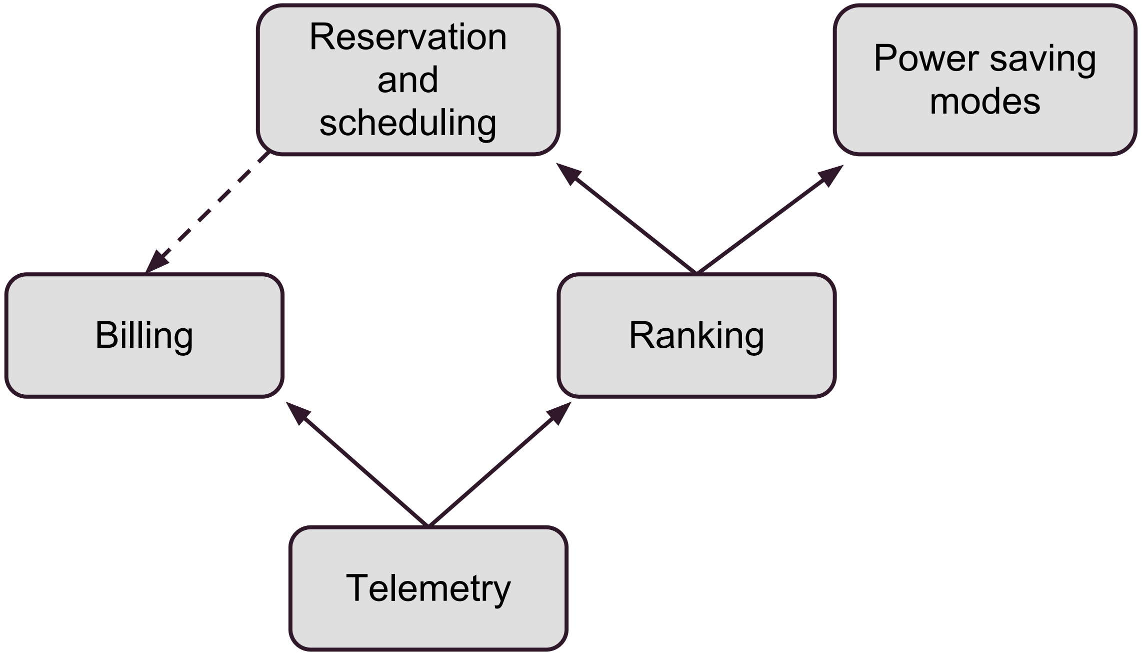
Technical workshop
Energy efficiency

François Rossigneux, Inria Lyon

25 Juin 2013

Functionalities architecture



Reservation and scheduling

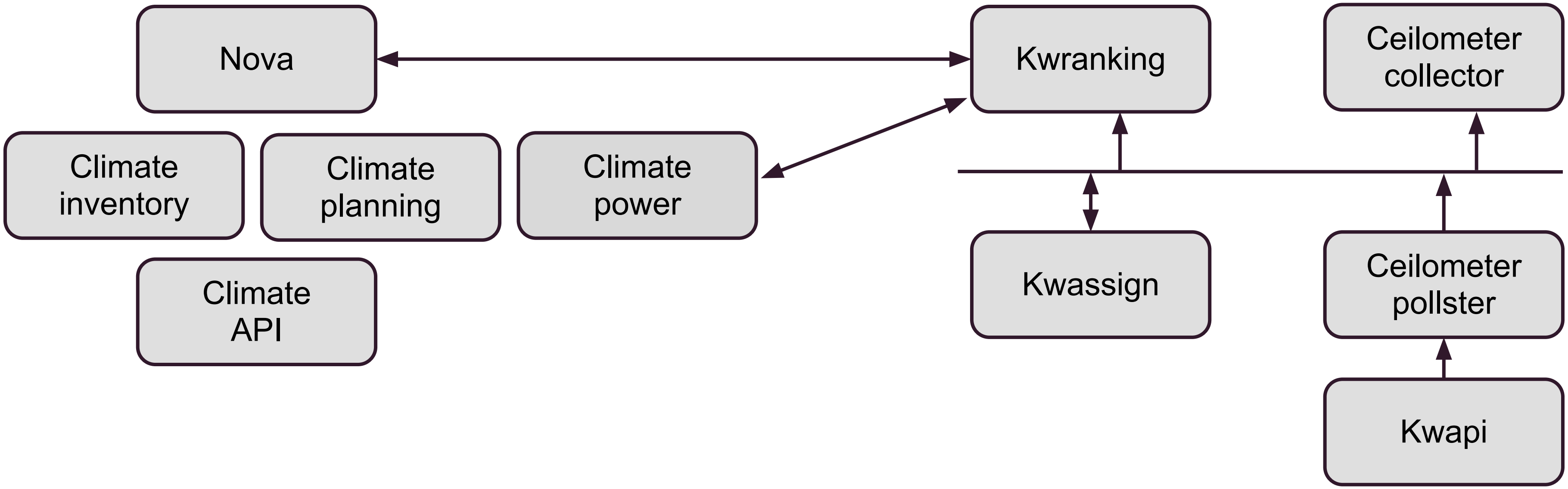
Power saving modes

Billing

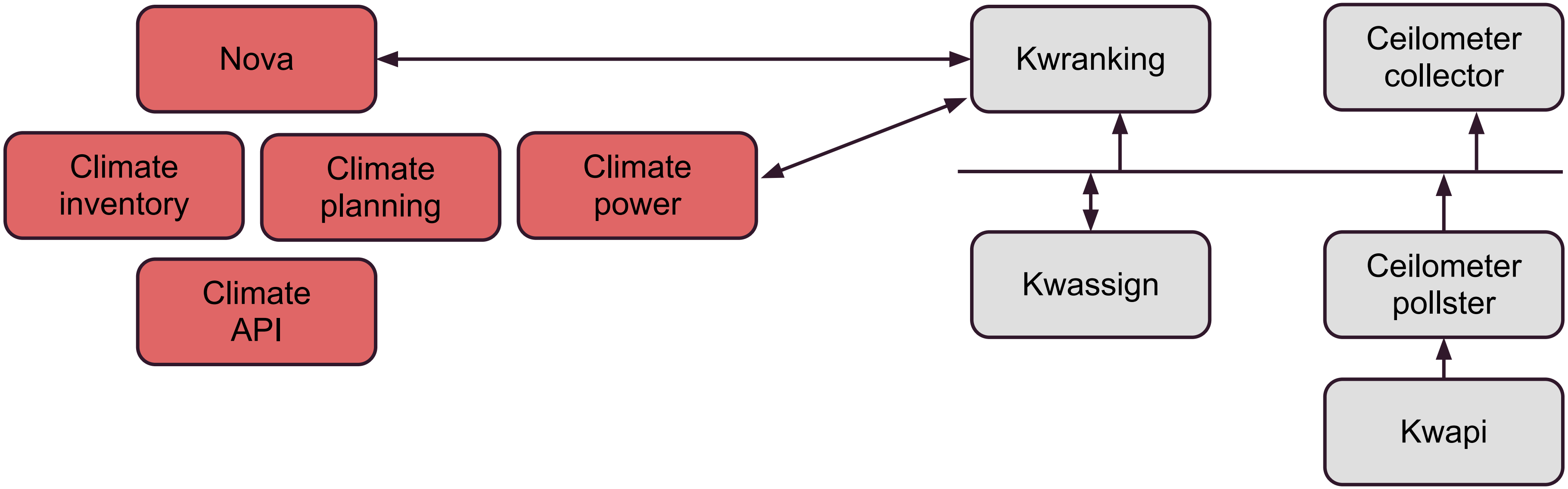
Ranking

Telemetry

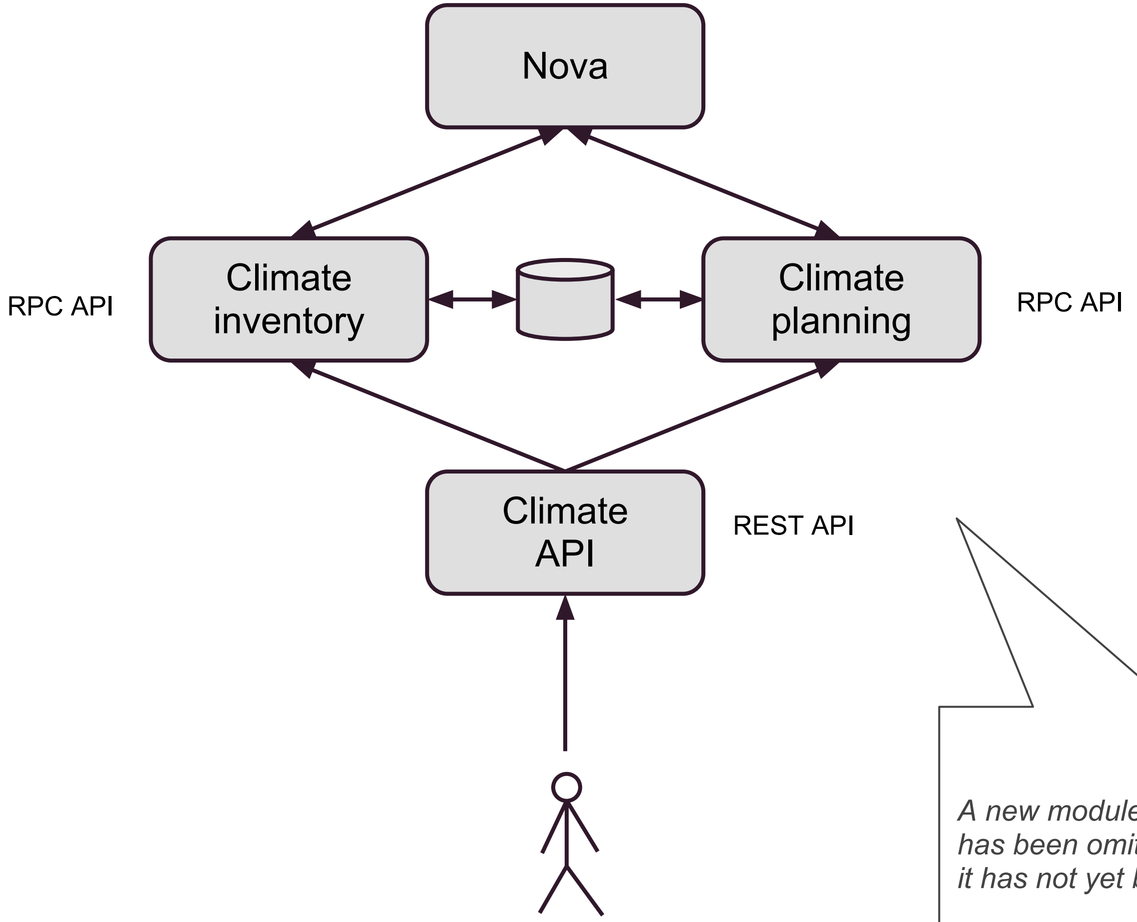
Software architecture



Climate architecture

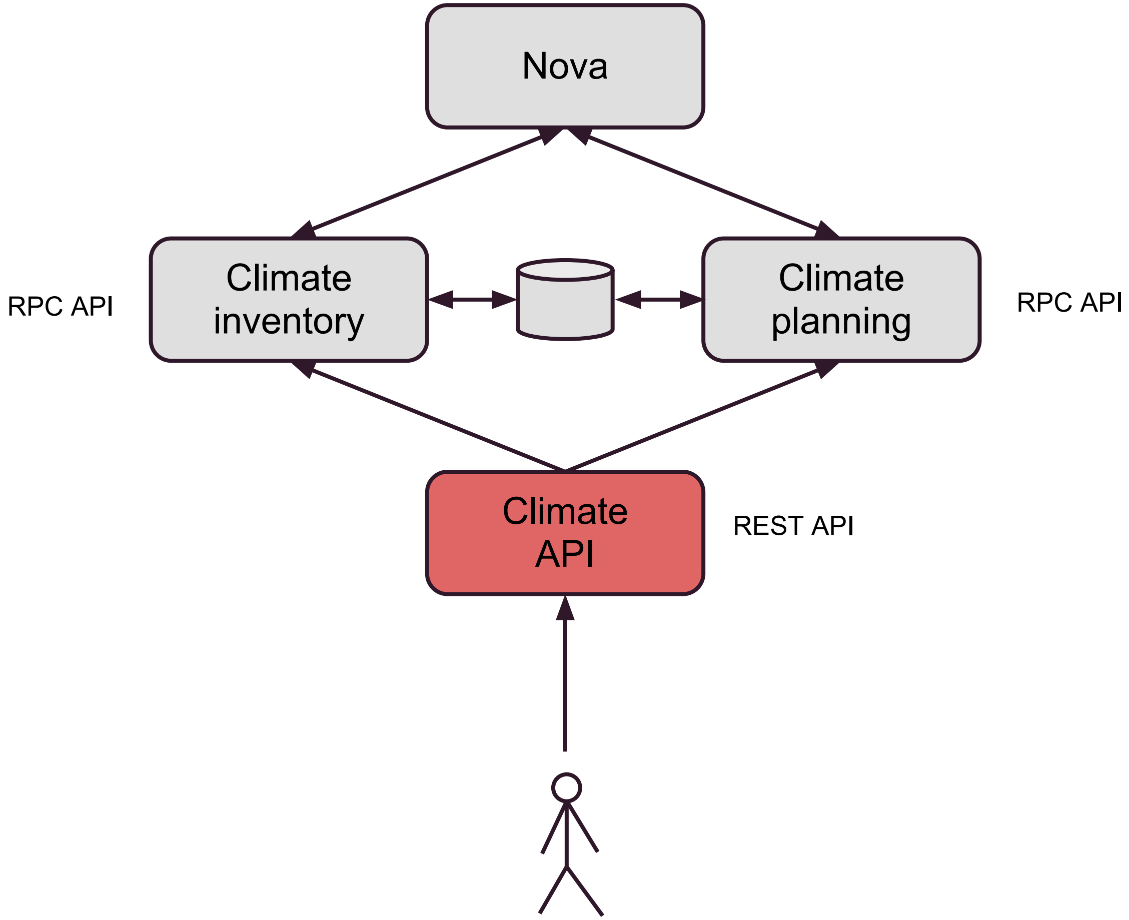


