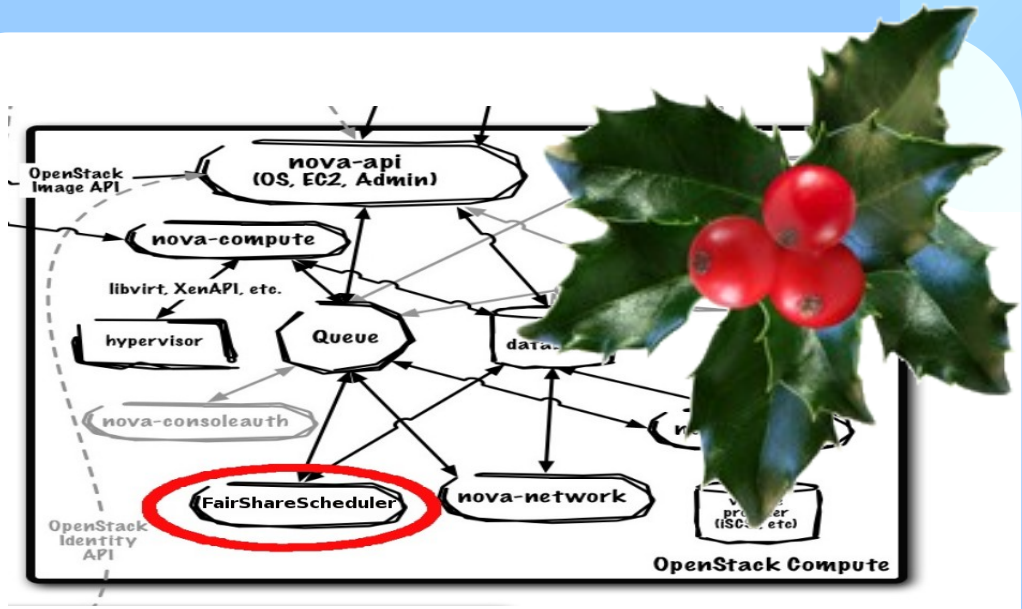
CLOUD AREA PADOVANA

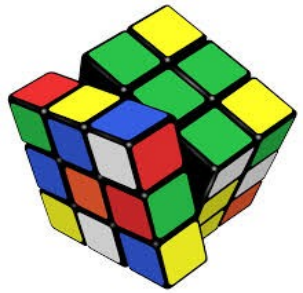


Lisa Zangrando
INFN Padova

The FairShareScheduler

A problem 20 years old reappears today

- Before the batch system affirmation (~20 years ago) our resources were partitioned only statically
  - agreed amount of resources always available



- Usually our requests for resources is much greater than the amount of the available resources
  - it becomes necessary to seek to maximize their utilization by adopting a proper resource sharing model

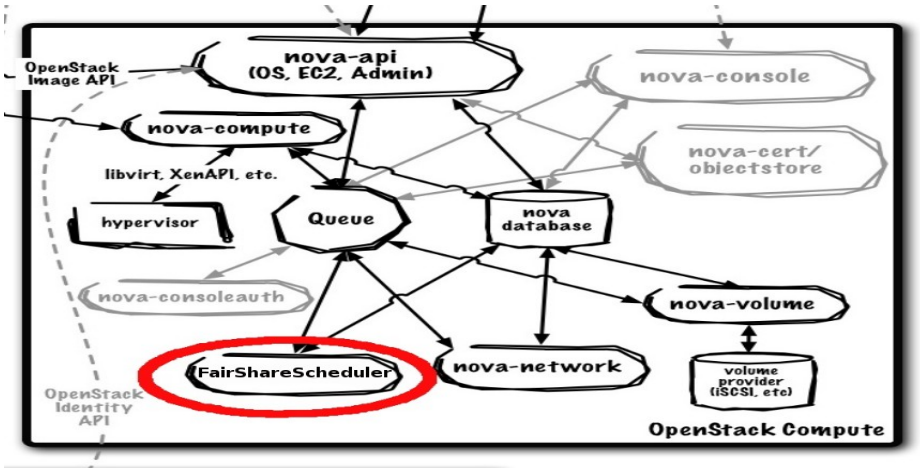
- The solution comes with the batch system adoption: dynamic partitioning
  - provision of an average computing capacity to be guaranteed during a long period (~1 year)
  - advanced scheduling algorithms (i.e. fair-share)

