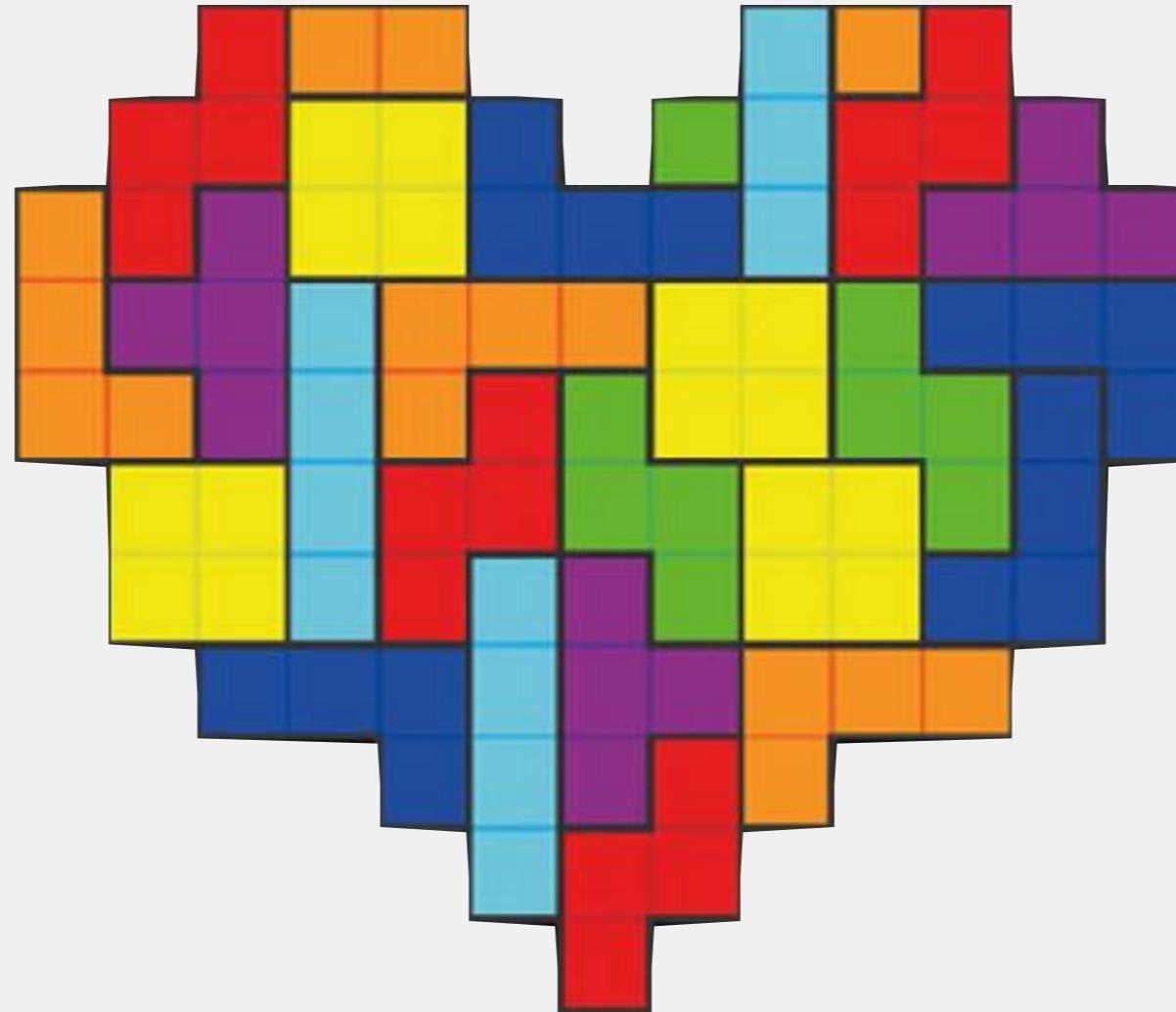


# Placing Virtual Machines in a Cloud under Constraints



Fabien Hermenier  
– placing rectangles since 2006

@fhermeni  
[fabien.hermenier@nutanix.com](mailto:fabien.hermenier@nutanix.com)  
<https://fhermeni.github.io>