Climate architecture



A new module, called Climate Power, has been omitted on this figure because it has not yet been designed

Climate architecture



Climate API (REST)

Method	URL	Description
POST	/v1/leases/	Creates a lease
GET	/v1/leases/<lease_id>	Shows details
DELETE	/v1/leases/<lease_id>	Deletes a lease

Error codes are returned if something goes wrong...

Climate API (REST)

Creating a lease:

1- The user sends a JSON file to /v1/leases/ (POST request) :

```
{"start_time":10,
  "end_time":100,
  "duration": 5,
  "quantity":1,
  "host_properties": "[\"=\", \"$id\", 1]"
}
```

Working example, but doesn't make sense...
TODO : look how to filter nested properties (like CPU infos)
I use the Nova json_filter module (extracted and slightly modified).

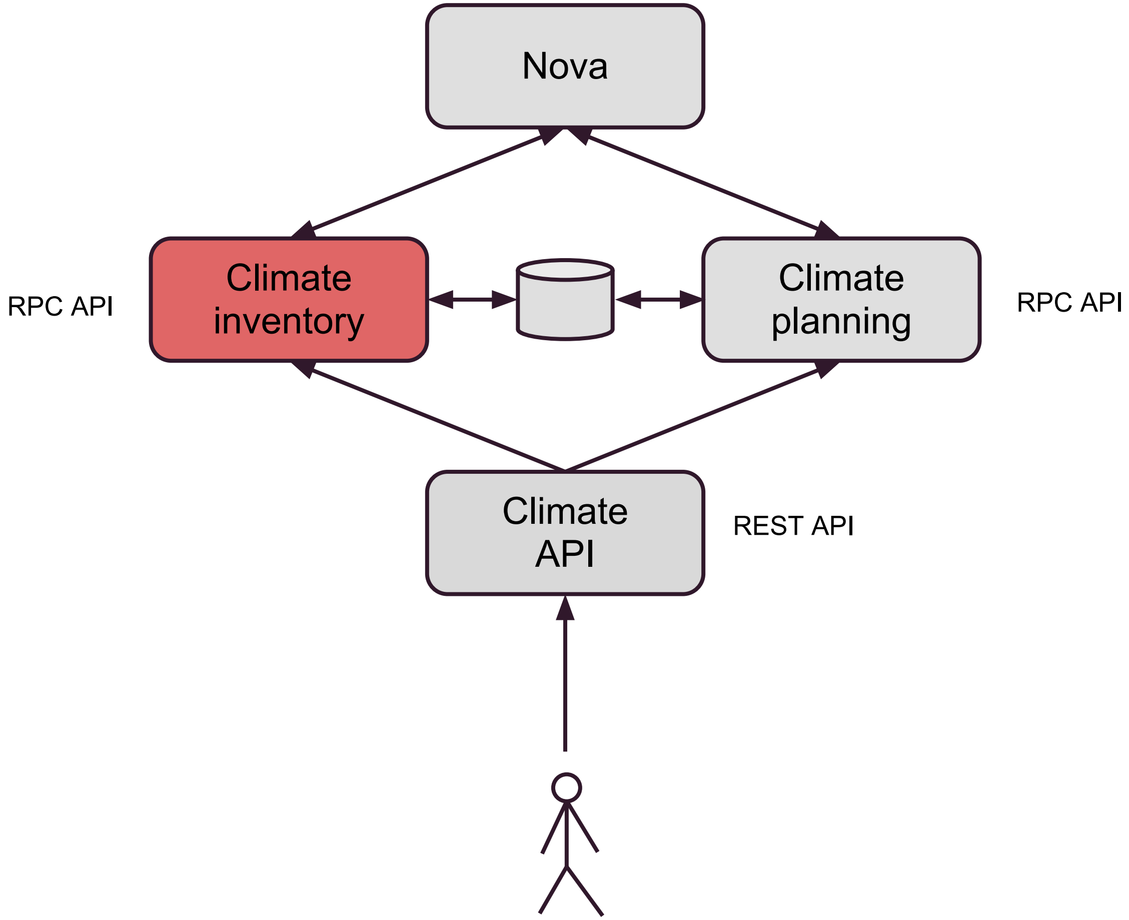
2- Climate API queries Climate Inventory to get the matching hosts