Partitioning model in OpenStack

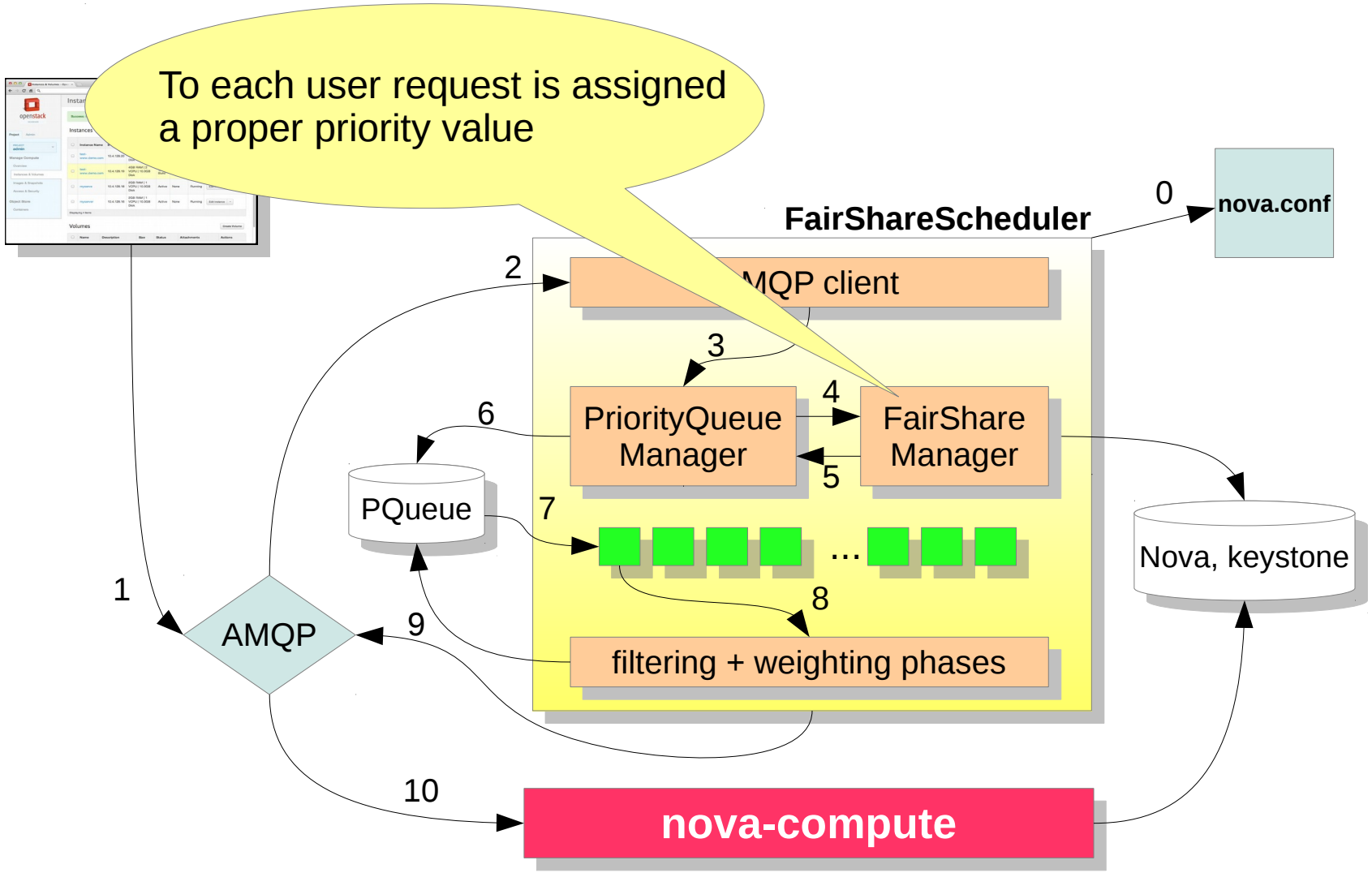
- Currently OpenStack allows just the static partitioning model
  - resource allocation can be done only by granting fixed quotas
  - one project cannot exceed its own quota even if there are unused resources allocated to other projects
  - low global efficiency and increased cost in terms of resource usage
  - it doesn't allow continuous full utilization of all available resources
- in a scenario of full resource usage for a specific project, new requests are simply rejected
- we need to find a better approach to enable a more effective and flexible resource allocation and utilization in OpenStack

