Energy-aware VM Allocation on An Opportunistic Cloud Infrastructure

Harold Castro, Mario Villamizar
Department of Systems and Computing Engineering
Universidad de los Andes
Colombia

Cesar Díaz, Johnatan Pecero, Pascal Bouvry
Computer Science and Communications Research Unit
University of Luxembourg
Luxembourg

THE PROBLEM

ENERGY IN AN OPPORTUNISTIC CLOUD ENVIRONMENT

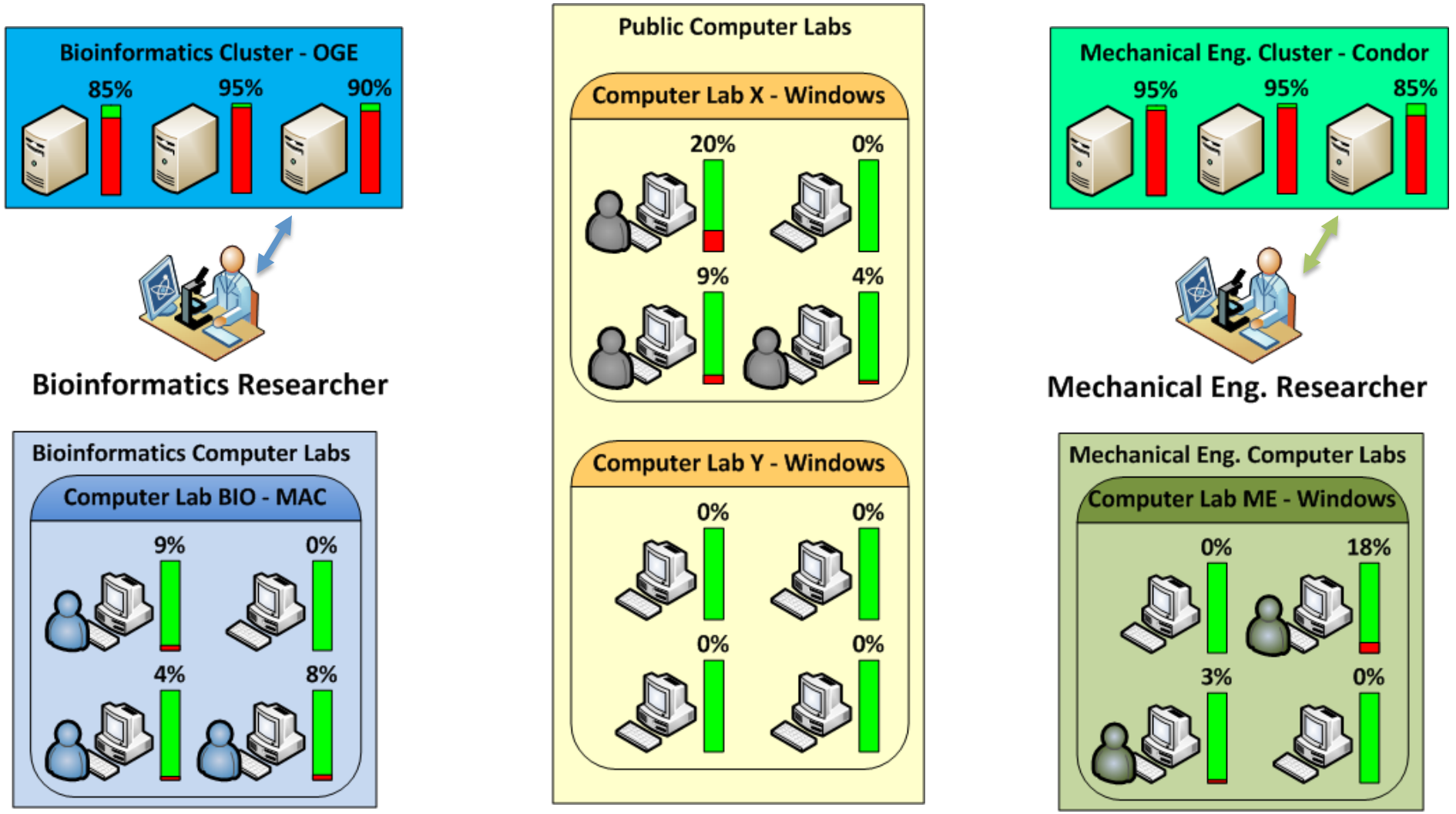
ENERGY-AWARE TECHNIQUES

ALGORITHM TO CALCULATE THE ECR

EXPERIMENTAL RESULTS

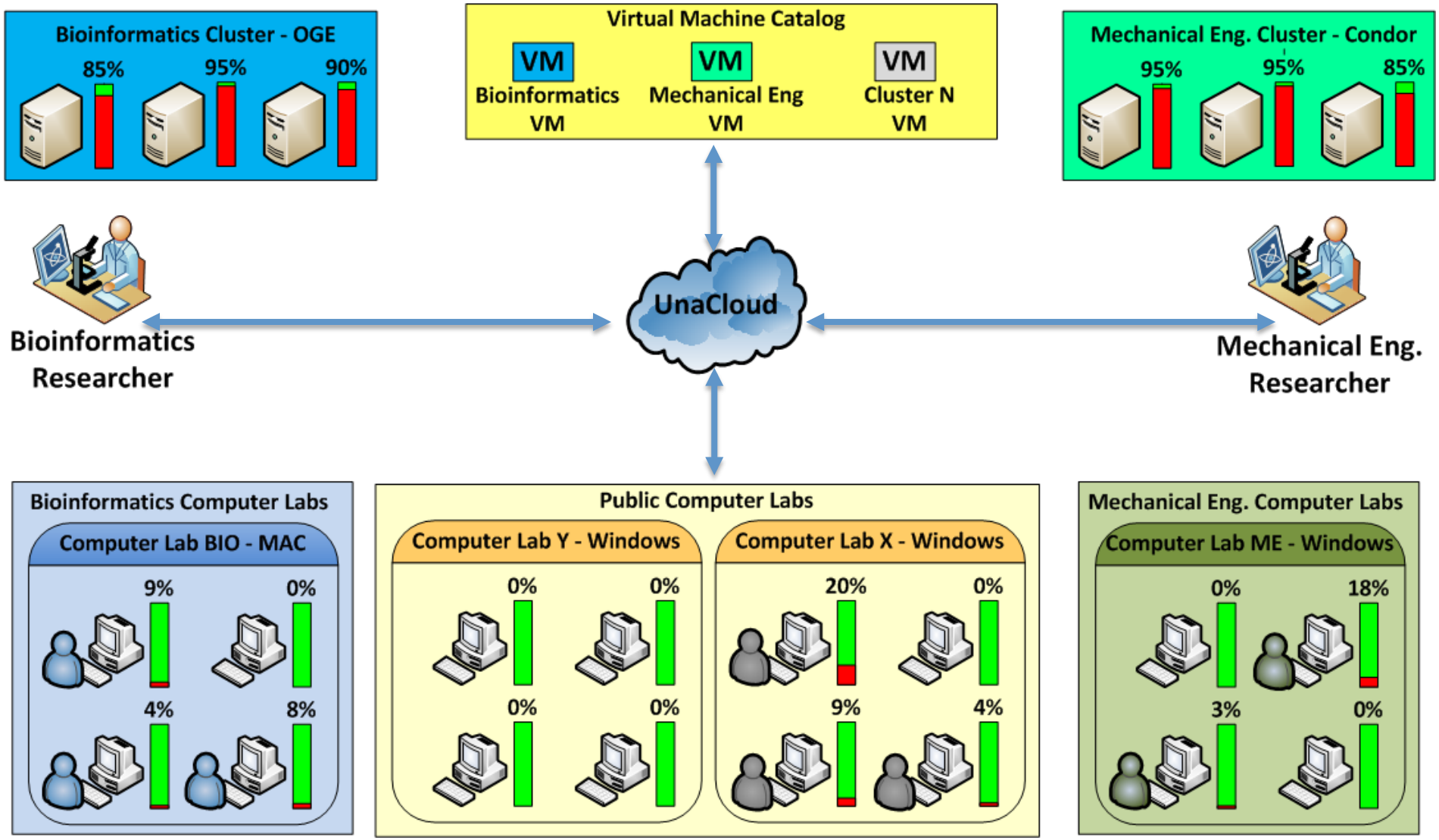
CONCLUSIONS AND FUTURE WORK

THE PROBLEM

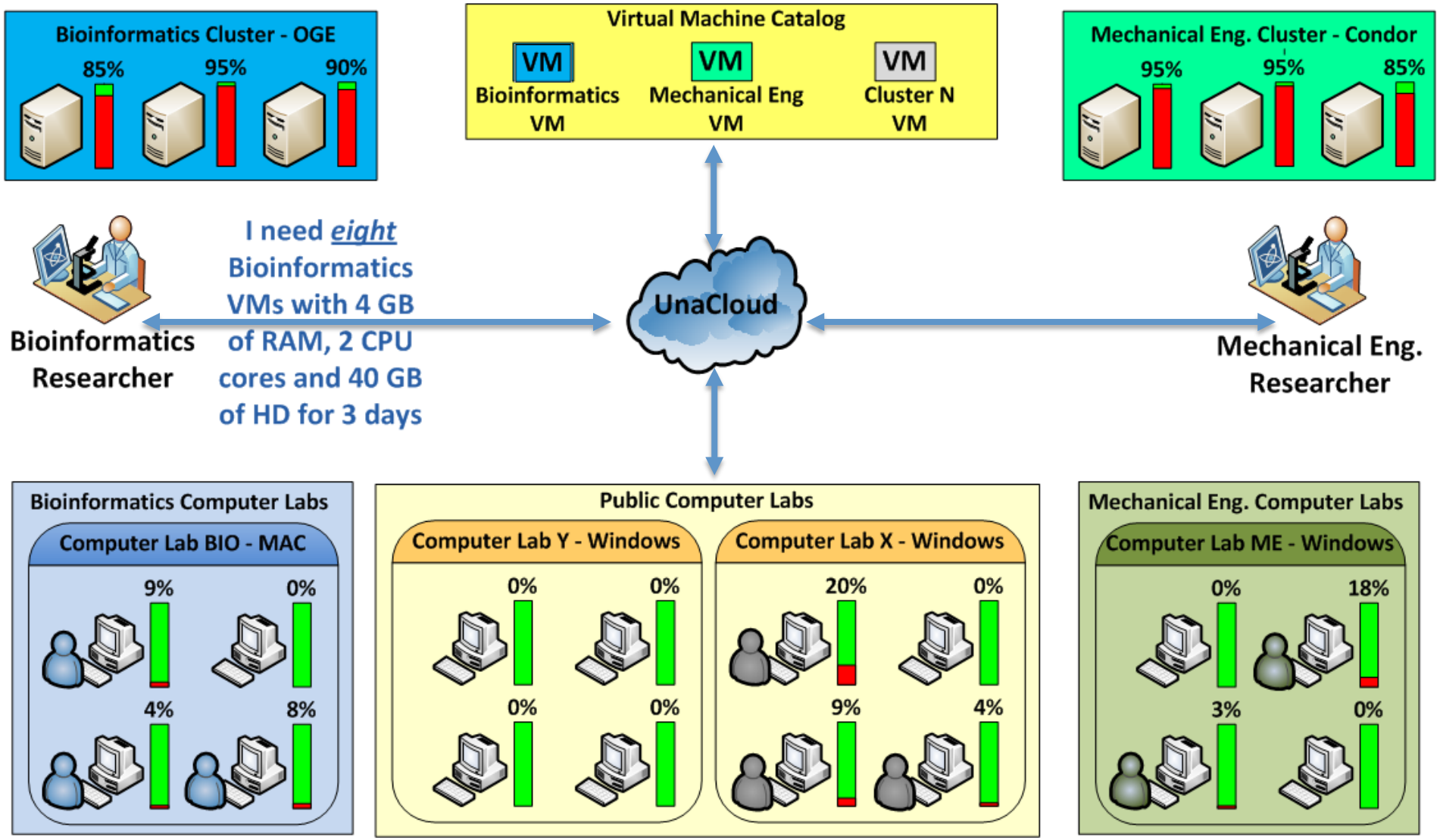


More than 2000 CPU cores

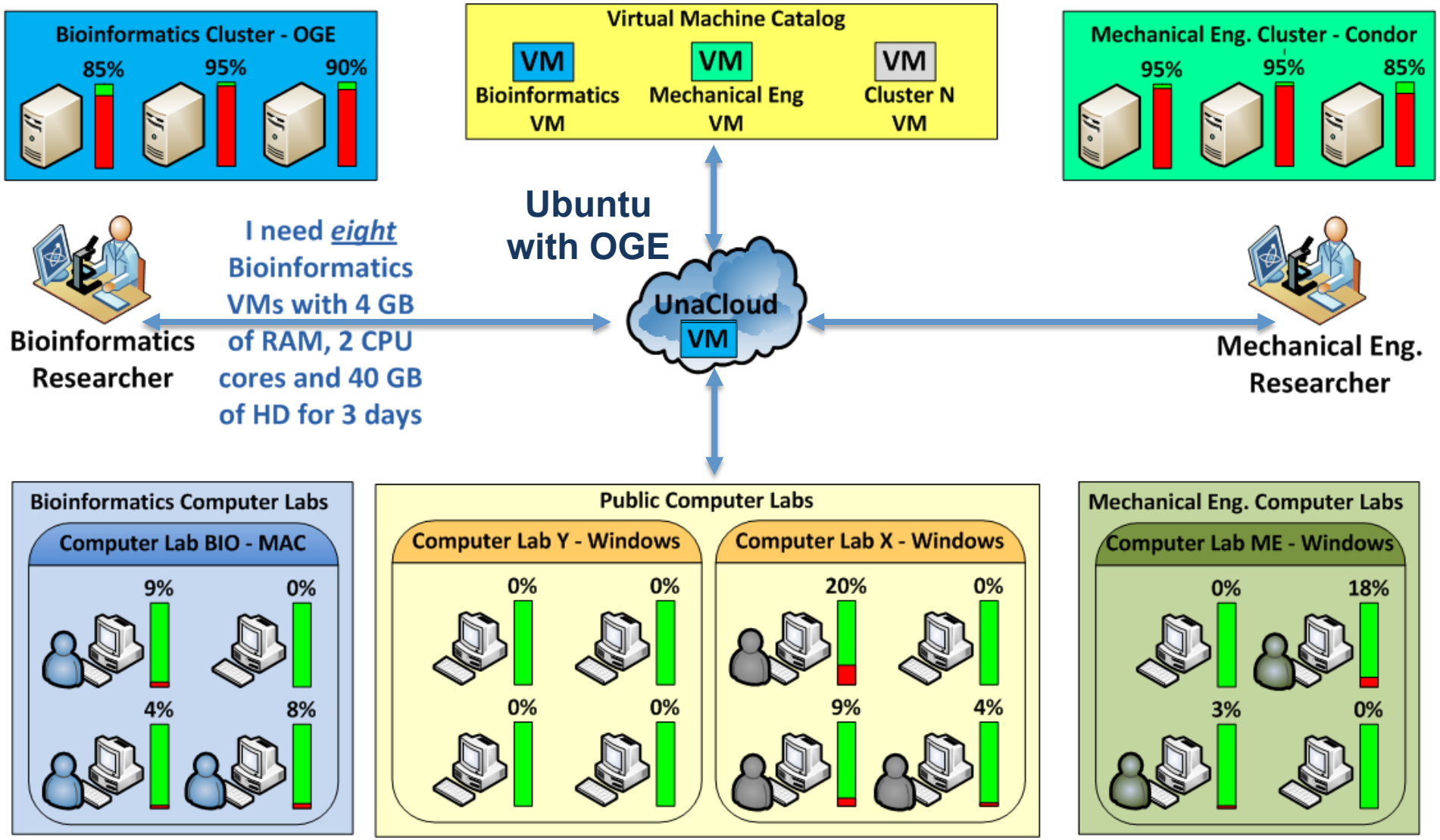
HOW UNACLOUD WORKS