- Matching the requirements
- With running_vms = 0

"Running VM" not yet implemented... will need a locking mechanism.

3- Climate API queries Climate Planning to get the free time periods

Climate architecture



Climate inventory (RPC API)

Components:
- RPC API (callable with OpenStack RPC, using RabbitMQ...)
- A database (role = caching mechanism)
- A worker refreshing the database regularly
- A Python module implementing the inventory functionalities (using Nova Client)

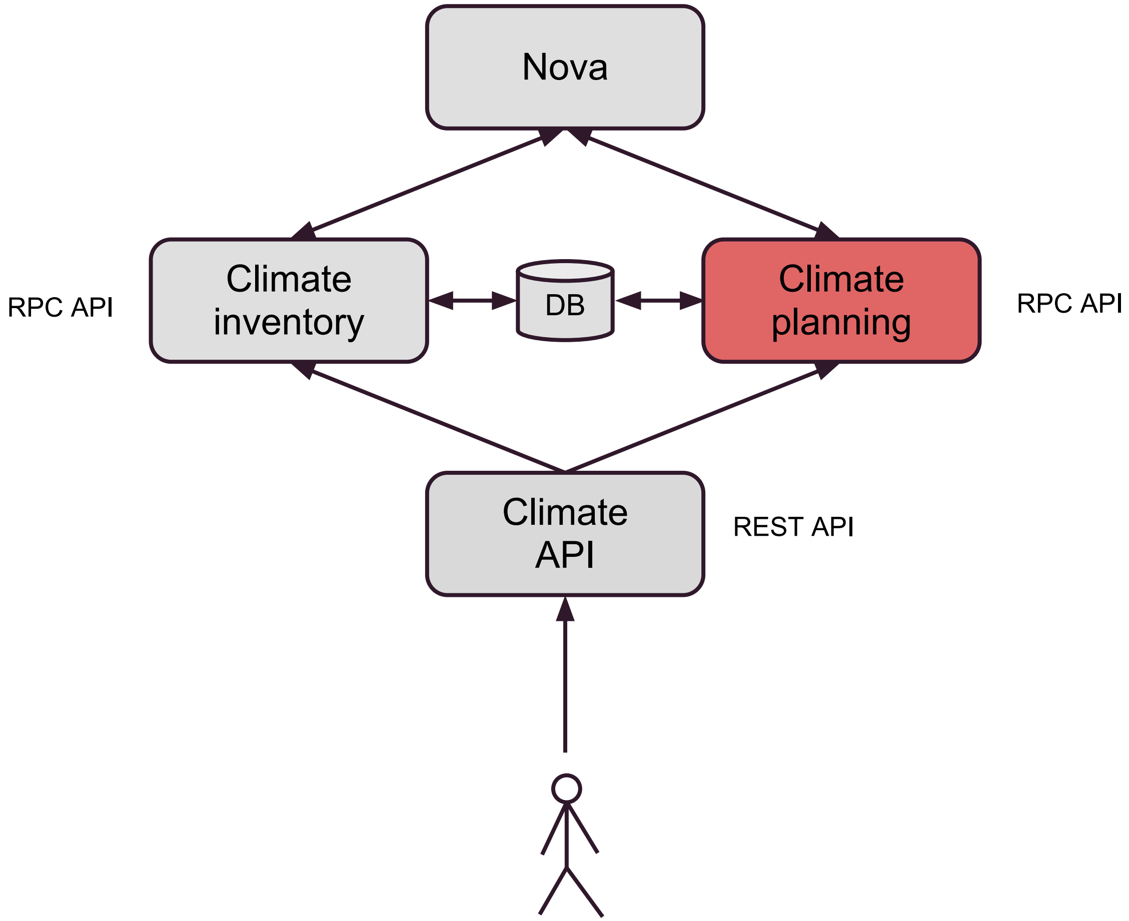
Nova Client:

novaclient.hypervisors.list()	=>	host ids
novaclient.hypervisors.get(host_id).__dict__	=>	details (+ cpu_info)

API:

get_hosts(properties='[]')
get_host_details(host_id)

Climate architecture



Climate planning (RPC API)

Components:

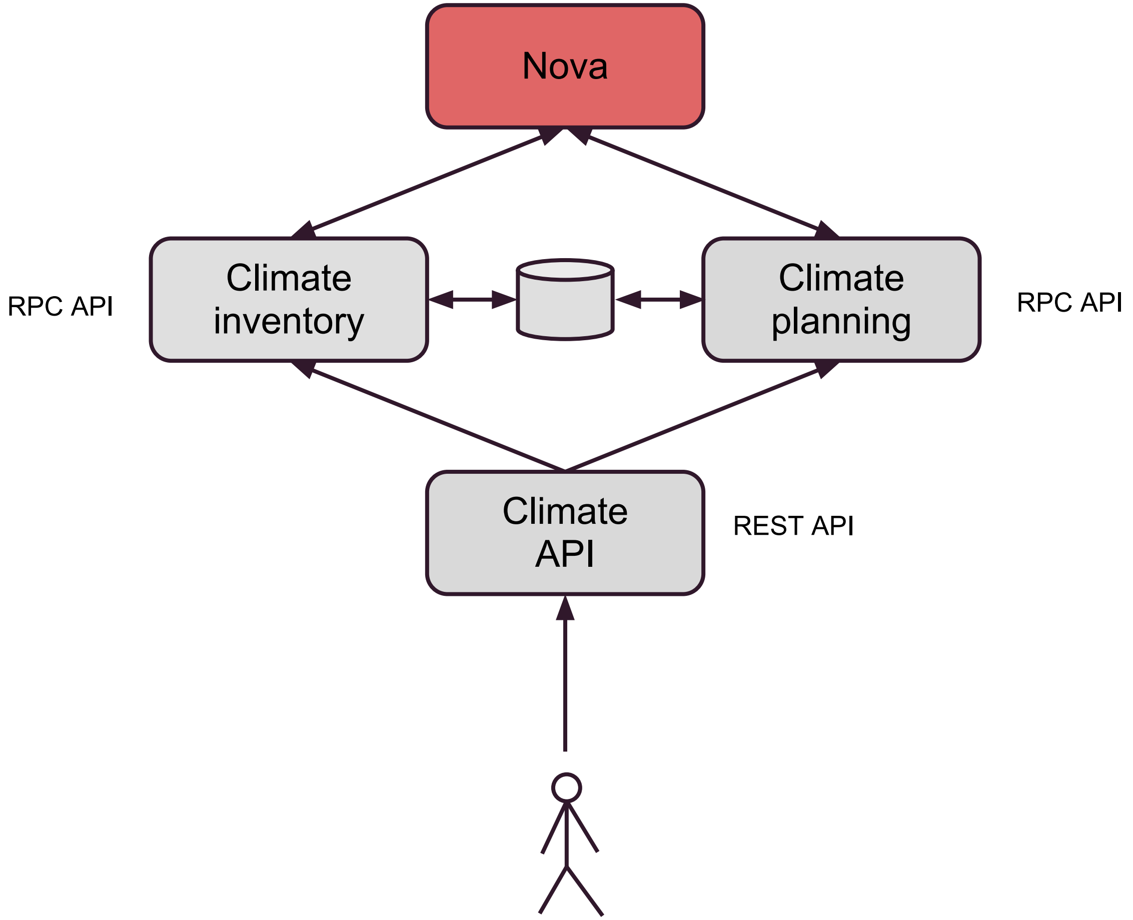
- RPC API (callable with OpenStack RPC, using RabbitMQ...)
- A database for storing reservations
- A worker deleting obsolete reservations
- A Python module implementing the planning functionalities

API:

make_proposal(hosts, start_time, end_time, duration, quantity)
get_proposal_details(lease_id)
confirm_proposal(lease_id)
cancel_proposal(lease_id)

List of hosts, builded by querying Climate Inventory...

Climate architecture



Nova

Running an instance inside a lease:

1- The user provides a scheduler hint

The hint is the lease id.

2- Nova retains only the reserved nodes (filtering)

It retrieves the nodes attached to the reservation by calling Climate API
For each node, it looks whether it is attached to the lease
TODO: is it easy to share the attached nodes array in Nova?

3- Nova finds the best host (weighing)

How? Not very clear today...
Contact the Kwranking module, by passing the list of hosts.
And the Kwranking module return a sorted list?
TODO: today, the Kwranking module doesn't know flop/w units...
Find performance info? Use benchmarks, or deduce it from cpu_info...

Nova

Running an instance outside a lease (multi-tenancy environment):

1- The user doesn't provide a scheduler hint

2- Nova retains only the non-reserved nodes (filtering)

Not clearly defined today...

Solution 1:

- Add a API URL to retrieves all lease ids belonging to a user...
  ... and the admin has the right to get all these lease ids
- It calls the API for each lease id, to obtain the attached host ids