Our proposal: the FairShareScheduler

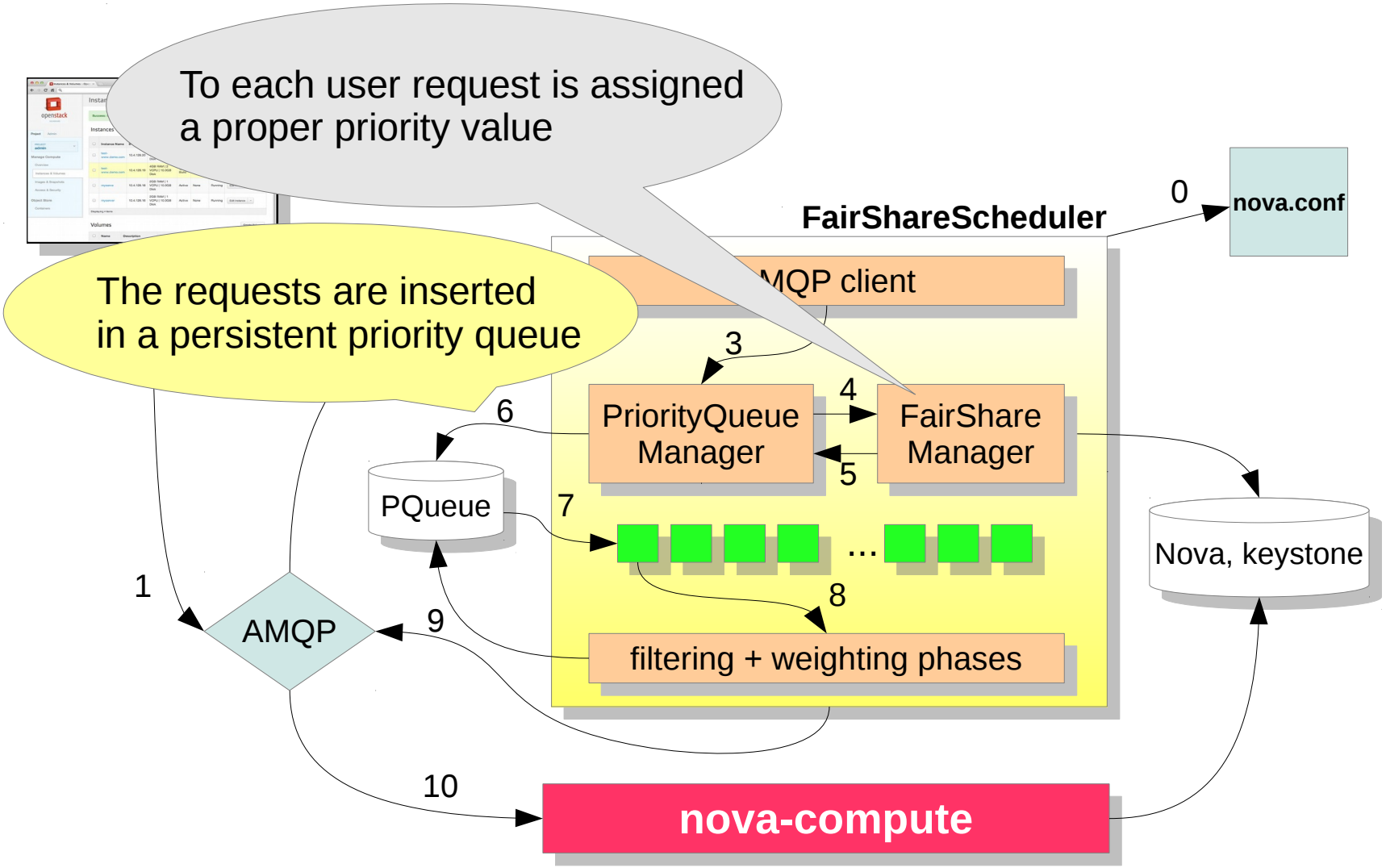
- We (INFN-PD) started to address the problem by developing a pluggable scheduler, named FairShareScheduler, as extension of the current OpenStack scheduler (i.e FilterScheduler)
- FairShareScheduler provides:
  - queuing mechanism for handling the user requests
  - fair-share algorithm based on the SLURM Priority MultiFactor strategy