2006 - 2010

PhD - Postdoc

Gestion dynamique des tâches dans les grappes,  
une approche à base de machines virtuelles



2011

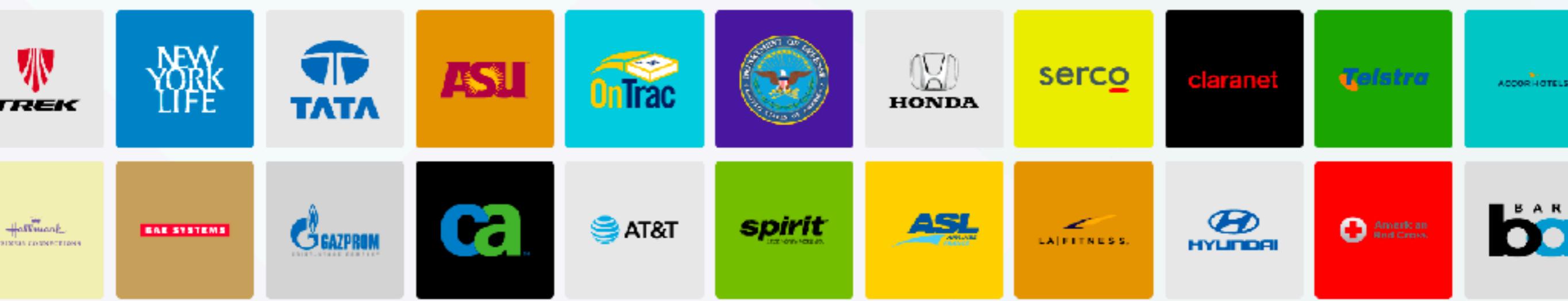
Postdoc

How to design a better testbed:  
Lessons from a decade of network experiments

2011 - 2016

Associate professor

VM scheduling, green computing

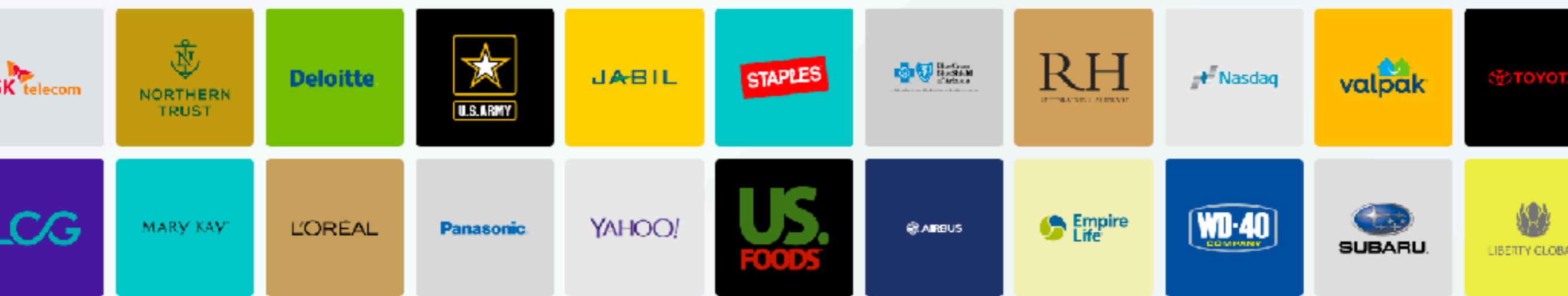


# NUTANIX™

Enterprise cloud company

“Going beyond hyperconverged infrastructures”