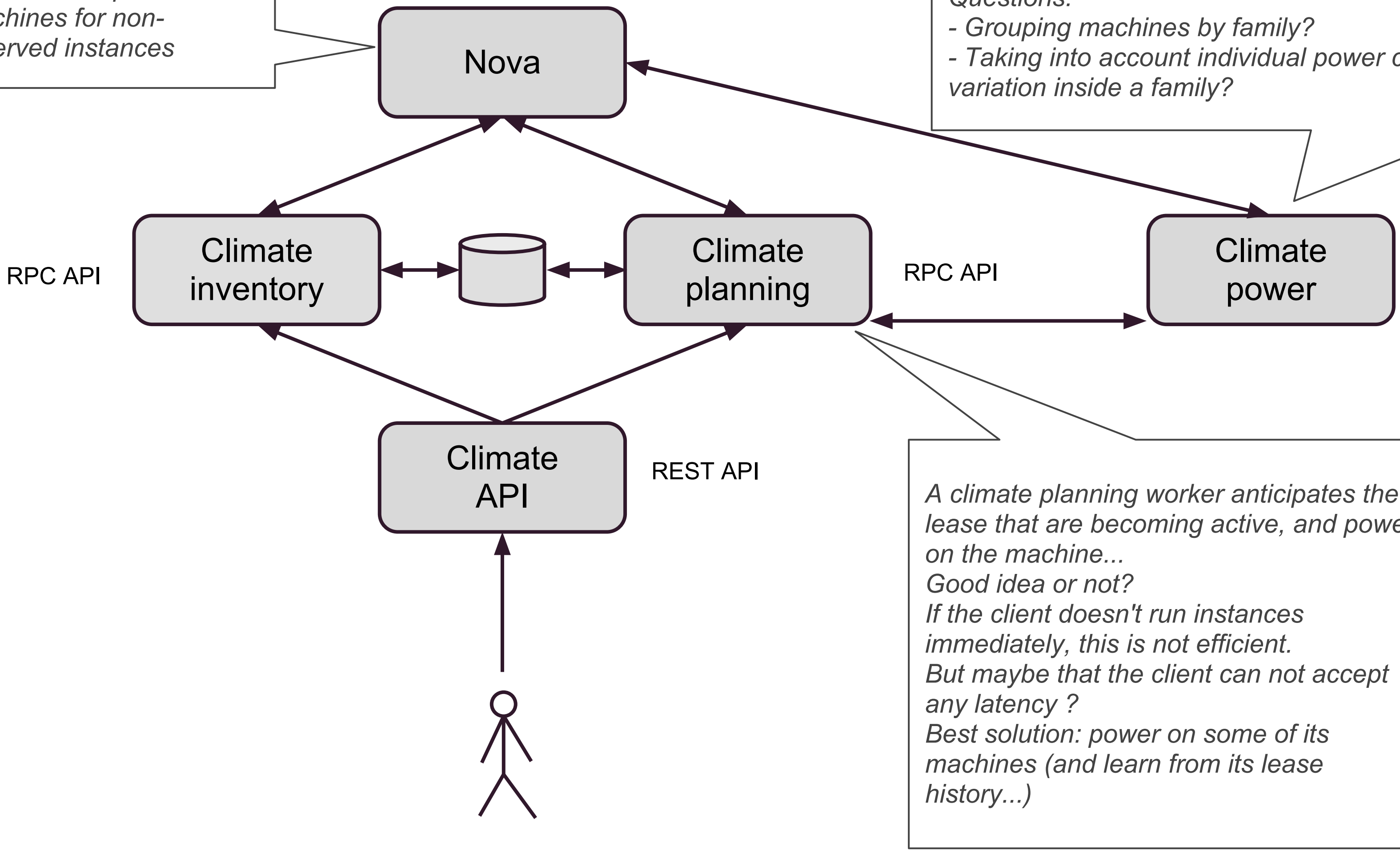
Solution 2:

- Add a method to get all available hosts (specify the quantity in parameter?)

The solution 1 proposes a useful method (add it to the TODO list anyway?)...
... but it is iterative and slow. So the solution 2 seems better.

3- Finds the best host (weighing)

Climate power (REST API)



Nova asks to power on machines for non-reserved instances

Looks into the reservation DB
Shuts down non-reserved and running machines
Queries Kwranking to know the greenest hosts

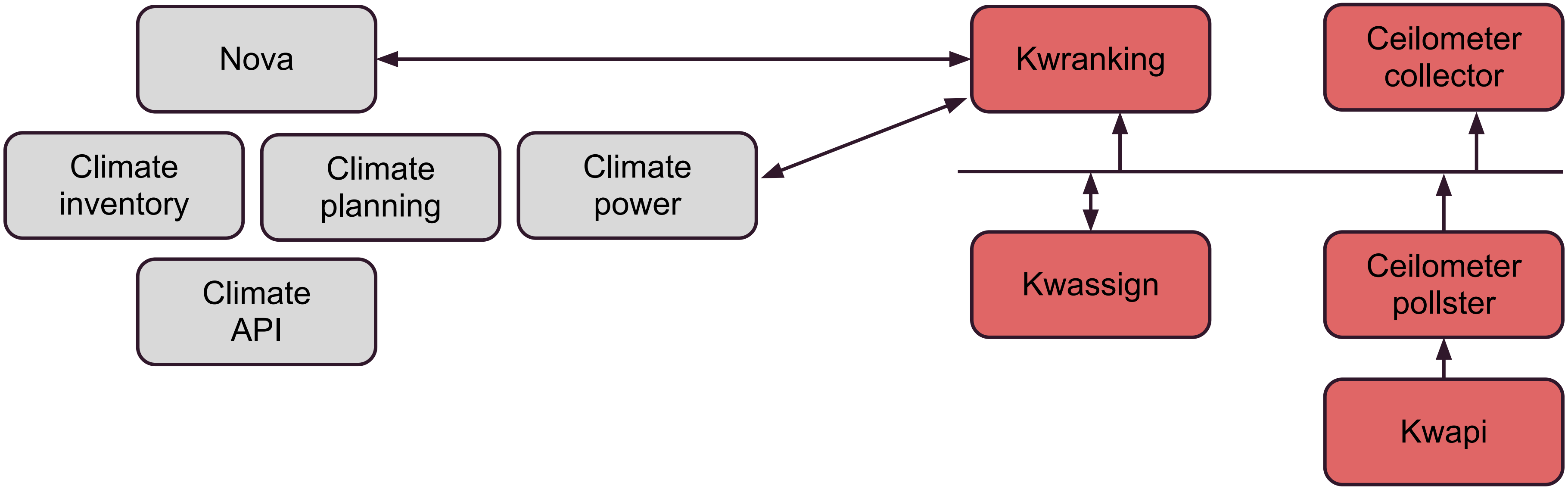
Questions:
- Grouping machines by family?
- Taking into account individual power consumption variation inside a family?

A climate planning worker anticipates the lease that are becoming active, and power on the machine...
Good idea or not?
If the client doesn't run instances immediately, this is not efficient.
But maybe that the client can not accept any latency ?
Best solution: power on some of its machines (and learn from its lease history...)

