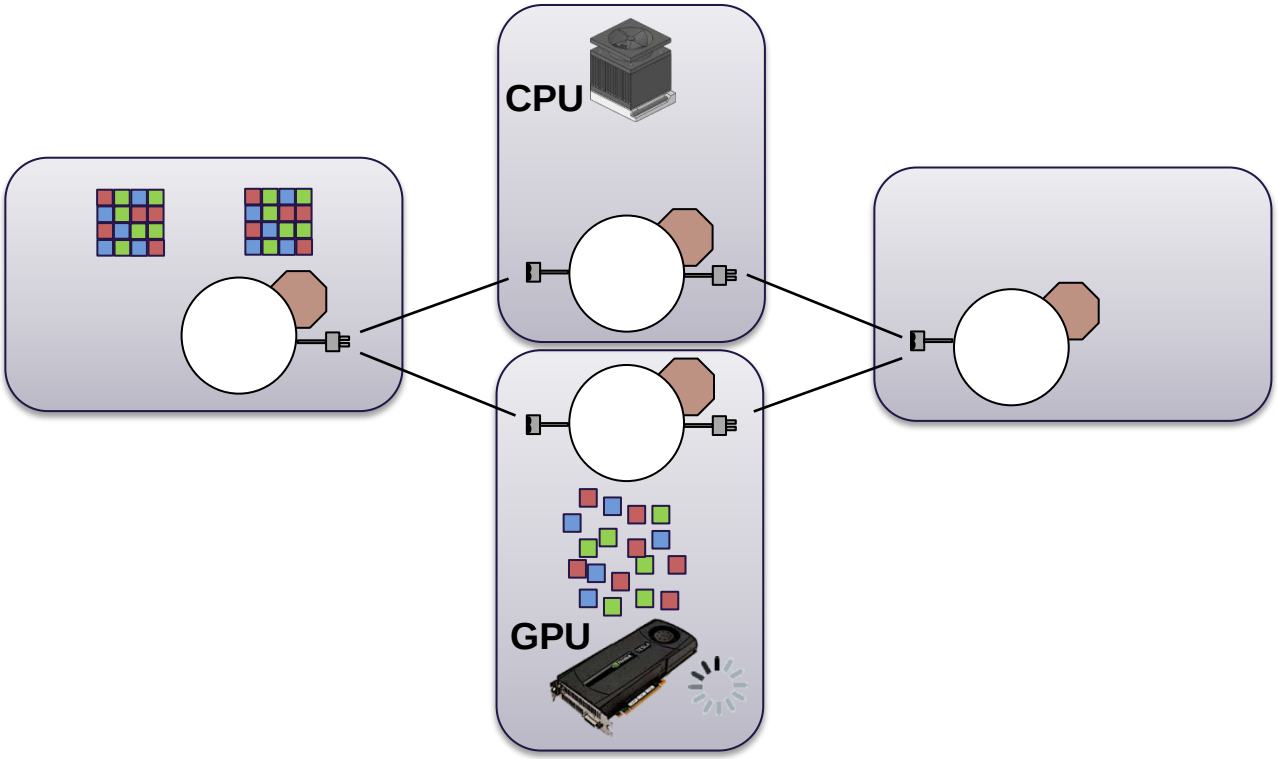
European XFEL

Workflows support load balancing



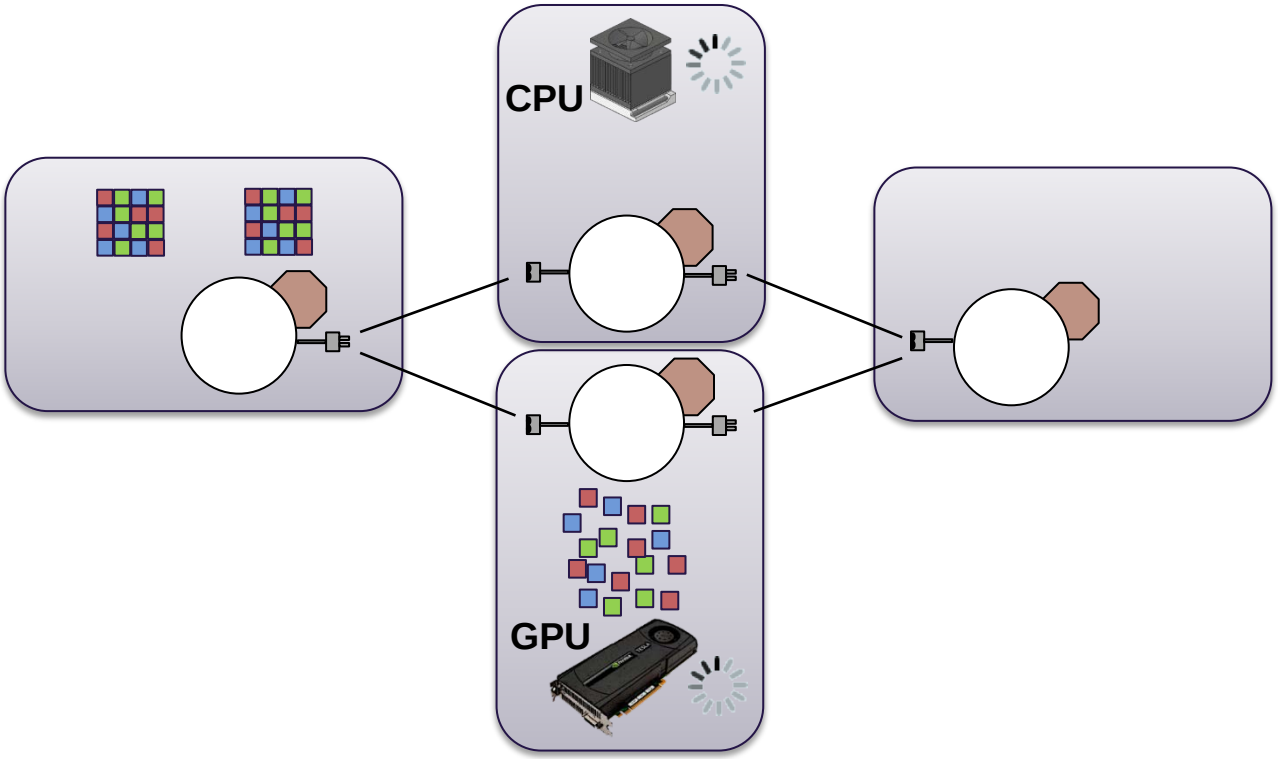
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