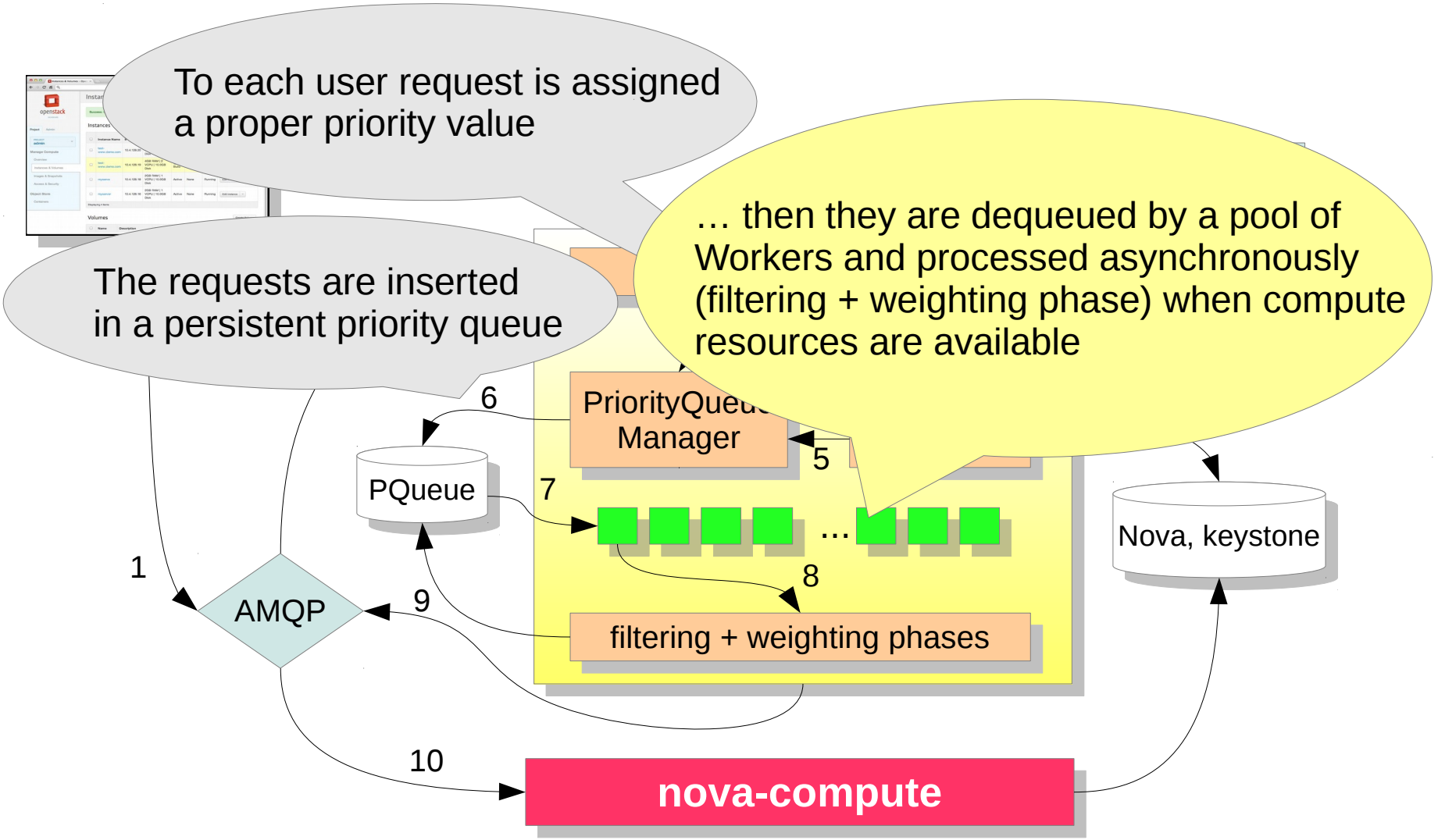
FairShareScheduler: the high level architecture