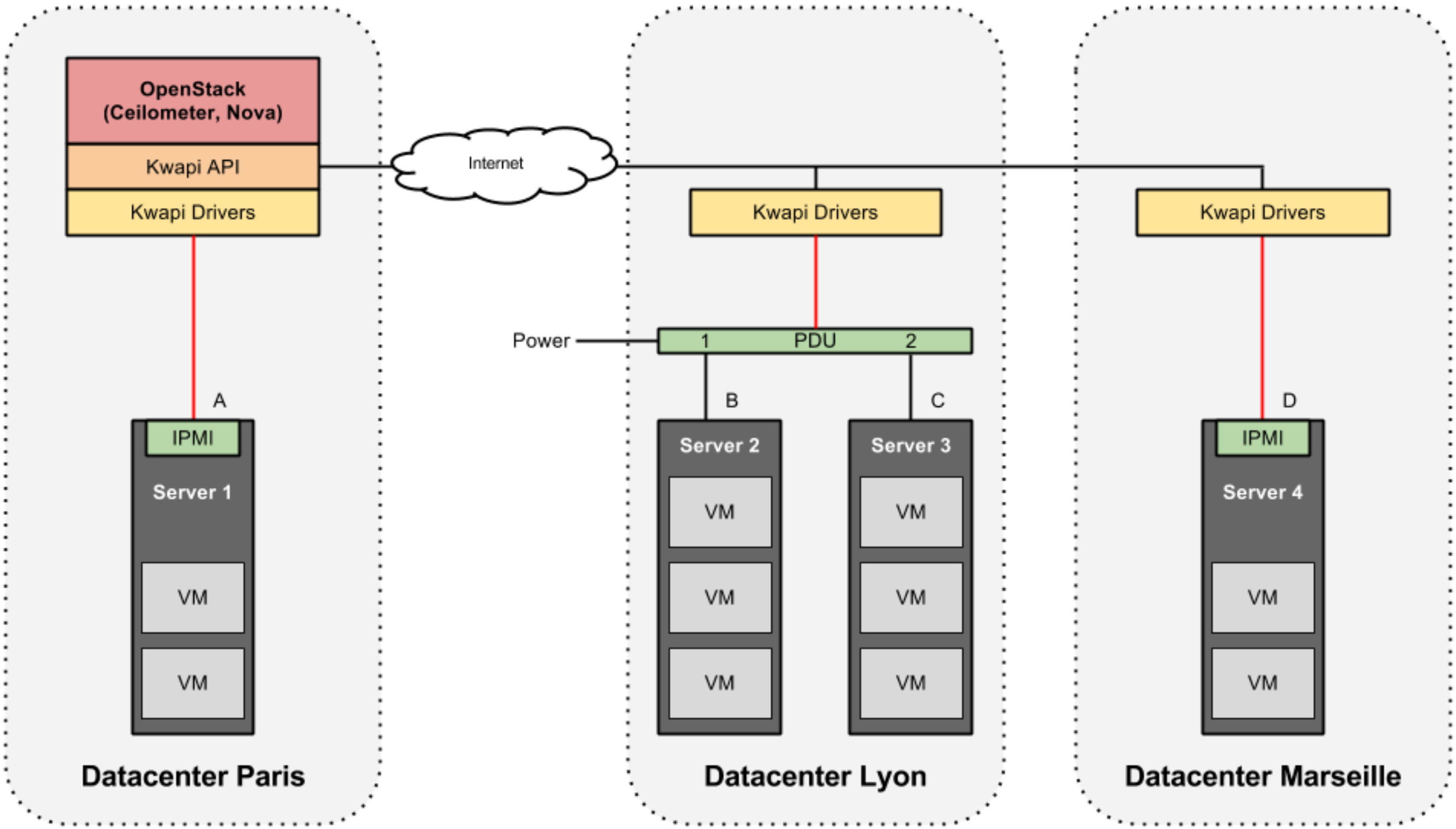
Climate and Nova roadmap

Component	Progression	TODO
Climate API	90%	Manage lease owner Use xxx instead of Flask (OpenStack best practices?) Test security (code ok but not tested...)
Climate Inventory	95%	"running_vm" filter
Climate Planning	99%	Optionally, use SQL? Worker for calling Climate power
Climate Power	0%	All
Nova filter	0%	All
Nova weighing	0%	All