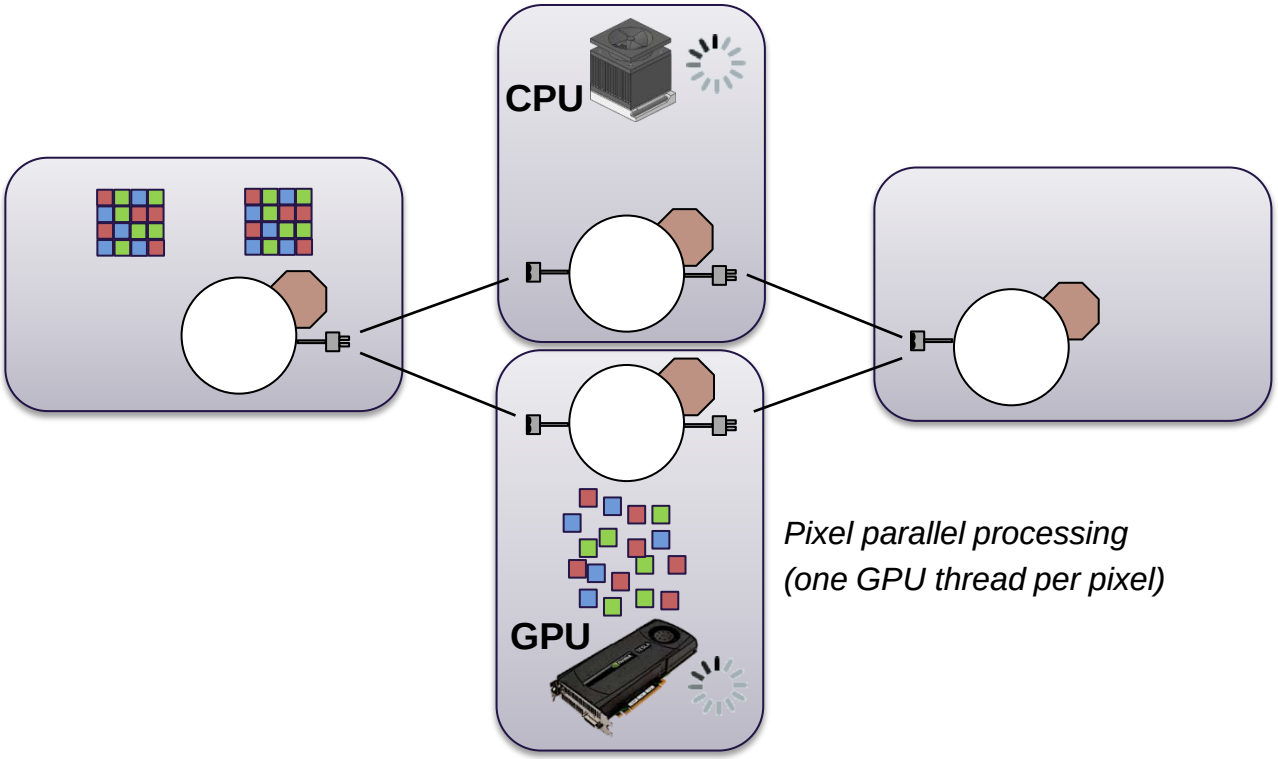
Workflows support load balancing



- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

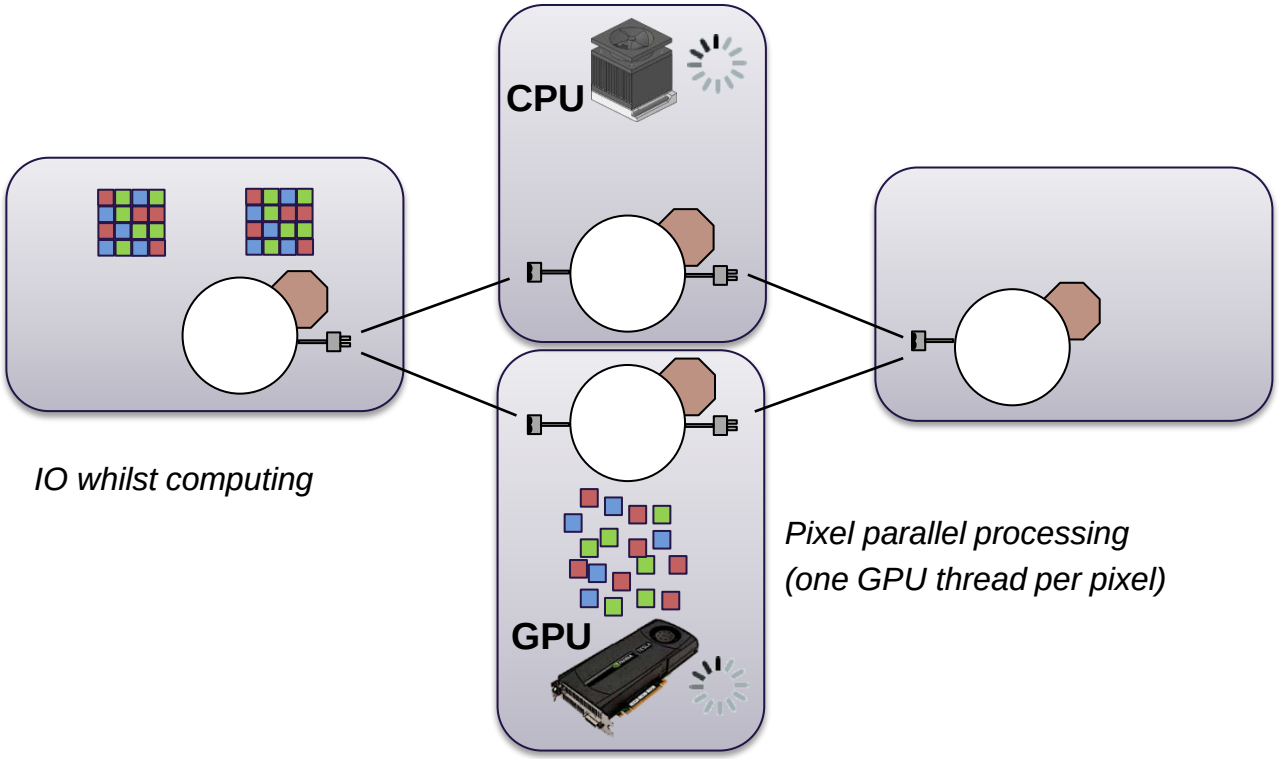
European XFEL

Workflows support load balancing