VM scheduling, resource management  
Virtualization



I am  
from the (distributed) system community  
not  
from the CP community



# Inside a private cloud



# Clusters

from 2 to  $x$  physical servers

---

isolated applications

---

virtual machines  
containers

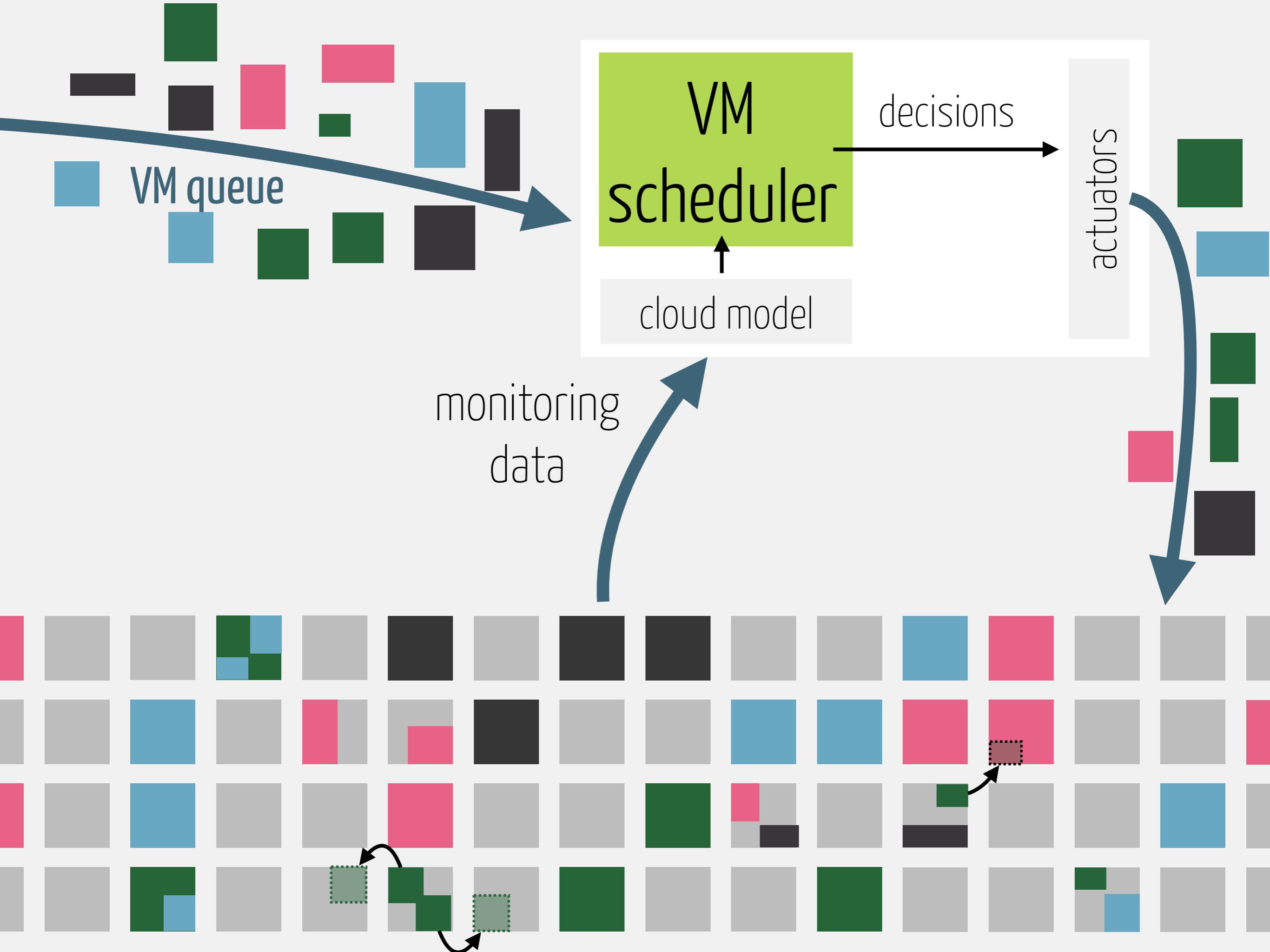
---

storage layer

---

SAN based: converged infrastructure  
shared over the nodes: hyper-converged infrastructure