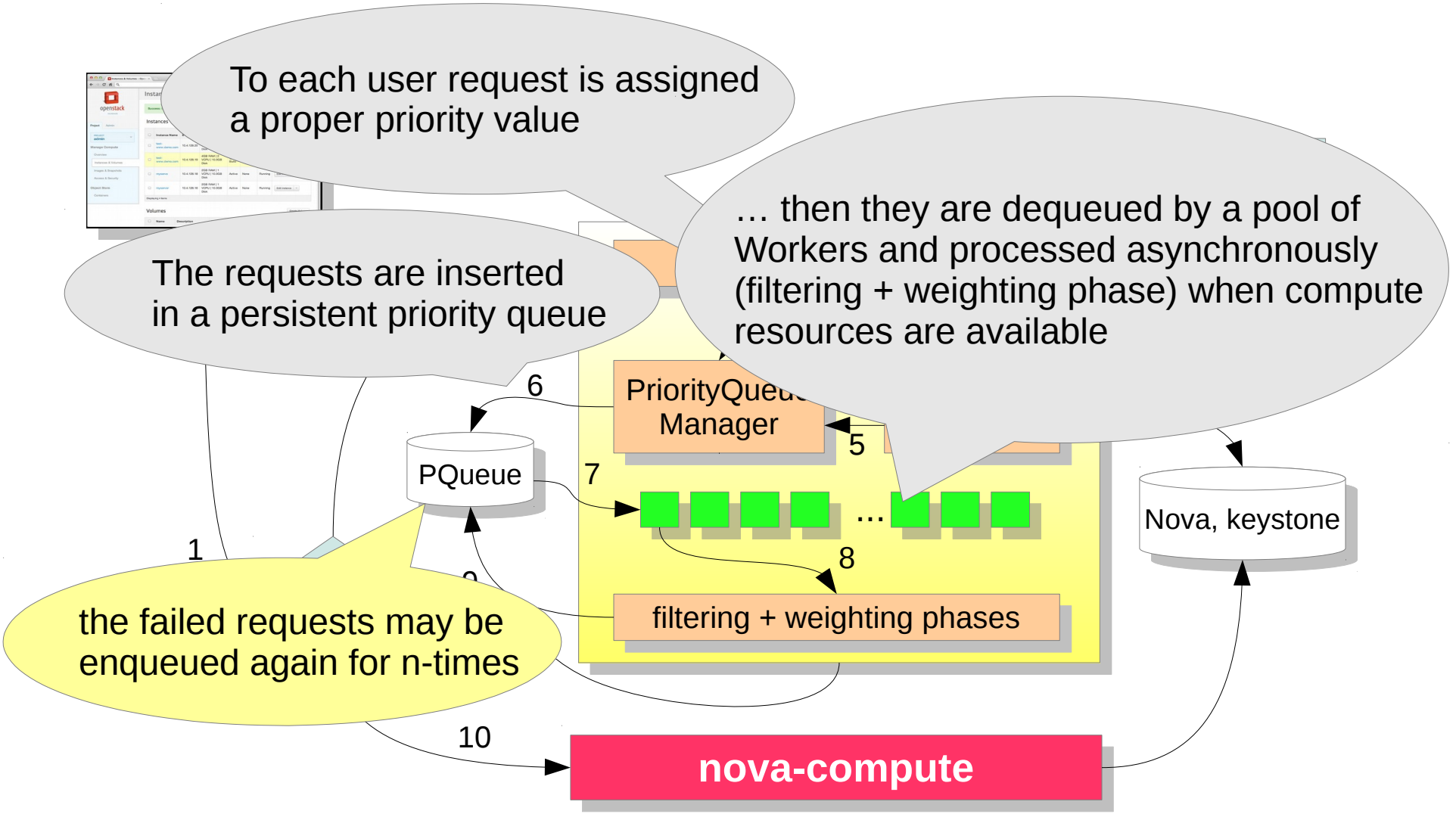
FairShareScheduler: the high level architecture



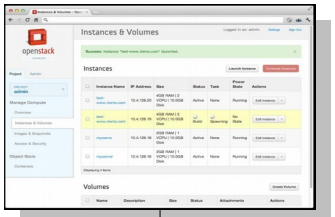
FairShareScheduler: the high level architecture



FairShareScheduler: the high level architecture



FairShareScheduler: the high level architecture



- **no new states have been added**
  - this prevents any possible interaction issue with the OpenStack clients
  - from the client point of view the queued requests remain in “Scheduling” state till the compute resources are available
- **the priority of the queued requests is periodically recalculated**