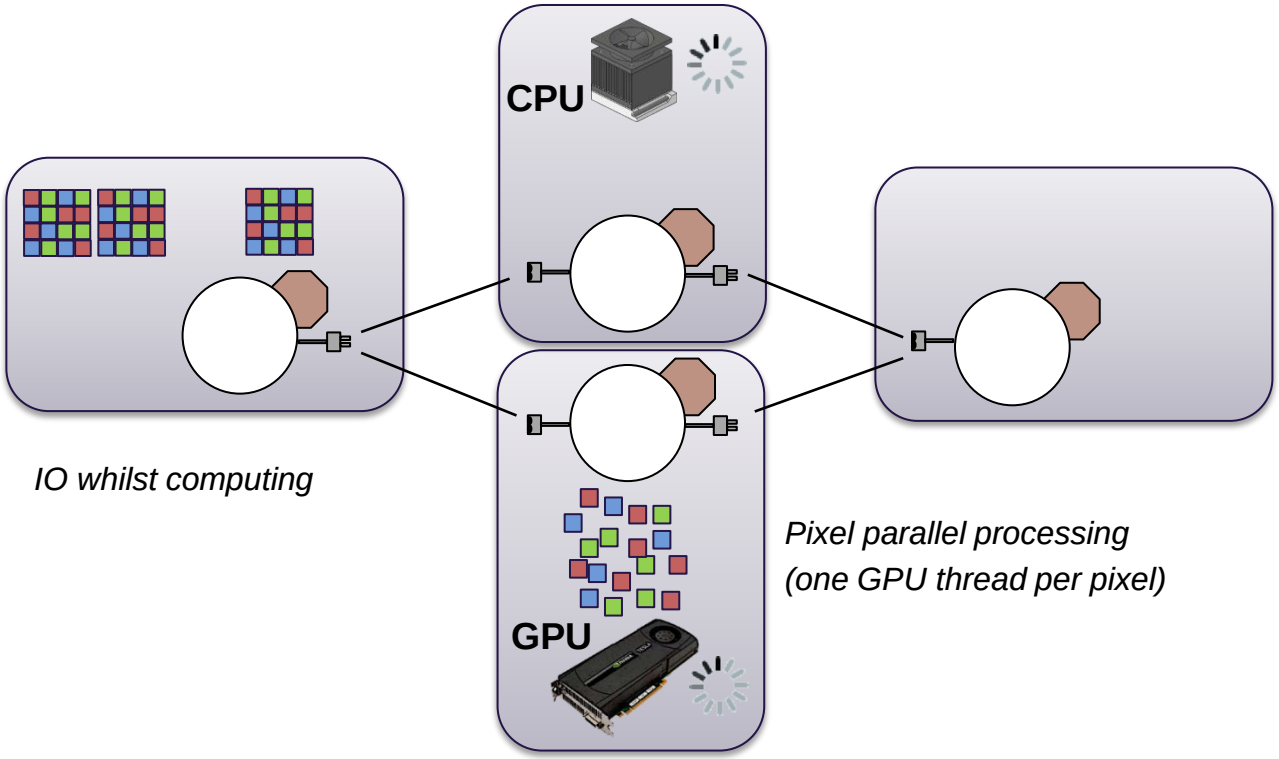
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



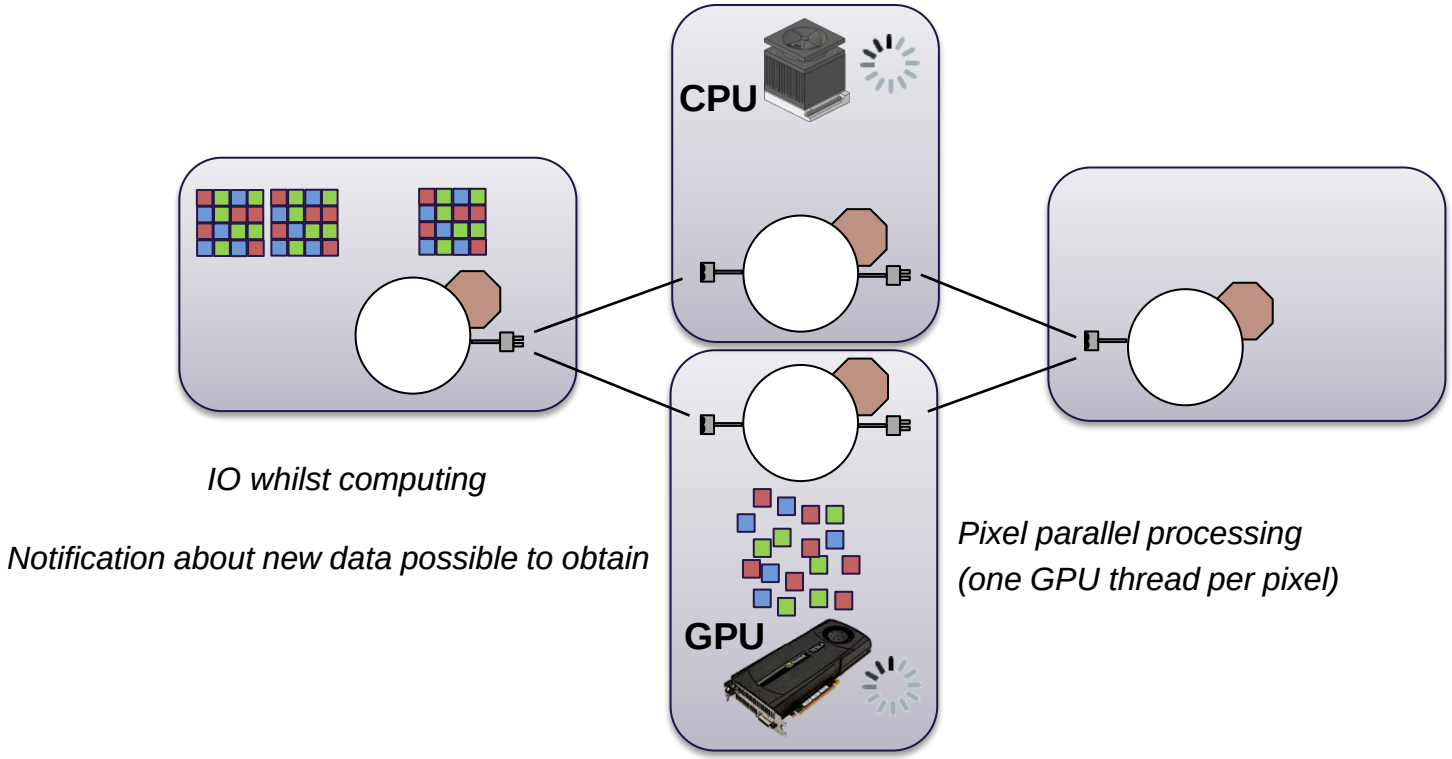
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