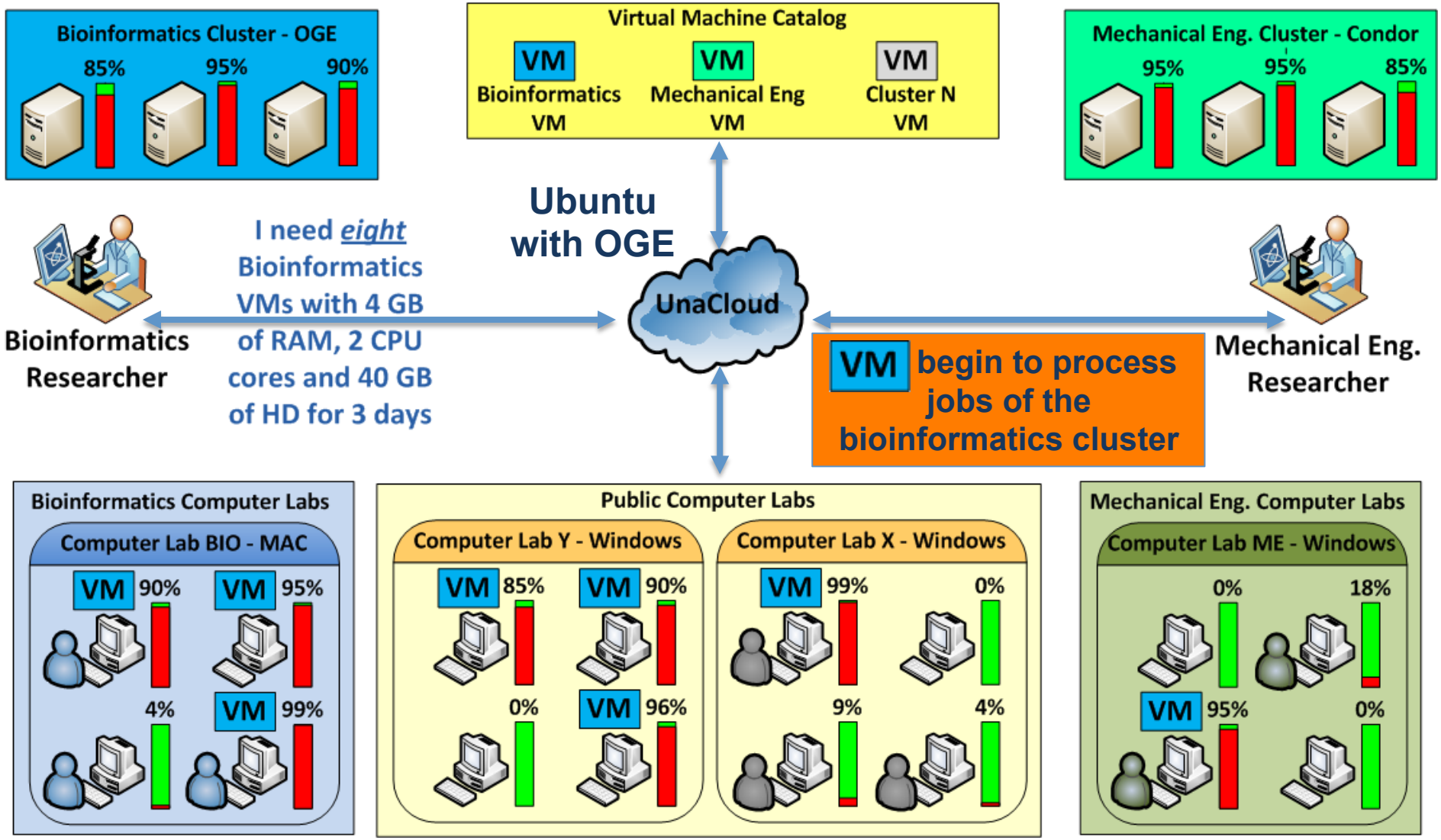
HOW UNACLOUD WORKS



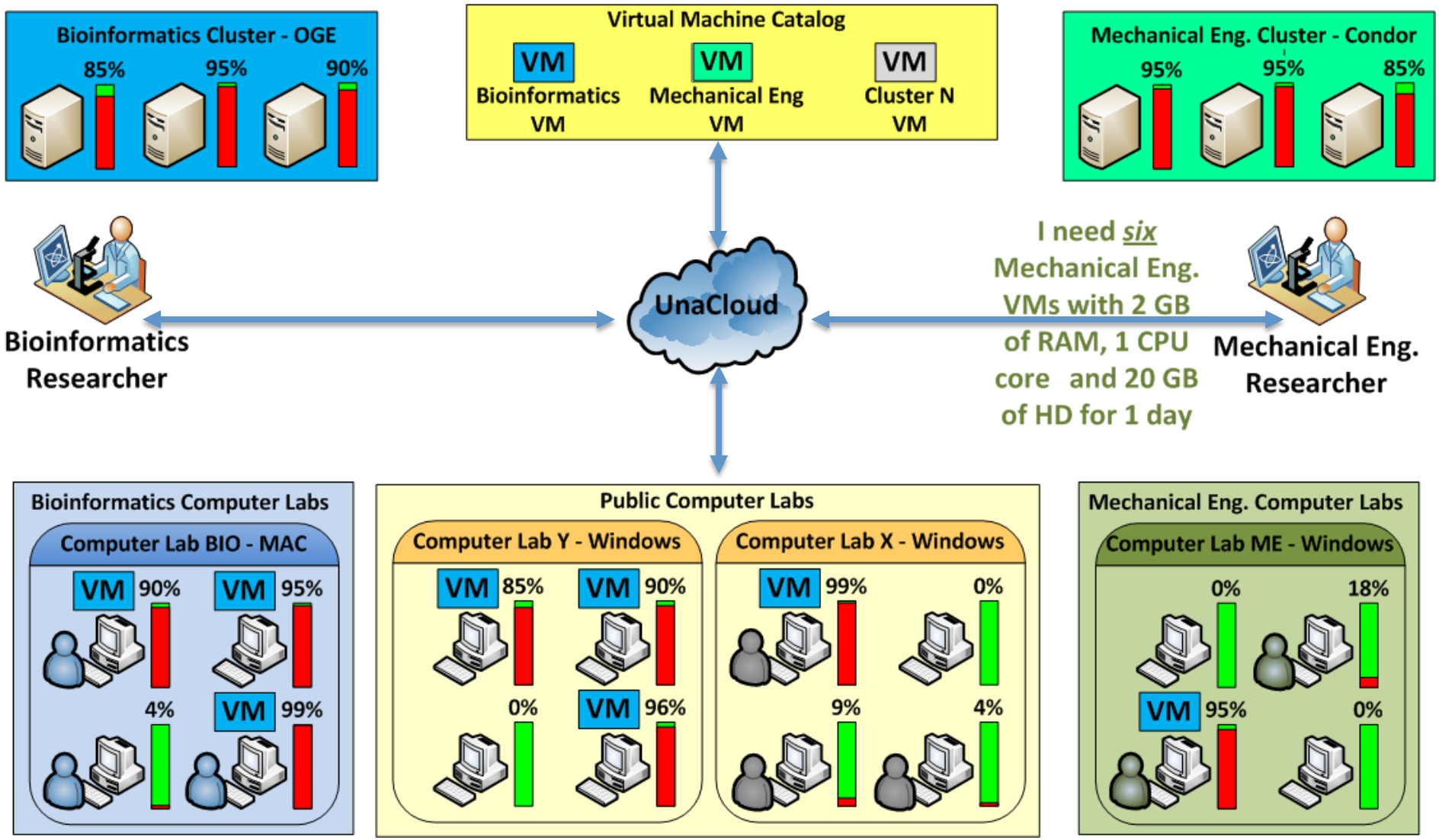
HOW UNACLOUD WORKS



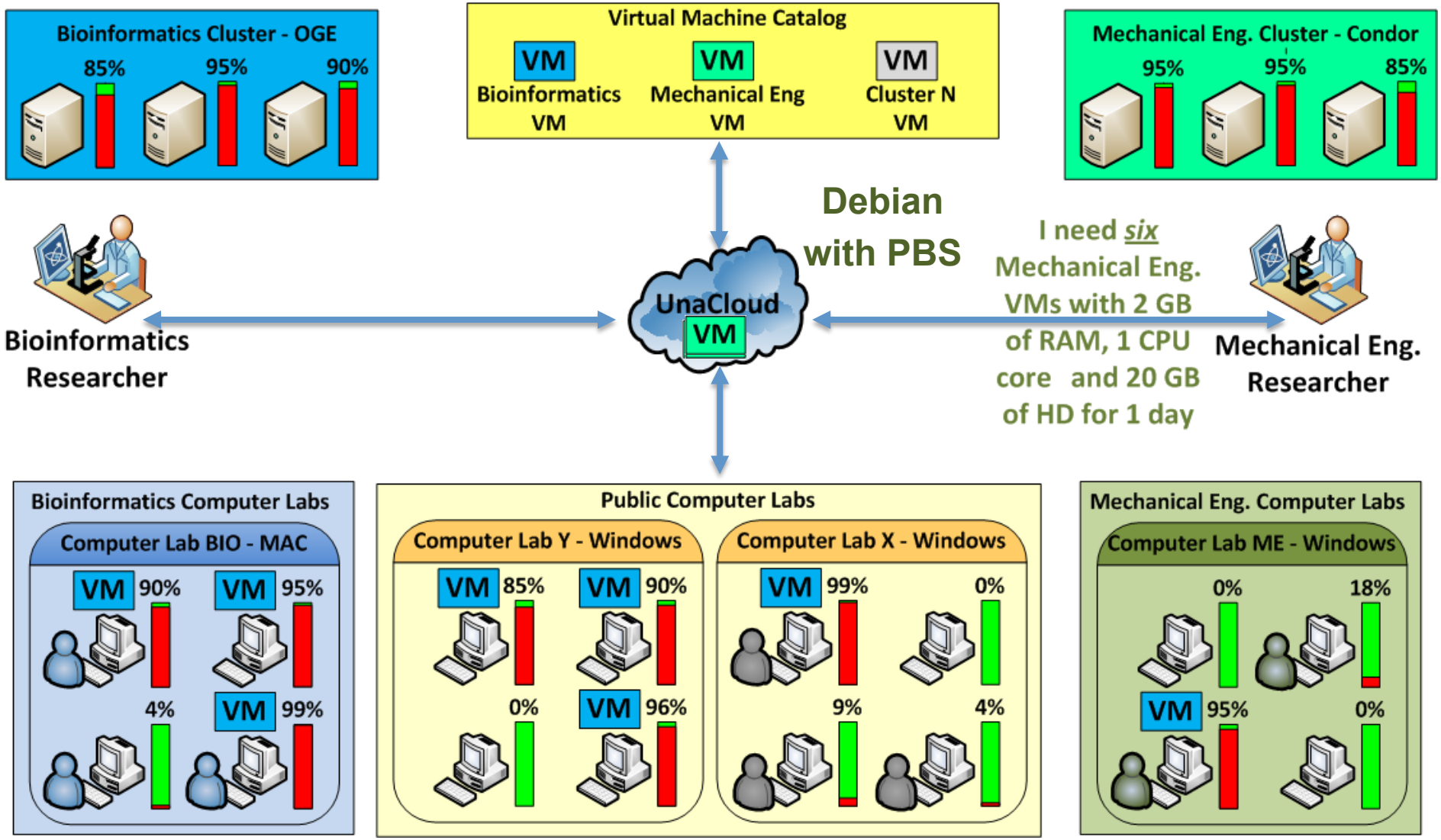
HOW UNACLOUD WORKS



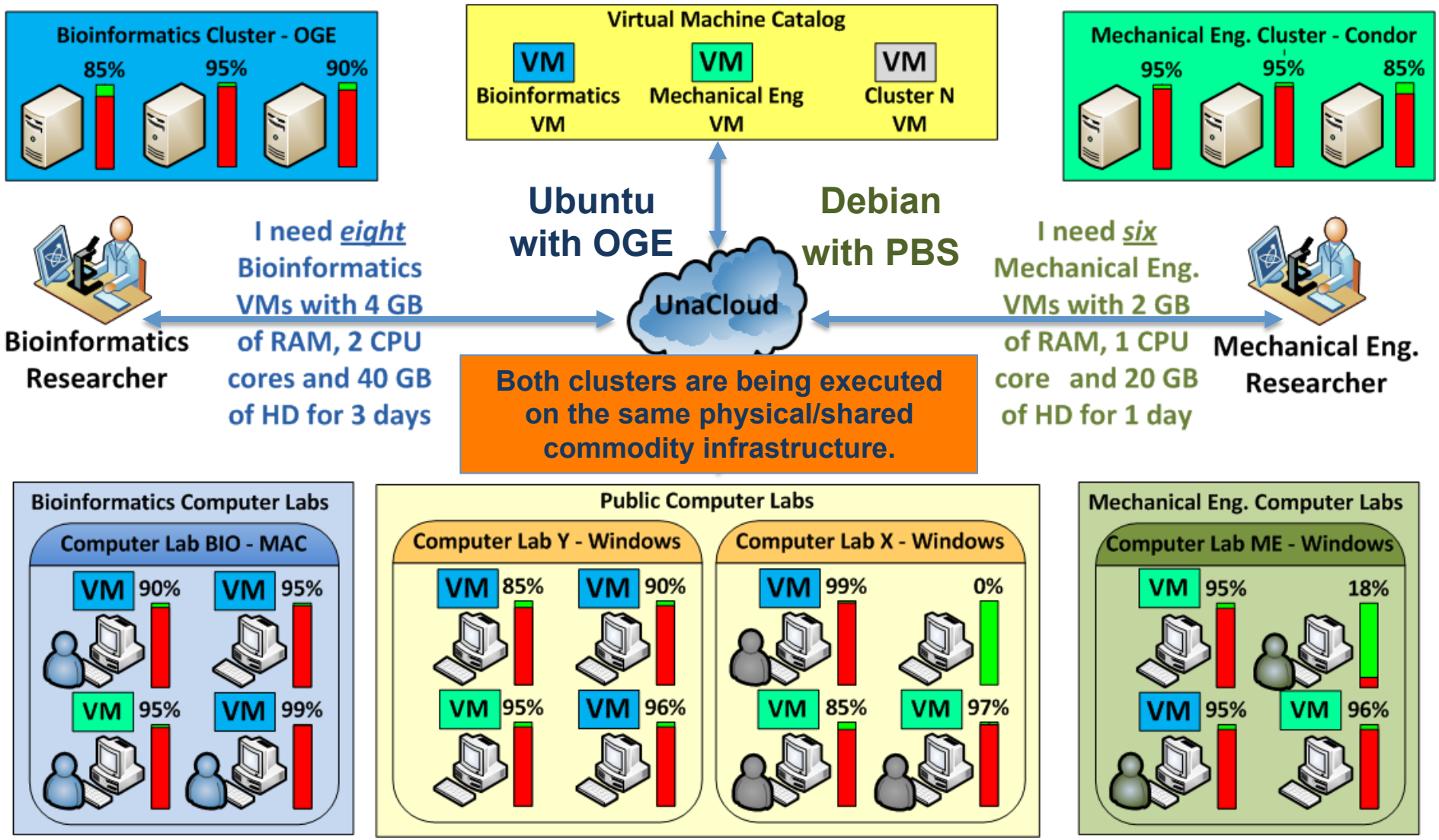
HOW UNACLOUD WORKS



HOW UNACLOUD WORKS



HOW UNACLOUD WORKS



HOW UNACLOUD WORKS

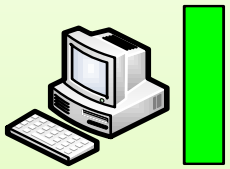
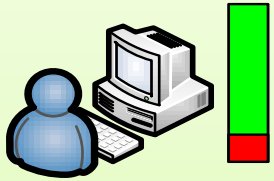
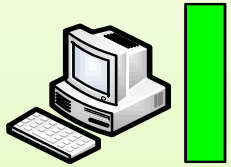
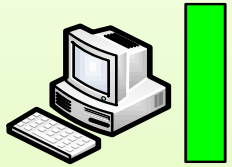
THE PROBLEM

Set of Virtual Machines (SVM)

2 CPU Cores	2 CPU Cores	2 CPU Cores	2 CPU Cores
VM1	VM2	VM3	VM4

How to assign the Set of Virtual Machines (SVM) to the Set of Physical Machines (SPM) trying to reduce the Energy Consumption?