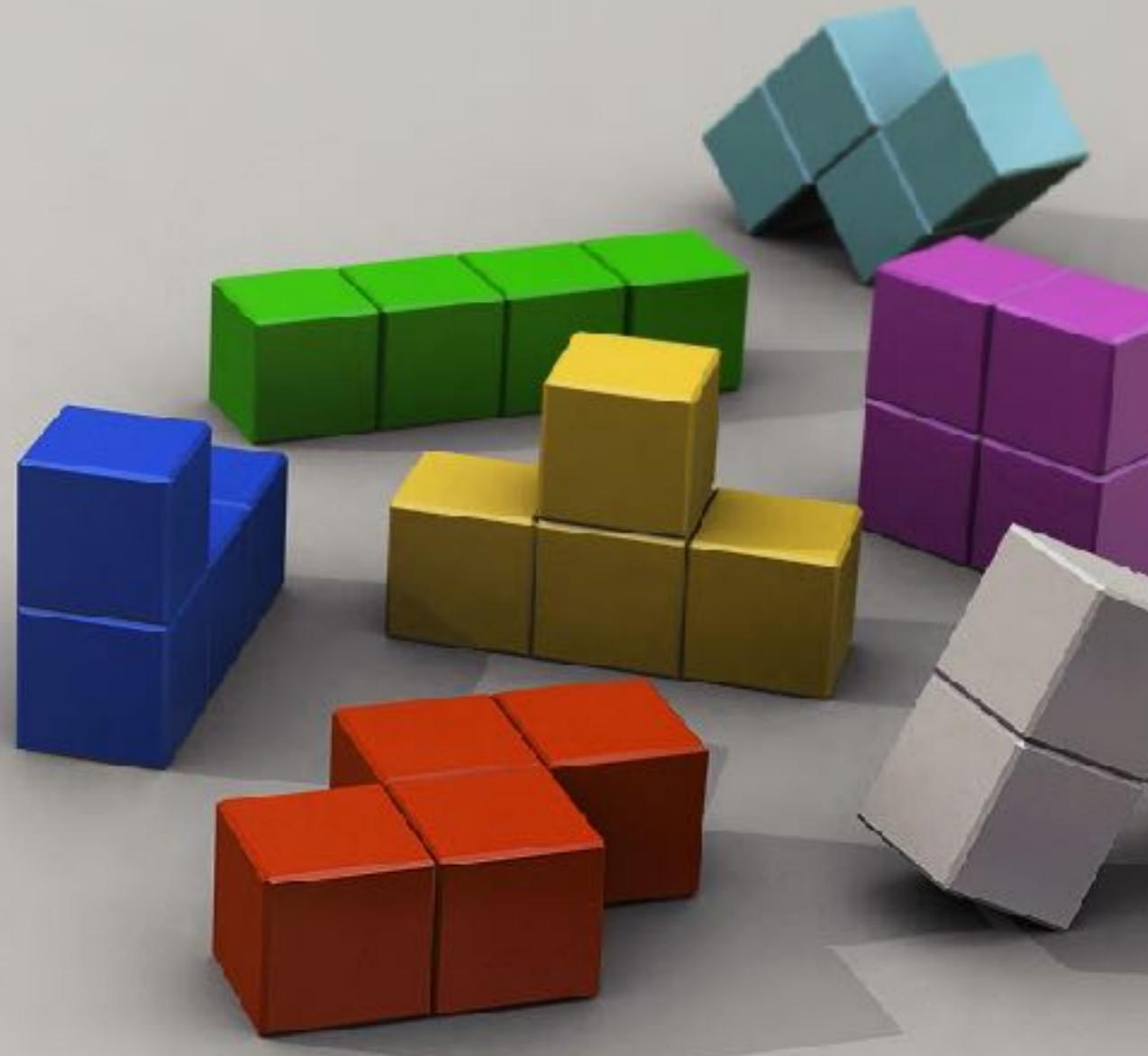
# VM scheduling

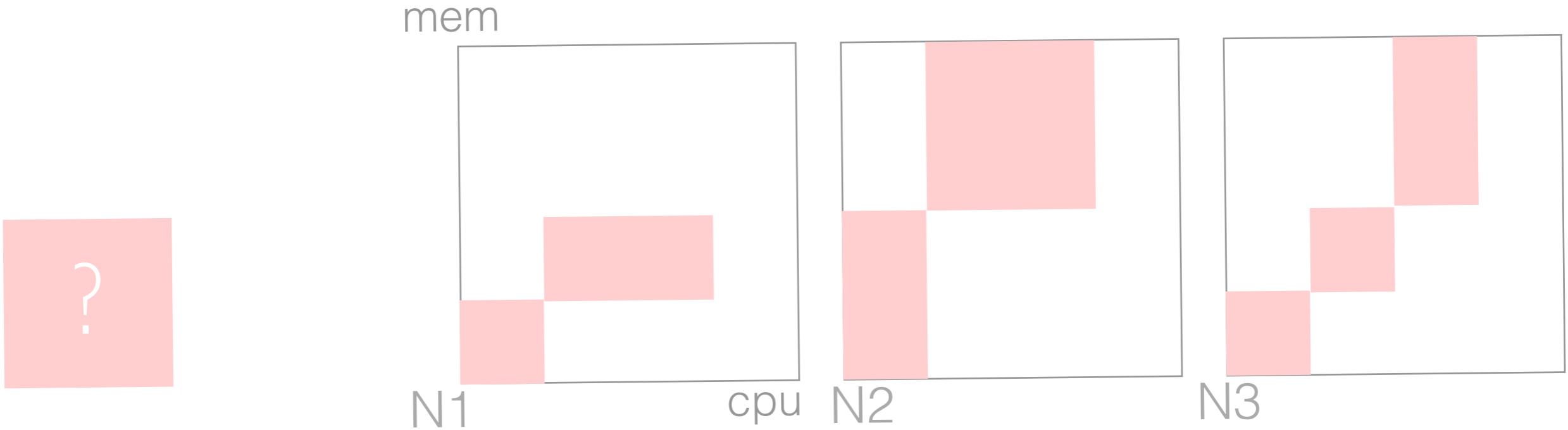
find a server to every VM to run

Such that

- compatible hw