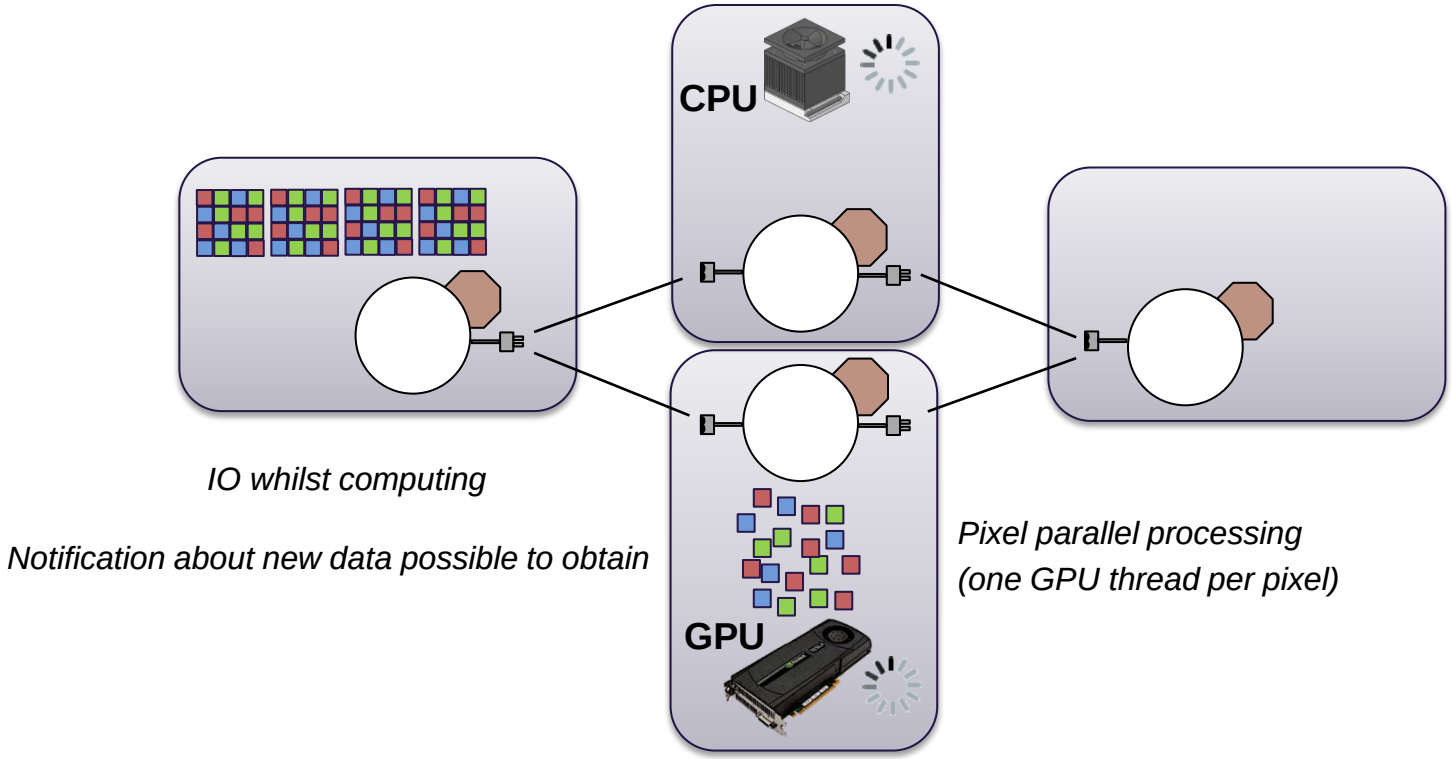
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



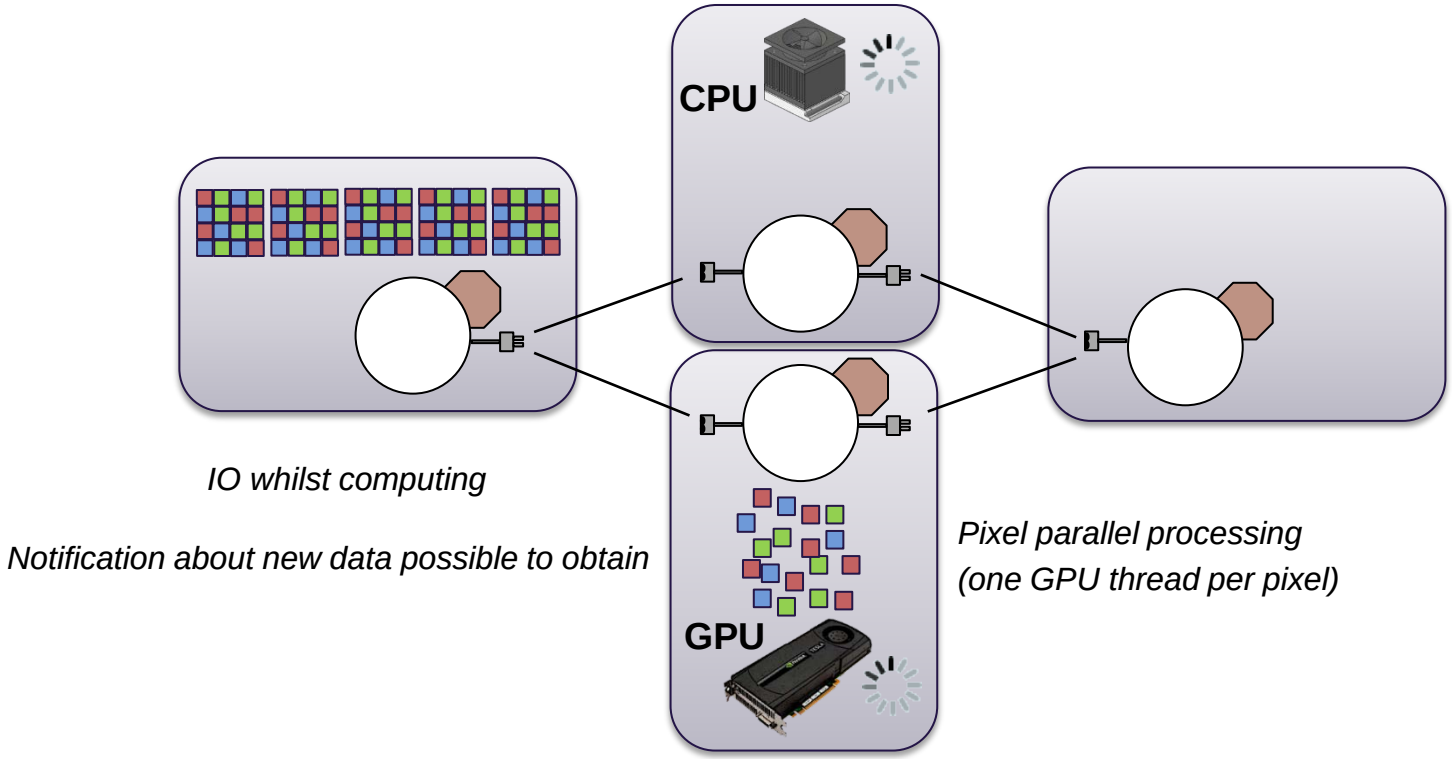
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing



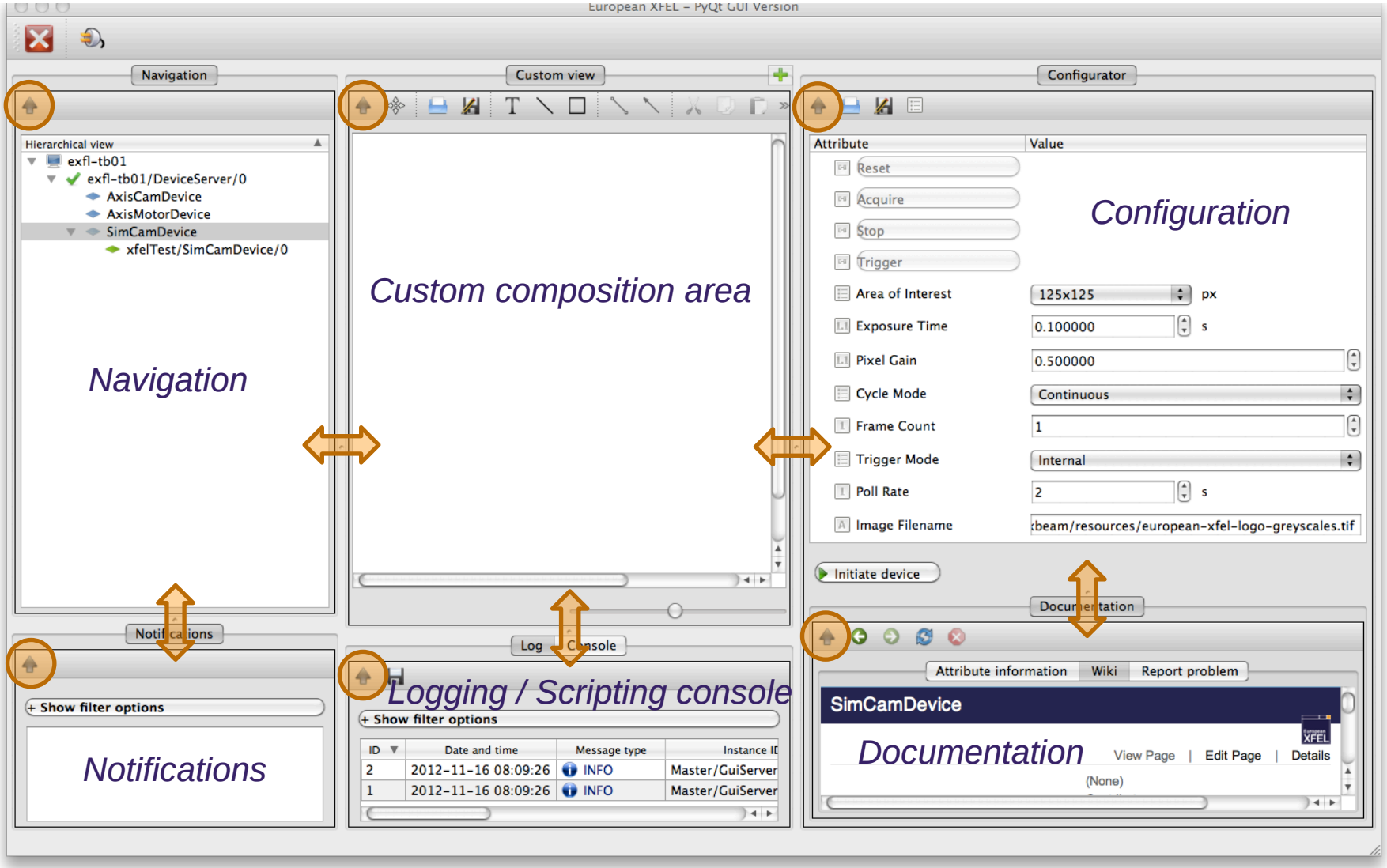
- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Workflows support load balancing