Set of Physical Machines (SPM)

1. IDLE	2. BUSY	3. HIBERNATED	4. TURNED OFF
0%	10%	0%	0%
			
8 CPU Cores	8 CPU Cores	8 CPU Cores	8 CPU Cores

ENERGY IN AN OPPORTUNISTIC CLOUD ENVIRONMENT

Table I: ECR used by a VM executing a CPU-intensive task.

Computer state	Execution with VM	ECR for an intensive CPU Task (ECR with VM - without VM)
1 (idle)	T	$f(100) - f(0)$
2 (busy)	$\frac{100}{L_{free}} T$	$f(100) + ECR_{MON} - ECR_{user}$
3 (hibernation)	T	$f(100) - ECR_{hib}$
4 (turned off)	T	$f(100)$

T = Execution Time of a Task

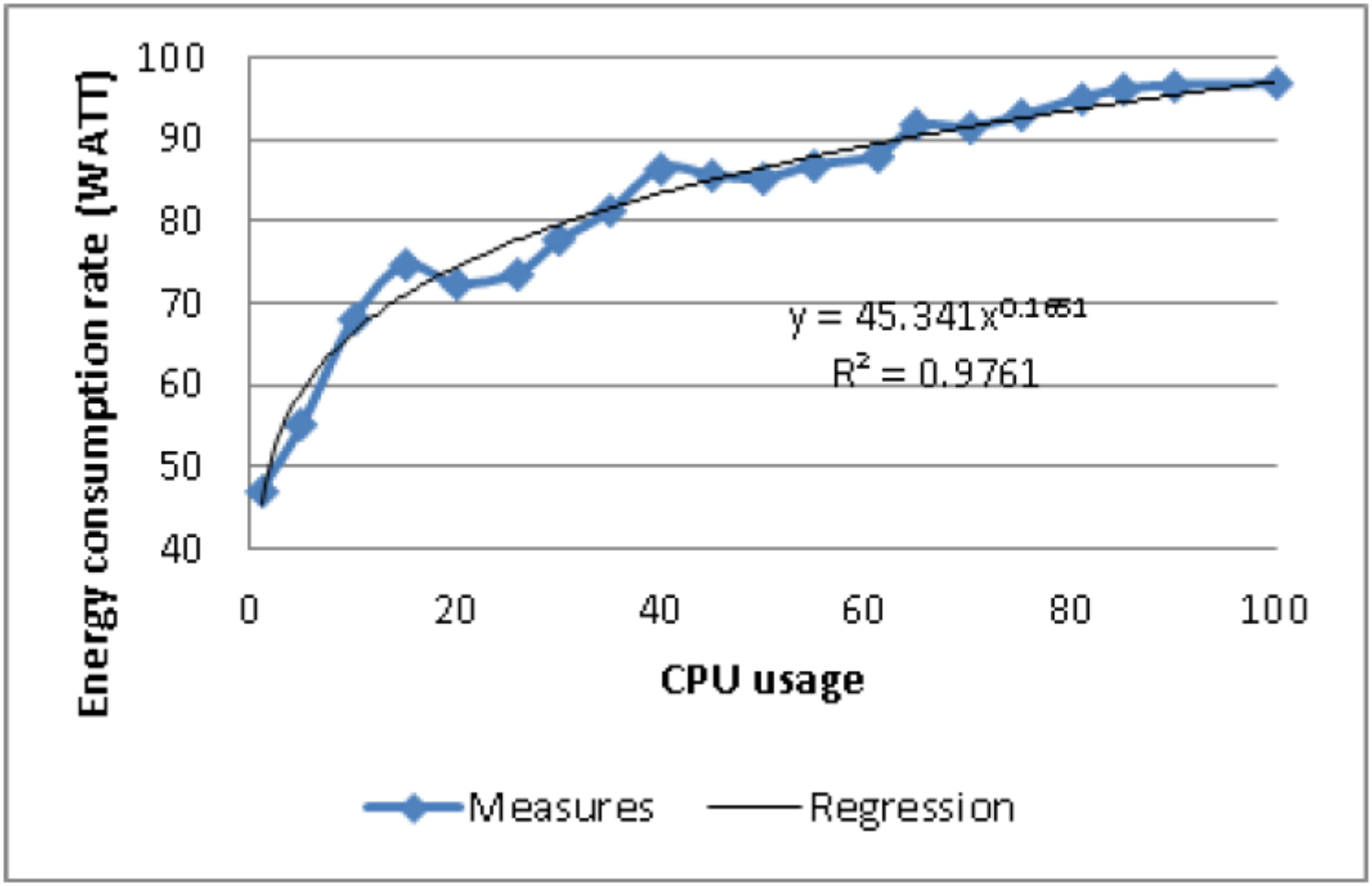
f(x) = Function that return the ECR of a PM given its CPU used

ECR_{mon} = ECR consumed by the monitor of the PM

ECR_{user} = ECR consumed by the user using a PM

ECR_{hib} = ECR consumed by the PM whet it is hibernated

ENERGY IN AN OPPORTUNISTIC CLOUD ENVIRONMENT



f(x) = Function that return the ECR of a PM given its CPU used

ENERGY IN AN OPPORTUNISTIC CLOUD ENVIRONMENT

Table II: Experimental results of energy consumption used by an ***UnaCloud*** intensive-computing task

Computer state	Execution time	Mean ECR (W)	Energy consumption
1 (idle)	T	$49W$ $(96-47)$	$49W \times T$
2 (busy)	$\frac{10}{9}T$	$29W$ $(96+20-87)$	$32W \times T$
3 (hibernation)	T	$93W$ $(96-3)$	$93W \times T$
4 (turned off)	T	$96W$	$96W \times T$

The best desktop machines to deploy a VM are those being used (state 2), followed by the machines that are on idle (state 1), hibernation (state 3) and turned off (state 4) state.

ENERGY-AWARE TECHNIQUES

REDUCE ECR

A. Minimize the ECR using a minimum set of Physical Machines preferably in busy state; increasing the Execution Time of the metatask (SVM) and executing several VMs on a PM.

RESULTS FASTER

B. Assign to each PM (preferably in idle, hibernated or turned off state) only one VM, avoiding the penalty caused by the execution of multiple VMs on the same PM, and increasing the ECR due to more PMs are in execution.