Kwapi and modules around the Ceilometer bus



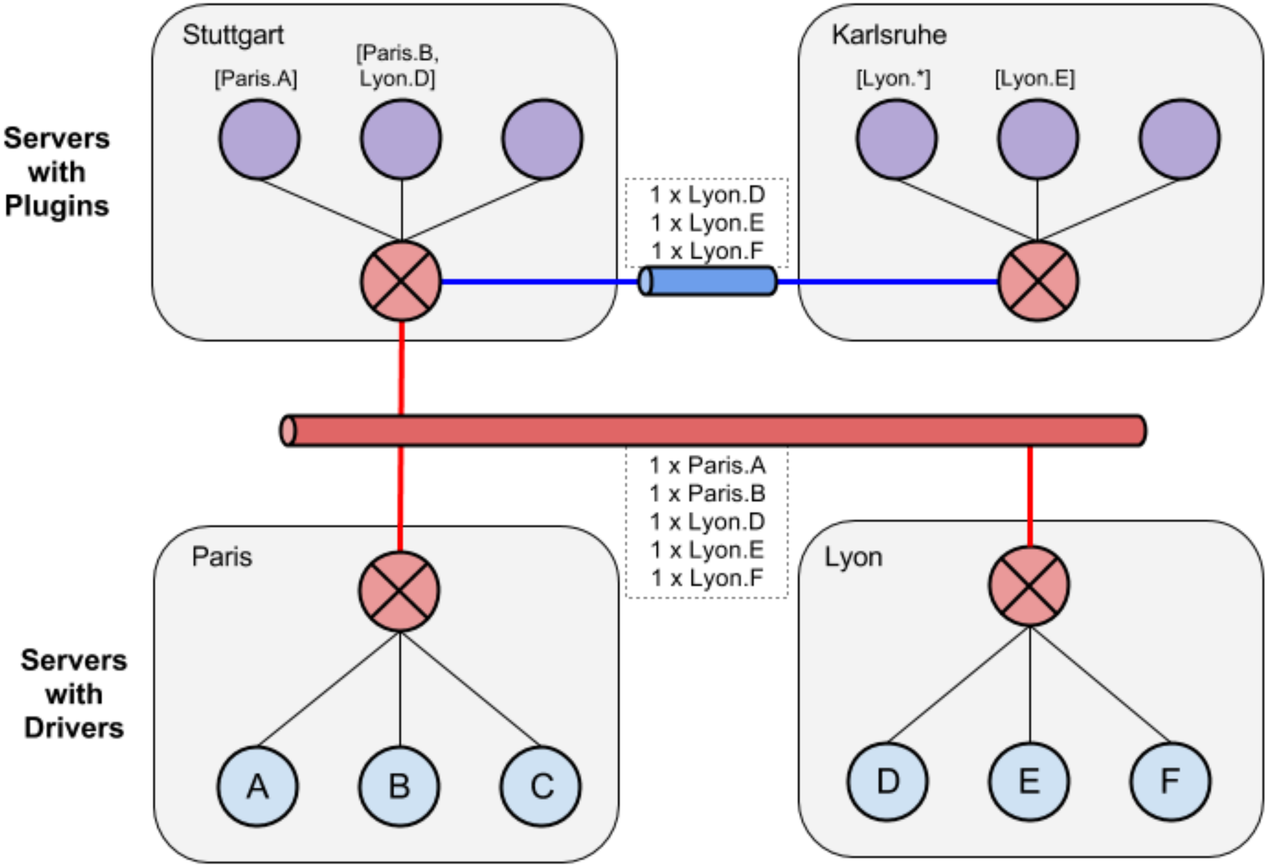
Kwapi architecture



Kwapi recent improvements

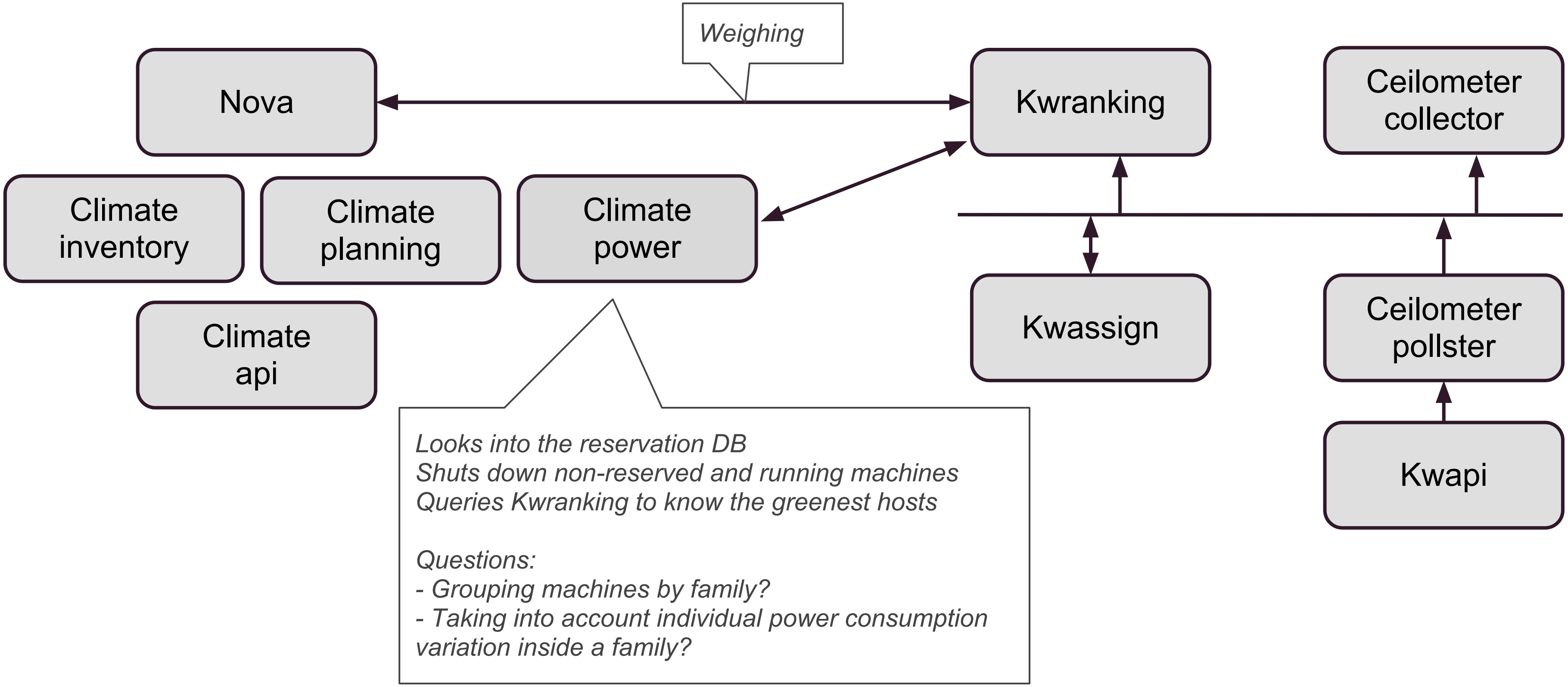
- Writing a driver to support Eaton PDU 24-outlets (SNMP queries)

- Optimizing the network usage (probe subscriptions, forwarding devices) :



Kwapi: exploiting the measurements

- Assigning a tenant to the power consumption metrics (TODO)
- Ranking the machines (need to compute flop/w)
- Creating the Climate power module



Weighing

Looks into the reservation DB
Shuts down non-reserved and running machines
Queries Kwranking to know the greenest hosts

Questions:
- Grouping machines by family?
- Taking into account individual power consumption variation inside a family?

Conclusion and roadmap

1 - Finalising Climate (except the power module)

2 - Writing Nova filters and weighers

3 - Solving the blocking points:
- Assigning the metrics to a tenant (so Kwassign will be completed)
- Finding or estimating the machines performances, to build flop/w metrics
- Finalising the Kwranking design

4 - Writing the Climate power module

Thank you!
Any questions?