Current status

- Prototype ready for HAVANA and IceHouse
  - source code available in our github repository: https://github.com/CloudPadovana/openstack-fairshare-schedule
- Testing in progress at:
  - University of Victoria
  - BILS (Bioinformatics Infrastructure for Life Sciences)

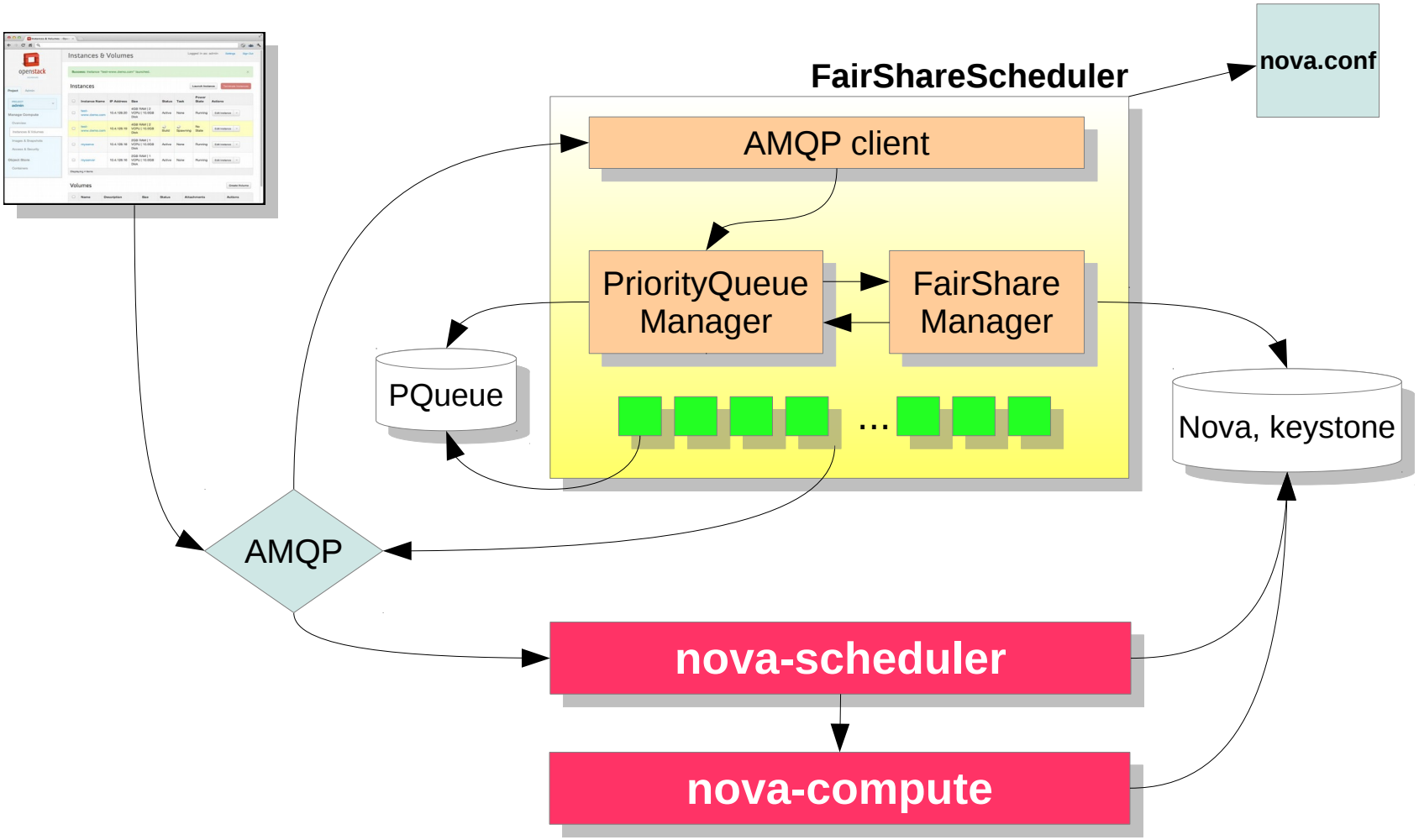
Integration in OpenStack

- We wish to join the OpenStack development team in order to integrate our work in the official distribution