ENERGY-AWARE TECHNIQUES

We propose three resource allocation algorithms that consider the following assumptions and goals:

- ✓ **Reduce the ECR required to execute a metatask (SVMs).**
- ✓ **Several VMs can be executed on a PM.**
- ✓ **The assignation of VMs to PMs is made in batch mode.**
- ✓ **There are enough CPU cores to supports the CPU requirements of the SVM.**
- ✓ **It is better to assign a VM to a PM with a user (busy state) or to a PM (with available cores) that is already executing another VM (busy state).**
- ✓ **Physical machines are homogeneous.**

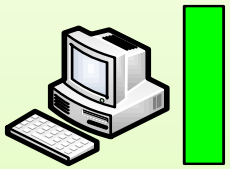
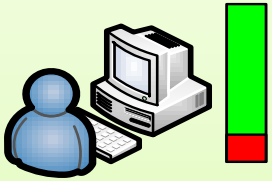
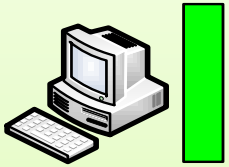
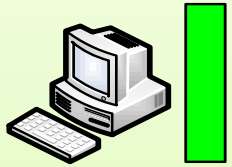
ENERGY-AWARE TECHNIQUES

Set of Virtual Machines (SVM)

2 CPU Cores	2 CPU Cores	2 CPU Cores	2 CPU Cores
VM1	VM2	VM3	VM4

Three different strategies/techniques were defined and tested to reduce the ECR consumed by UnaCloud

Set of Physical Machines (SPM)

1. IDLE	2. BUSY	3. HIBERNATED	4. TURNED OFF
0%	10%	0%	0%
			
8 CPU Cores	8 CPU Cores	8 CPU Cores	8 CPU Cores

1. CUSTOM ROUND ROBIN ALLOCATION

Algorithm 1 Custom Round Robin Algorithm

```
1: for all VMs VM_i do                                   SVM in arbitrary order
2:     for all PMs PM_j do                               SPM in arbitrary order
3:         if PM_j.available_cores ≥ VM_i.cores then
4:             assign VM_i to PM_j;
5:         end if
6:     end for
7: end for
```

- It does not consider the state of the PM.
- It does not use a minimum SPM.
- VMs with low requirements may be initially assigned to PMs with large CPU capabilities (blocking VMs with larger requirements).

2. SORTING VMS AND PMs TO MINIMIZE THE USE OF PMs

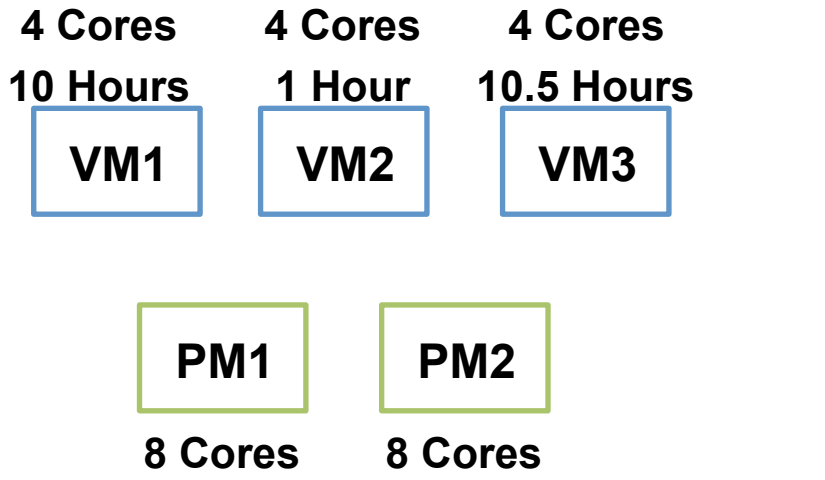
Algorithm 2 Sorting VMs and PMs to minimize the use of PMs

```
1: Sorting VMs and PMs;
2: for all VMs VM_i do
3:     Set the VMs;
4:     for all PMs PM_j do
5:         if PM_j.available_cores ≥ VM_i.cores then
6:             assign VM_i to PM_j;
7:             re-order PMs;
8:         end if
9:     end for
10: end for
```

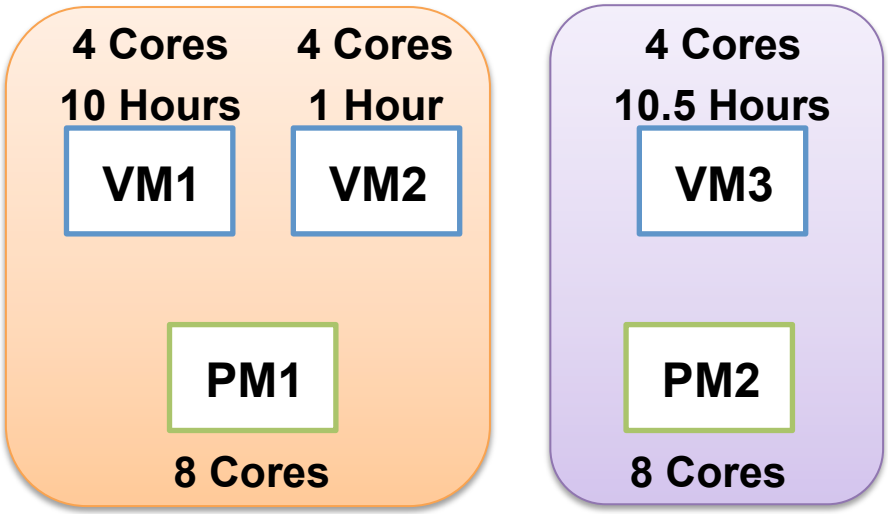
SVM descending ordered by cores required and execution time.

SPM ordered by (1) PMs running VMs, (2) PMs in busy state and (3) PMs with more available CPU cores.

2. SORTING VMS AND PMs TO MINIMIZE THE USE OF PMs



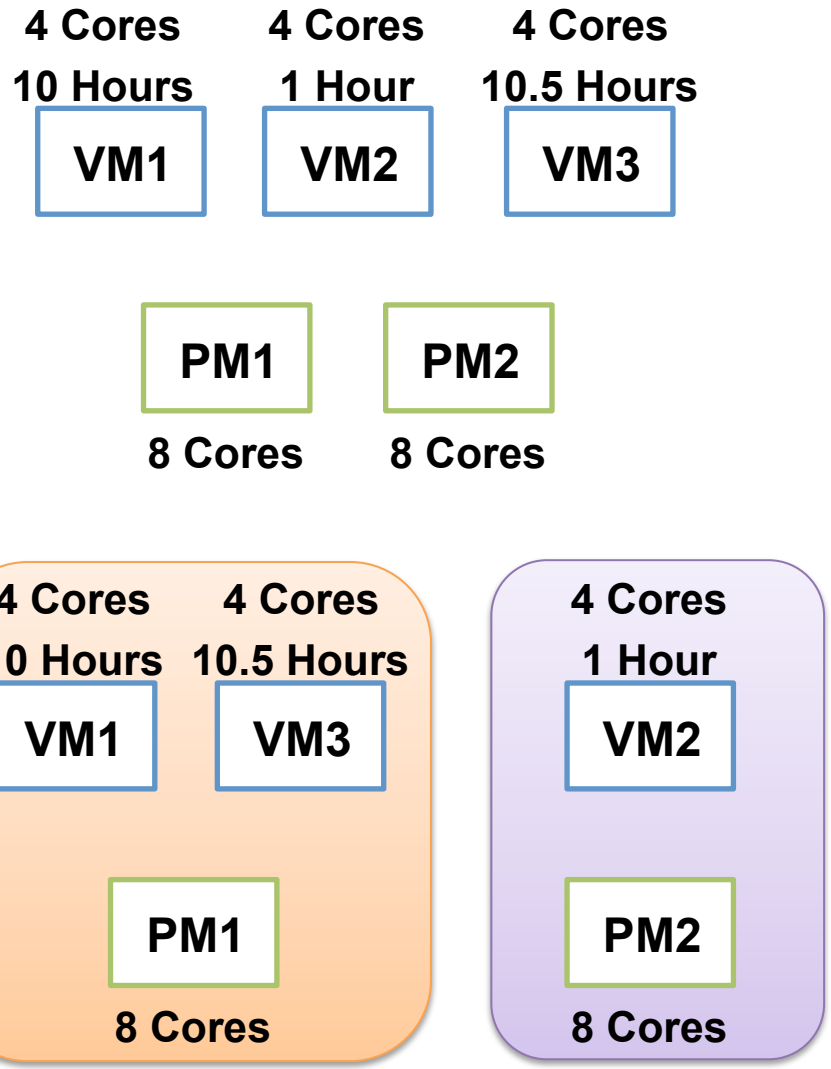
- ✓ We identified that if VMs with similar execution times are executed on the same PMs the ECR required to execute the SVM is lower.