- enough pCPU
- enough RAM
- enough storage
- enough whatever

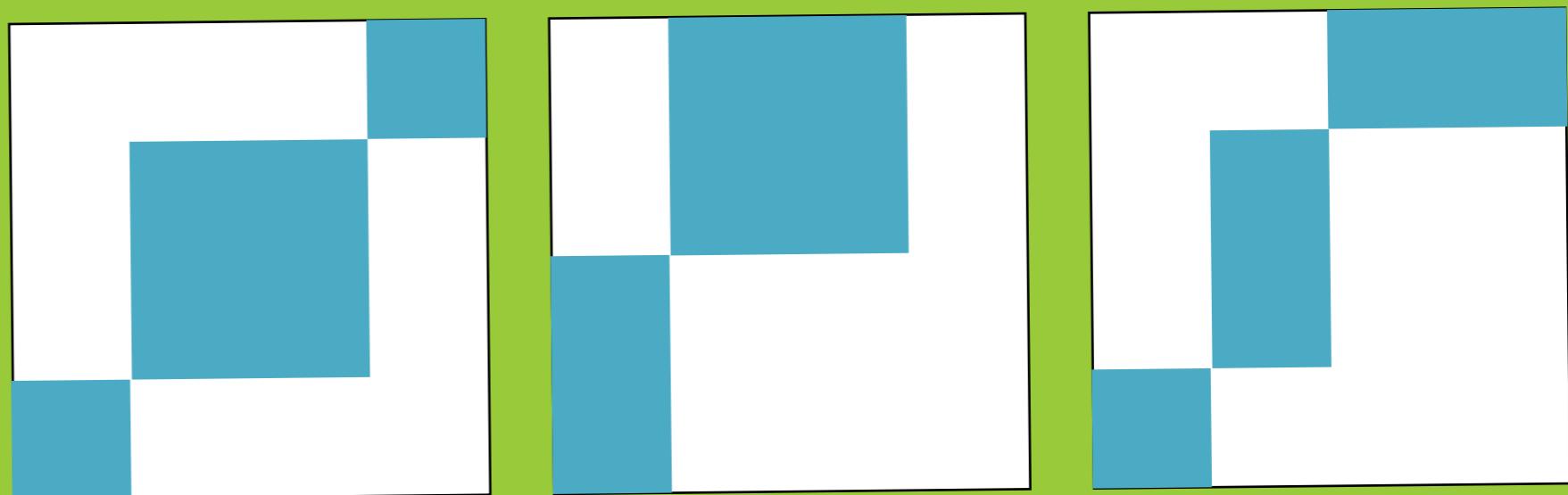
While  
min or max sth