- Started interaction with NOVA, GANTT and BLAZAR teams
  - not an easy task address our needs with such groups

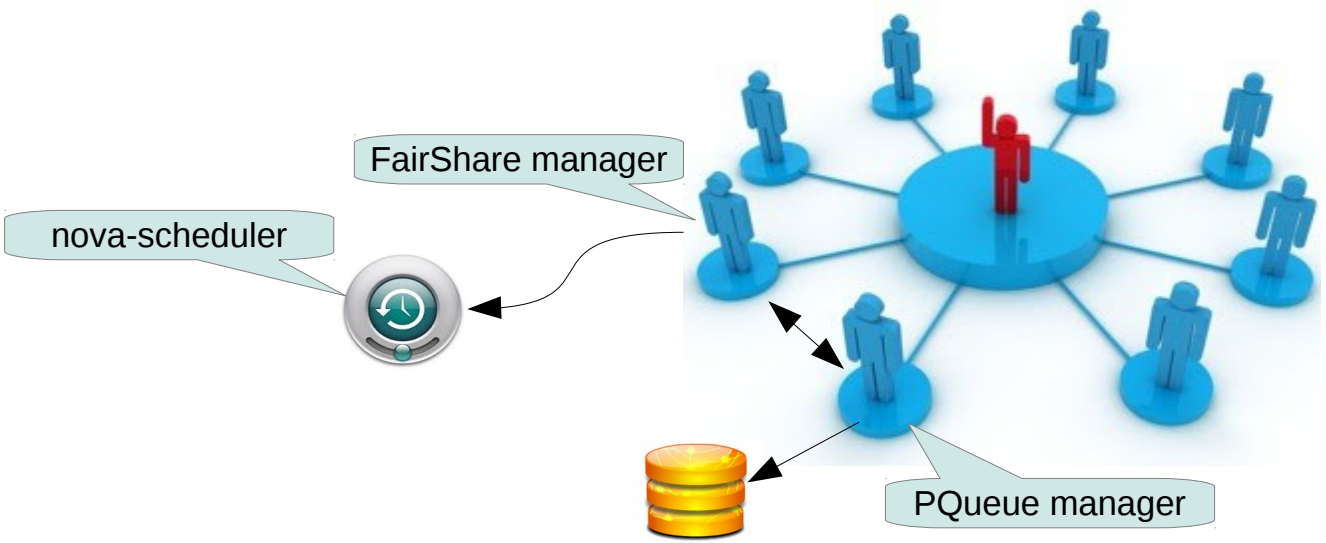
- At the OpenStack Summit 2014 (Paris) the Nova-Scheduler leader suggested us to consider the FairShareScheduler as an external manager which interacts with Nova-Scheduler
  - this approach should simplify the integration process
  - architecture redesign and redevelopment work required
  - new stackforge project creation

FairShareScheduler: architecture redesign



The evolution

- The possibility of designing the FairShareScheduler as indipendent project allows us to evolve it a little bit more as a new OpenStack service which handles pluggable managers
  - APIs for interacting with the service and the managers
  - a manager is a specific and independent task executed periodically or interactively
    - you can implement your manager by using the provided API
    - one manager will provide the same capabilities of the current FairShareScheduler



The manager interface

```
class Manager(Thread):
    def getName(self): # return the manager name
    def getStatus(self): # return the manager status
    def isAutoStart(self): # is AutoStart enabled or disabled?
    def setup(self): # allows custom initialization
    def destroy(self): # invoked before to destroy the manager
    def execute(self, cmd): # executes user command synchronously
    def task(self): # executed periodically at fixed rate
```

The evolution: status

- Development started (IceHouse, Juno)
  - foreseen two months of work for the first prototype
- New project at StackForge as soon the prototype is ready
  - interaction with other OpenStack projects not required
- StackForge provides a place for OpenStack contributors to create and maintain unofficial projects using the same tools and procedures as the official ones
  - it includes: Gerrit code review, Jenkins CI, GitHub repository, IRC...
  - it is a good first step for exposing new projects but doesn't guarantee eventual OpenStack incubation and integration
  - the new project must be self sufficient
- Incubation and integration phases need the approval of the OpenStack reviewers
- University of Victoria available to help with testing


Have a Merry Little
Christmas