- ✓ Every PMs used to executed the SVM will execute tasks at peak capacity during a similar period of time.

With Algorithm 2

3. EXECUTING VMS WITH SIMILAR EXECUTION TIME



With Algorithm 3

- ✓ We identified that if VMs with similar execution times are executed on the same PMs the ECR required to execute the SVM is lower.
- ✓ Every PMs used to executed the SVM will execute tasks at peak capacity during a similar period of time.

3. EXECUTING VMS WITH SIMILAR EXECUTION TIME

Algorithm 3 Executing VMs with Similar Execution Time within same PM

```
1: Sorting VMs and PMs;
2: for all PMs PM_j do
3:     Set the PMs;
4:     for all VMs VM_i do
5:         Set the VMs for all PMs;
6:         if !VM_i.assigned and PM_j.available_cores ≥
           VM_i.cores then
7:             taskI = VM_i
8:             repeat
9:                 assign taskI to PM_j
10:                if PM_j still have cores then
11:                    assigns VMs with similar execution time;
12:                end if
13:            until PM_j have available cores and !taskI
14:        end if
15:    end for
16: end for
```

ALGORITHM TO CALCULATE THE ECR

Algorithm to Calculate the ECR Consumed by Set of Virtual Machines

```
1: totalPower =0
2: for all PMs PM_j do
3:    PM_j = PMs.get(j)
4:    PM_j.sortDescVMsAssignedByExecutionTime
5:    for all VMs VM_i do
6:       VM_i = VMs.get(i)
7:       if i is 0 then
8:          if PM_i.available_core is 0 and PM_i.with_a_user
            then
9:             executionTime = VM_j.executionTime × 1.1
10:         else
11:            executionTime = VM_j.executionTime
12:         end if
13:      else
14:         executionTime = VM_i.executionTime −
            VM_{i−1}.get.executionTime
15:      end if
16:      if PM_j.availableCores is 0 then
17:         percentageOfCPUUsed = 100
18:      else
19:         percentageOfCPUUsed = (PM_j.usedCores / PM_j.totalCores) ×
            100
20:      end if
21:      if PMi.withUser and percentageOfCPUUsed ≠
         100 then
22:         percentageOfCPUUsed+ = 10
23:      end if
24:      ECRDuringTheExecutionTime = 45.341 ×
         percentageOfCPUUsed^0.1651
25:      totalPower+ = ECRDuringTheExecutionTime ×
         executionTime
26:      PM_j.usedCores− = VM_i.usedCores
27:   end for
28:   if PM_j.VMs.size is 0 and PM_i.with_a_user then
29:      totalPower+ = 87
30:   end if
31: end for
```

- ✓ **This algorithm is used once the scheduling process (using one of the 3 previous algorithms) has finished.**
- ✓ **The ECR consumed by every PM is calculated based on the ECR of each VM.**
- ✓ **The goal of this algorithm is to calculate the total power consumed by the VMs.**

EXPERIMENTAL RESULTS

- ✓ We conduct experiments based on simulations.
- ✓ We use workloads based on real VMs production traces (collected during one year).
- ✓ We consider that a given percentage of PMs has a user working on it.
- ✓ We vary the percentage of busy machines from 10% up to 50% with increments of 10%.

EXPERIMENTAL RESULTS

Table III: Simulation results when 10% of PMs are used

VMs	T. Cores	Alg.	UPMs	En.(kW)	ECR Gain
40	127	1	29	91.20	-
		2	20	80.94	11.35
		3	20	79.25	13.20
50	155	1	36	112.00	-
		2	23	99.6	11.63
		3	23	98	13.04
60	181	1	42	123.46	-
		2	27	101.18	18.05
		3	27	99.48	19.43
70	219	1	50	161.74	-
		2	31	132.56	18.00
		3	31	132.35	18.17
80	235	1	55	169.87	-
		2	33	132.6	22.18
		3	33	128	24.83

EXPERIMENTAL RESULTS

Table IV: Simulation results when 50% of PMs are used

VMs	T. Cores	Alg.	UPMs	En.(kW)	ECR Gain
40	127	1	29	96.55	-
		2	16.46	66.63	30.99
		3	16.46	64.9	32.69
50	155	1	36	118.55	-
		2	20	81.52	31.24
		3	20	79.67	32.79
60	181	1	42.2	129.15	-
		2	27	86.70	32.87
		3	27	84.75	34.38
70	219	1	50	168.29	-
		2	36	112.4	33.21
		3	36	113.56	32.52
80	235	1	55	176.00	-
		2	40.2	117.78	33.08
		3	40.2	115	34.62

EXPERIMENTAL RESULTS

- ✓ On average Algorithm 2 and Algorithm 3 save up to 36% of PMs more than Algorithm 1 when only 10% of the PMs are busy by users.
- ✓ When the number of PMs with users increases up to 50%, Algorithm 2 and Algorithm 3 require 26% less PMs than RR.

EXPERIMENTAL RESULTS

- ✓ For the first scenario, the average gain on ECR by Algorithm 2 is up to 19% and up to 20% by Algorithm 3.
- ✓ In scenario 2, Algorithm 2 and Algorithm 3 save up to 29% and 30% more than Algorithm 1, respectively.
- ✓ Algorithm 3 saves 2% more energy in both scenarios than Algorithm 2.

CONCLUSIONS AND FUTURE WORK

- ✓ We developed and simulated different energy-aware algorithms to allow UnaCloud to operate in an energy efficient way.
- ✓ We will implement the new proposed algorithms on the UnaCloud infrastructure.
- ✓ We will try more allocation strategies that consider VMs shared resources constraints that can lead to performance degradation on the quality of service.

THANKS FOR YOUR ATTENTION!