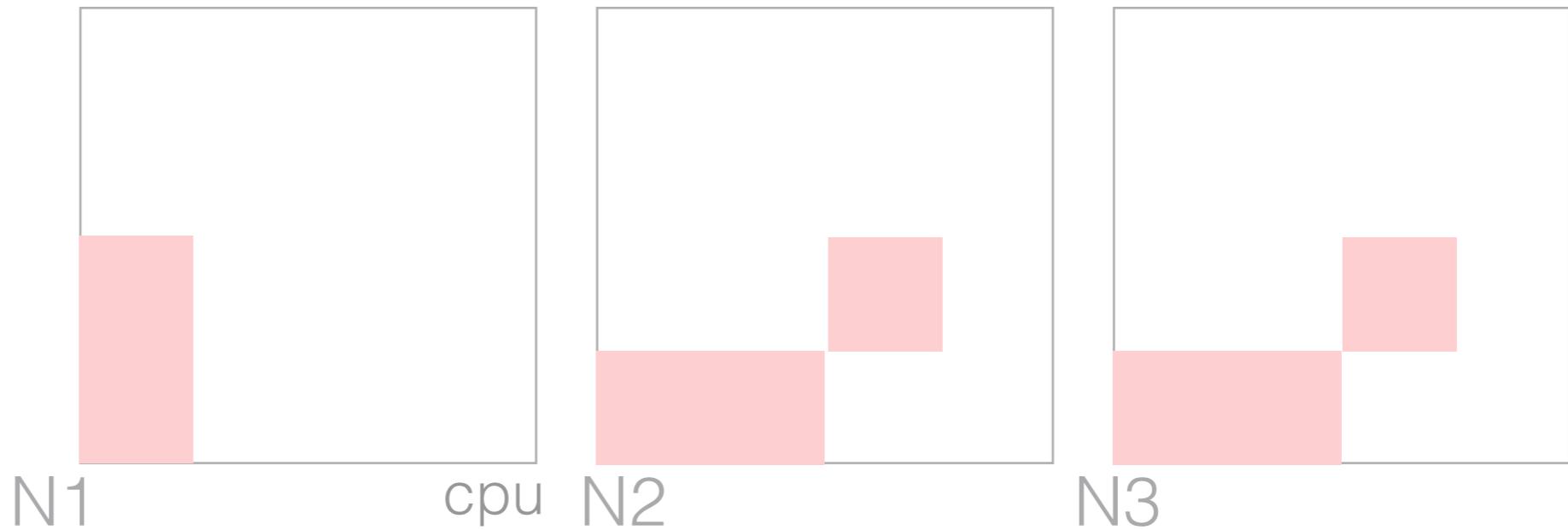


# A good VM scheduler provides

Bigger business value,  
same infrastructure

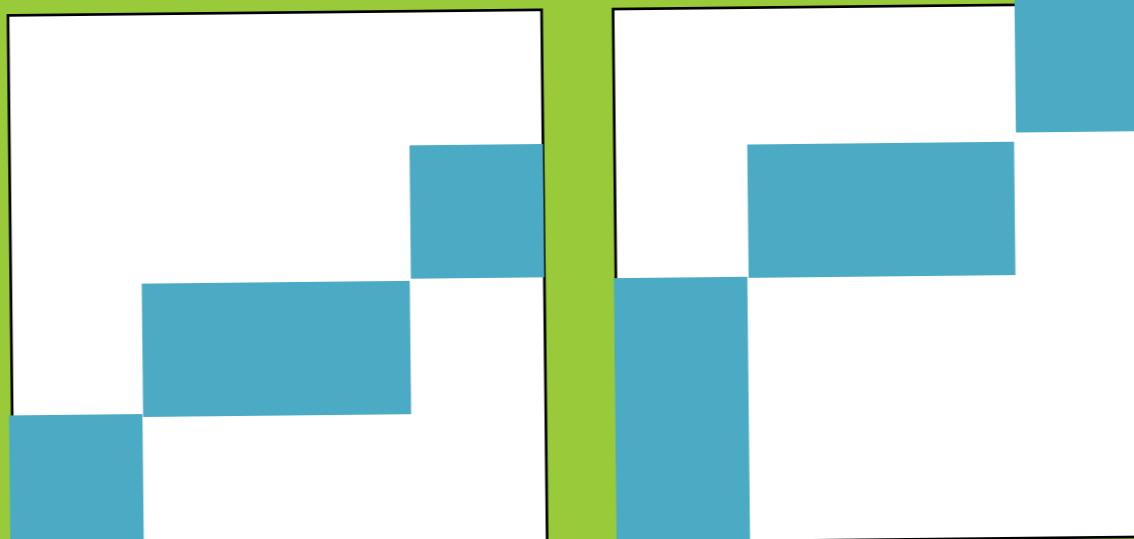


mem