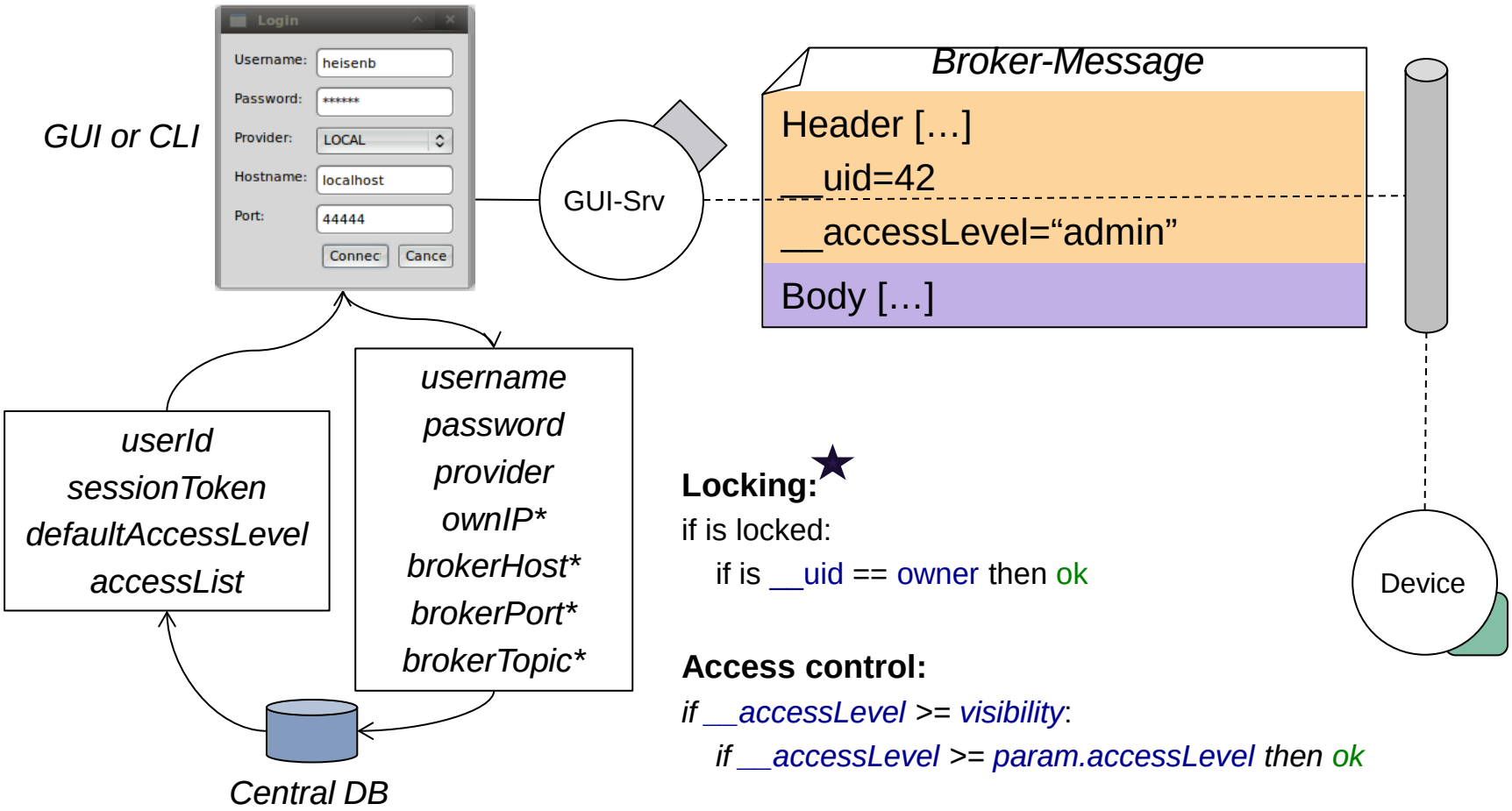


- Once resources are available, input channels request new data from connected output channels
- Configurable output channel behavior in case no input currently available: **throw, queue, wait, drop**

Multi-purpose graphical user interface



Before you even start: Login



Locking: ★

if is locked:

if is __uid == owner then ok

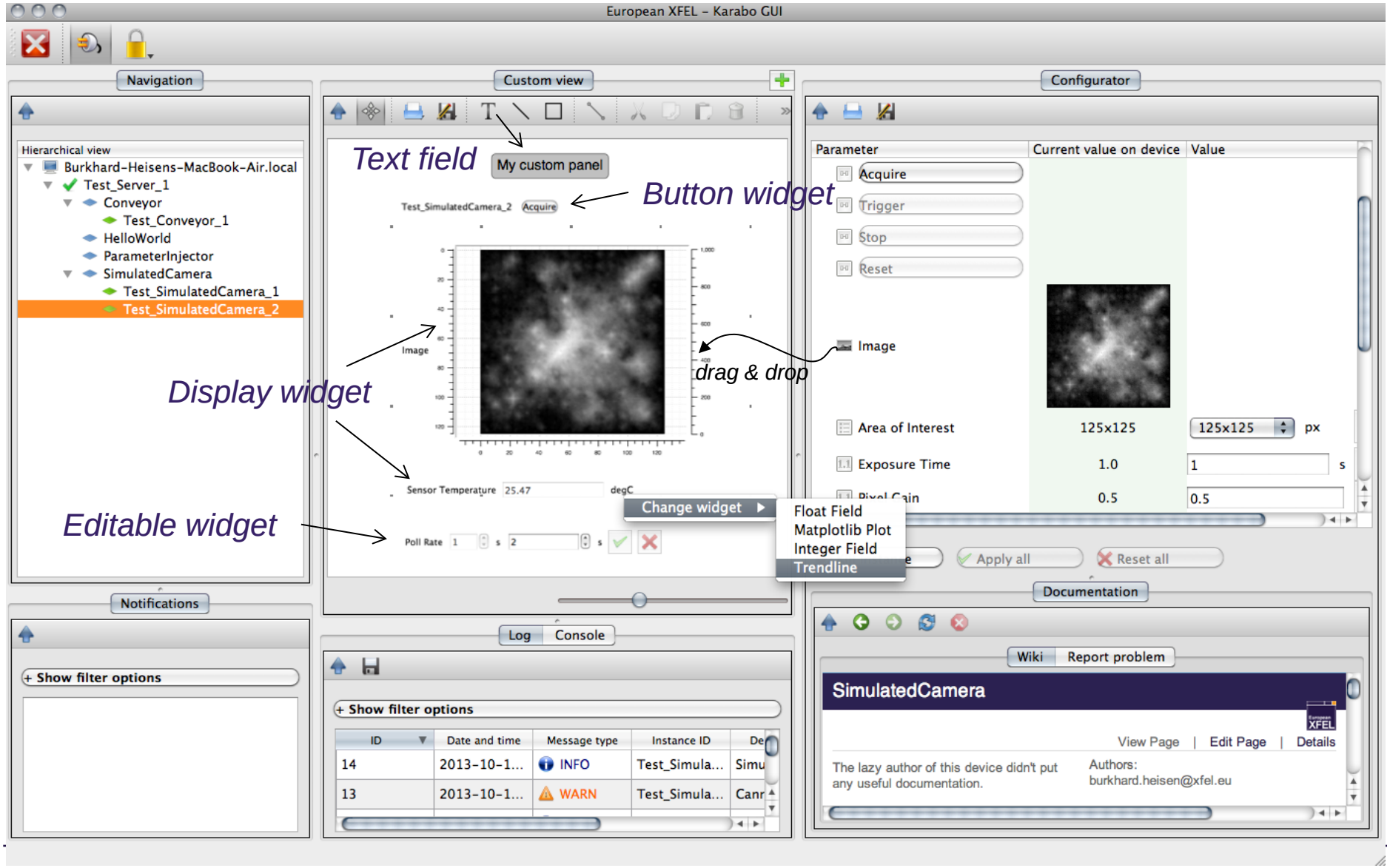
Access control:

if __accessLevel >= visibility:

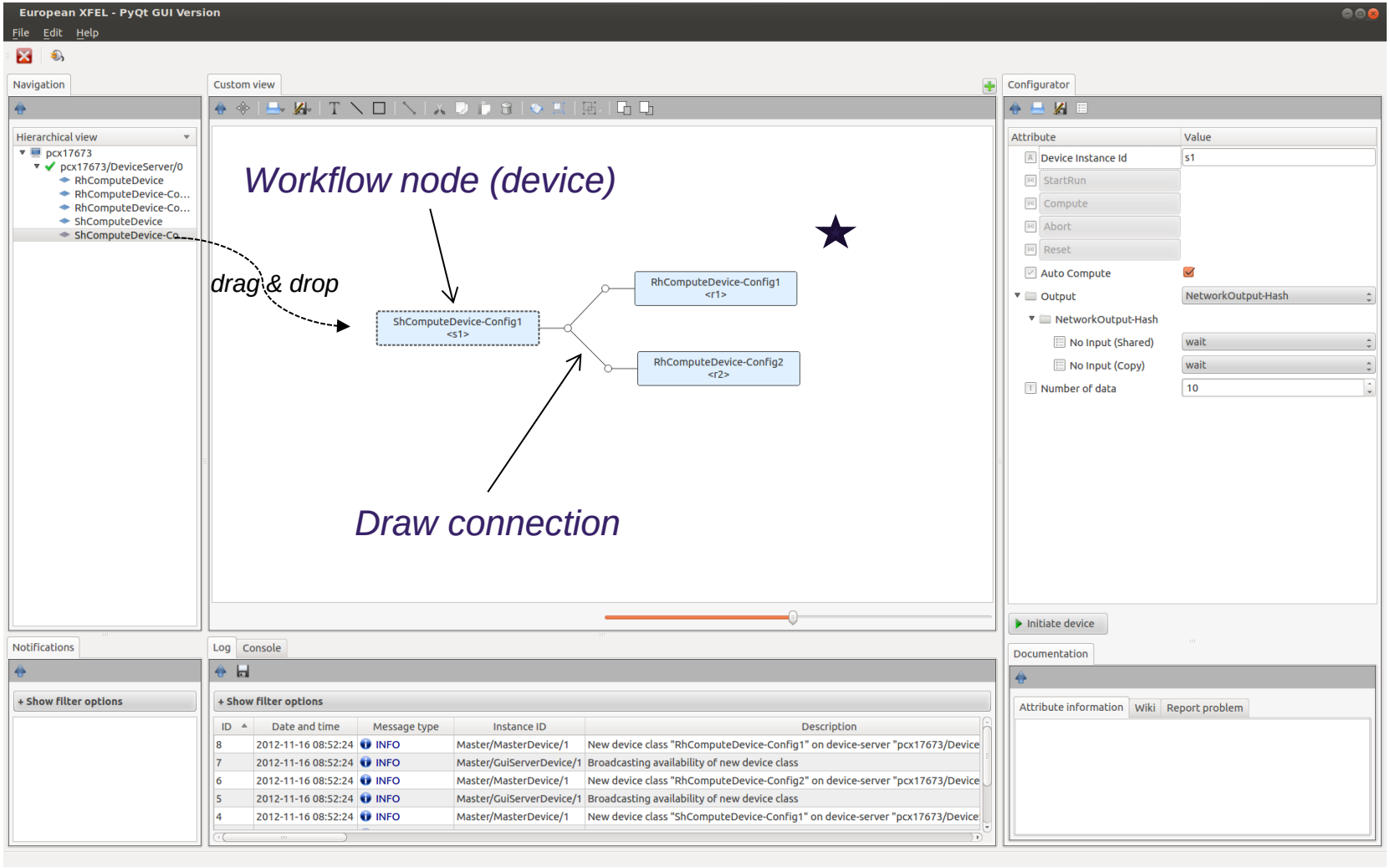
if __accessLevel >= param.accessLevel then ok

1. *Authorizes*
2. *Computes context based access levels*

Property/Command composition



Device (workflow) composition



The command line interface (CLI)

- Main functionality:
  - Exploring the distributed system topology (hosts, device-servers, devices, their properties/commands, etc.) **getServers**, **getDevices**, **getClasses**
  - **instantiate**, **kill**, **set**, **execute** (in “wait” or “noWait” fashion), **get**, **monitorProperty**, **monitorDevice**
  - Even polled interface will never really poll, but **is event-driven under the hood**
  - Remote auto-completion which is **access-role and state aware**

```
bheisen@Burkhard-Heisens-MacBook-Air: ~ — Python — 94×32
Python 2.7.3 (default, Apr 13 2012, 00:05:08)
Type "copyright", "credits" or "license" for more information.

IPython 0.13 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: c = DeviceClient()
    "Preparing for NSS initialization ..."
    "Initializing NSS ..."
    "Opened TCP connection to broker localhost:55655."
    "Connection ping enabled (ping interval = 20 second)."
    "Connection connected to broker"

In [2]: c.
c.execute                    c.killDeviceNoWait
c.executeNoWait              c.killServer
c.get                        c.killServerNoWait
c.getClassSchema             c.login
c.getClasses                 c.logout
c.getDeviceSchema            c.registerDeviceMonitor
c.getDevices                 c.registerPropertyMonitor
c.getFromPast                c.set
c.getServers                 c.setHash
c.help                       c.setNoWait
c.instantiate                c.show
c.instantiateNoWait          c.sleep
c.instantiateProject         c.unregisterDeviceMonitor
c.killDevice                 c.unregisterPropertyMonitor
```

The command line interface (CLI)

bheisen@Burkhard-Heisens-MacBook-Air: ~ — Python — 119×33

```
In [2]: c.help("
Test_Conveyor_1          Test_SimulatedCamera_1
Test_DataGenerator_1     Test_SimulatedCamera_2

In [2]: c.help("Test_DataGenerator_1")

----- HELP -----
Schema: DataGenerator

  .version (STRING)
     Assignment      : OPTIONAL
     Default value   : 1.0
     Description     : The version of this device class
     Access mode     : read only

  .connection (CHOICE_OF_NODES)
     Assignment      : OPTIONAL
     Default value   : Jms
     Description     : The connection to the communication layer of the distributed system

  .visibility (INT32)
     Assignment      : OPTIONAL
     Default value   : 0
     Description     : Configures who is allowed to see this device at all
     Access mode     : reconfigurable

  .classId (STRING)
     Assignment      : OPTIONAL
     Default value   : Device
     Description     : The (factory)-name of the class of this device
     Access mode     : read only
```

European XFEL

The command line interface (CLI)

bheisen@Burkhard-Heisens-MacBook-Air: ~ — Python — 99×26

```
In [3]: c.get("Test_Si
Test_SimulatedCamera_1  Test_SimulatedCamera_2

In [3]: c.get("Test_SimulatedCamera_1", "
areaOfInterest                      connection.Jms.port               imageFilename
cameraAcquiring                     connection.Jms.protocol           pixelGain
cameraModel                         connection.Jms.serializationType  pollRate
classId                             connection.Jms.trustBroker        progress
connection.Jms.acknowledgeMode      connection.Jms.username           sensorHeight
connection.Jms.acknowledgeSent      cycleMode                         sensorTemperature
connection.Jms.acknowledgeTimeout   deviceId                          sensorWidth
connection.Jms.deliveryInhibition   exposureTime                      serverId
connection.Jms.destinationName      frameCount                        sta
connection.Jms.hostname             image.channelSpace                tri
connection.Jms.messageTimeToLive    image.data                        ver
connection.Jms.messagingDomain      image.dims                        vis
connection.Jms.password             image.encoding
connection.Jms.ping                 image.isBigEndian

In [3]: c.get("Test_SimulatedCamera_1", "sen
sensorHeight       sensorTemperature  sensorWidth

In [3]: c.get("Test_SimulatedCamera_1", "sensorTemperature")
Out[3]: 25.49
```