CONTACT INFORMATION

mj.villamizar24@uniandes.edu.co

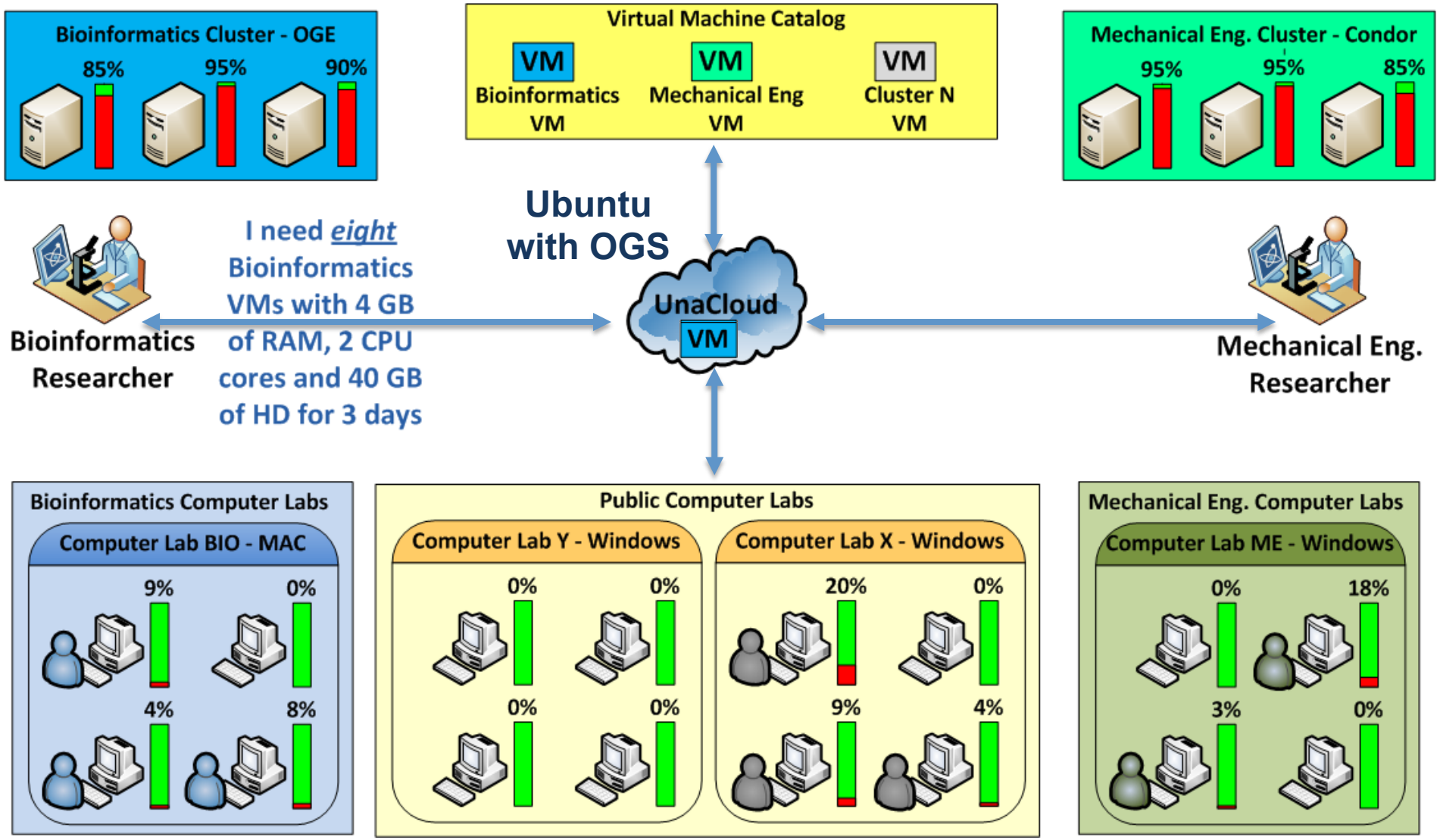
mjvc007@hotmail.com

@mariocloud

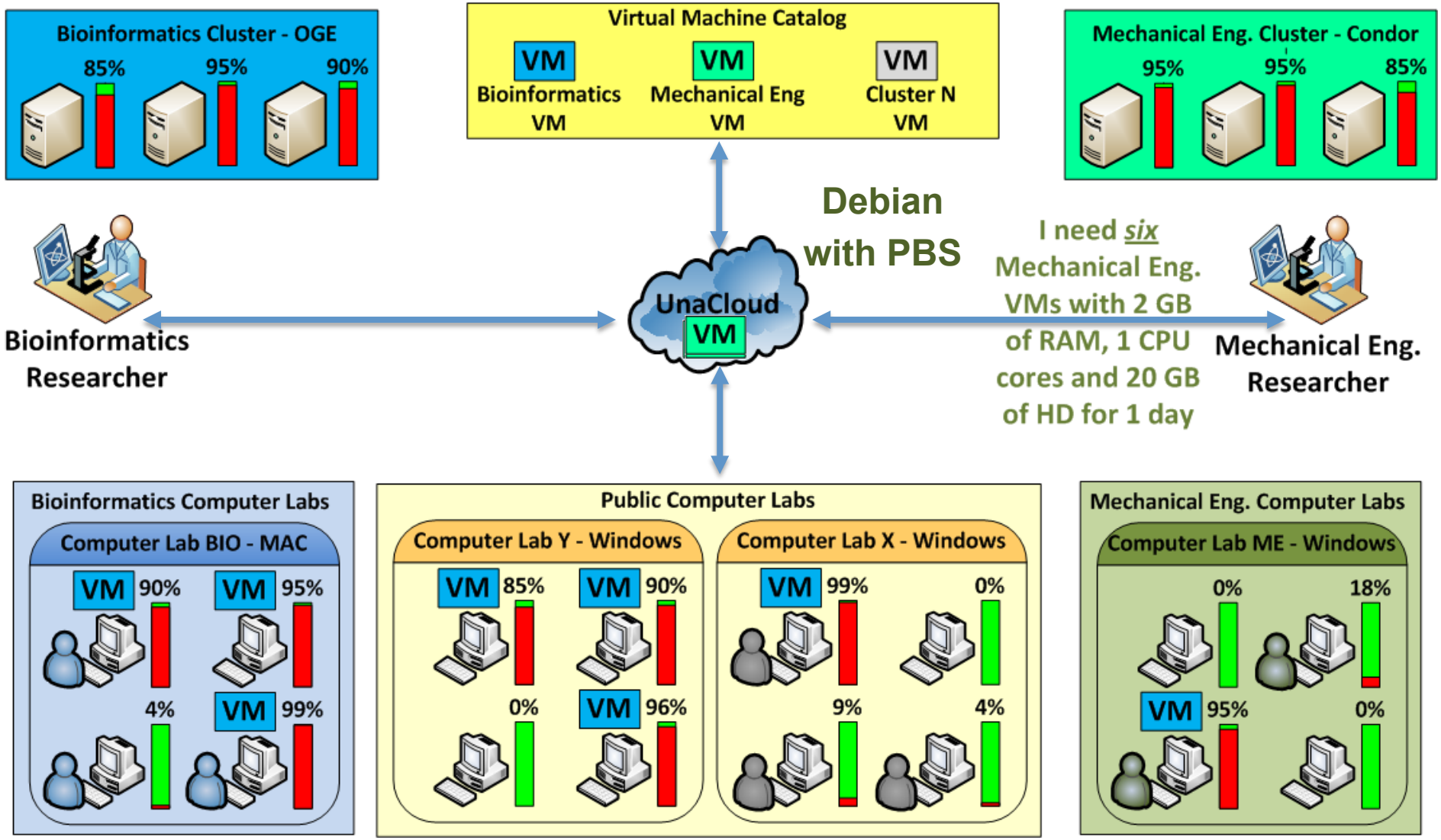
http://linkedin.com/in/mariojosevillamizarcano

Mario José Villamizar Cano

THE DESIRED SOLUTION



THE DESIRED SOLUTION



THE CONTEXT

Great Internet Mersenne Prime Search
GIMPS

SETI HOME

BOINC

InteGrade

distributed.net

Condor
High Throughput Computing

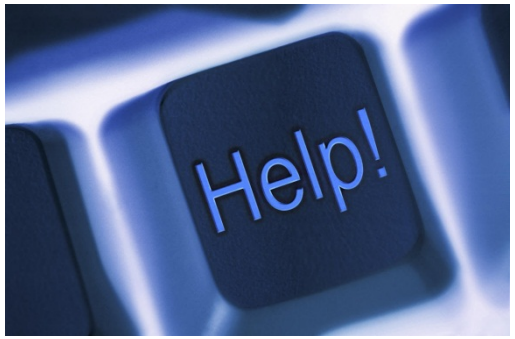
OurGrid

An alternative are Desktop Grids and Volunteer Computing Systems (DGVCS's):

- **Offer large scale computing infrastructures at low cost.**
- **Use inexpensive resources, most of them underutilized desktop computers.**
- **Interconnect thousands of computing resources available through Internet or Intranet environments.**
- **Are based on resources that are *non-dedicated*, distributed, highly heterogeneous, and part of independent administrative domains.**

THE PROBLEM

When a research group wants to use a DGVCS it regularly find that:



- They will need to recode, modify or adapt every application that is going to be executed on the DGVCS, for several research groups and tens of existing applications it is a complex process.
- The installation, configuration, maintaining and use of most DGVCSs require of people with some/advanced skills in applications and IT infrastructures.
- For using the idle processing capabilities of hundreds of commodity desktops, they will need to configure manually every desktop computer with the DGVCS software.

THE PROBLEM

When a research group wants to use a DGVCS it regularly find that:

- **Administrators of different computer labs do not want that external people modify the configurations of the physical machines.**
- **Most of the physical desktops (99%) machines available in computer labs have Windows operating systems.**
- **They would like to share easily with other research groups the idle capabilities available in computer labs using a shared model.**