bheisen@Burkhard-Heisens-MacBook-Air: ~ — Python — 56×11

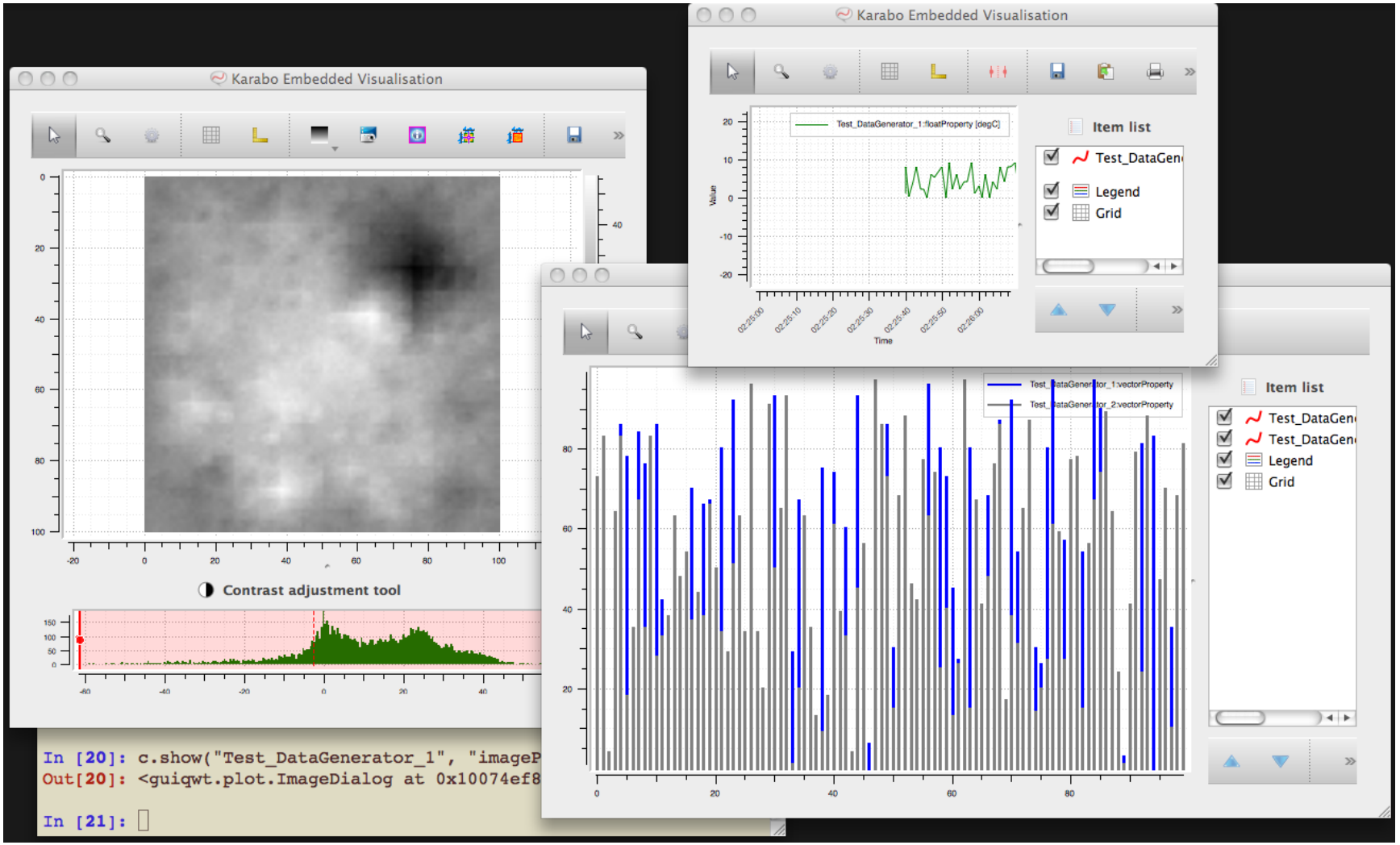
```
In [8]: c.execute("Test_SimulatedCamera_1", "acquire")
Out[8]: (True, 'Ok.Acquisition')

In [9]: c.execute("Test_SimulatedCamera_1", "
stop     trigger

In [9]: c.execute("Test_SimulatedCamera_1", "stop")
Out[9]: (True, 'Ok.Ready')
```

bheisen@Burkhard-Heisens-MacBook-Air: ~ — Python — 97×17

```
In [4]: c.set("Test_Sim
Test_SimulatedCamera_1  Test_SimulatedCamera_2

In [4]: c.set("Test_SimulatedCamera_1", "
areaOfInterest  exposureTime    pixelGain       triggerMode
cycleMode       frameCount      pollRate        visibility

In [4]: c.set("Test_SimulatedCamera_1", "exposureTime", 0.5)
Out[4]: (True, '')

In [5]: c.set("Test_SimulatedCamera_1", "triggerMode", "nop")
Out[5]:
(False,
 'Value nop for parameter "triggerMode" is not one of the valid options: Internal,Software\n')
```

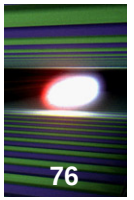
European XFEL

The command line interface (CLI)



European XFEL

Selected Features



- Cross-network **signals and slot** implementation (Qt style)
  - Native function calls providing native argument passing
  - Runtime setup of signals, slots and connect statements
- High performance **dictionary object**
  - Fully recursive key to any value mapper (also in C++)
  - Keeps insertion order and supports attributes
  - Serializes to XML, Binary, HDF5, DB, JMS-Message, etc.
- **Event driven system throughout**, incl. **event driven archiving**
  - Archiving happens completely transparent (no extra tooling needed)
  - Archiving policies are configured per property within device code
- **User centric, access controlled** system
  - **Context based authentication** (who, where, when)
  - Kerberos is integrated
- Full system available in C++ and “pythonic” Python
  - Devices and workflows can intermix both languages (access to **numpy, scipy**, etc.)
- Fully functional also without master or database (good for testing/developing)
- Easy install (both framework and devices) using **software bundle** architecture

European XFEL

Acknowledgments

Karabo contributors (internal)

K. Wrona
K. Weger
D. Boukhelef
A. Parenti
S. Hauf
N. Coppola
J. Elizondo
C. Youngman
S. Esenov
J. Szuba
I. Kozlova
L. Maia
P. Gessler
J. Tolkiehn
H. Höhne

Karabo contributors (external)

C. Yoon (CFEL)
T. Nicholls (STFC)
…
N. D. Loh (SLAC)
K. Rehlich (DESY / DOOCS)

Thank you for your kind attention.