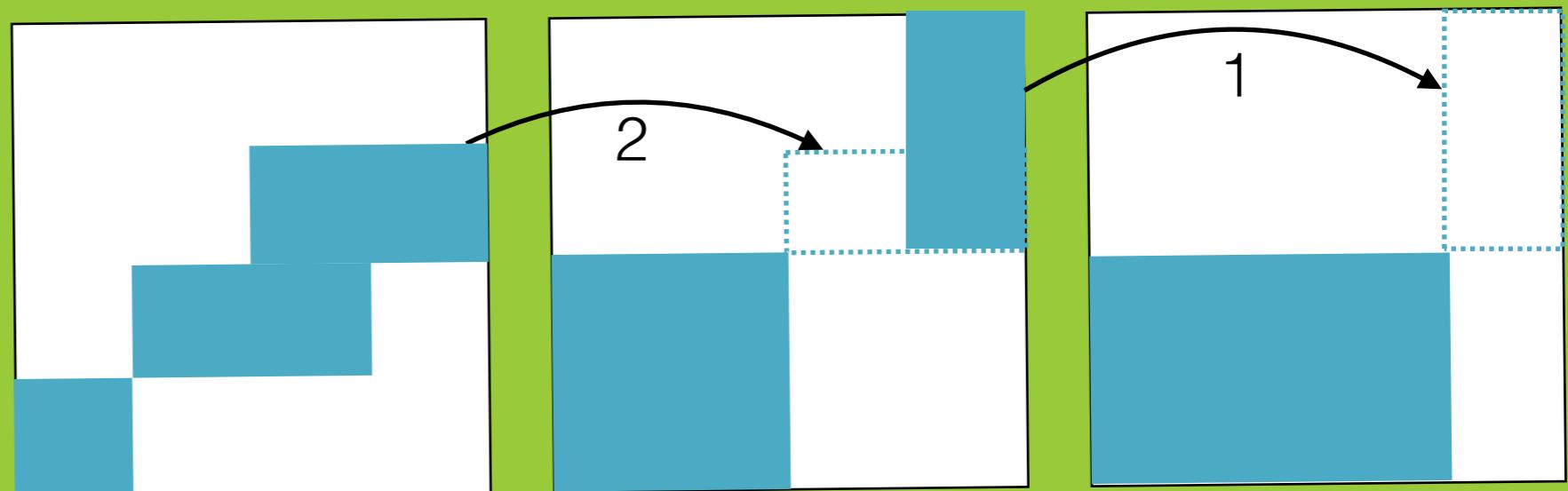
# A good VM scheduler provides

Same business value,  
smaller infrastructure



# A good **dynamic** VM scheduler fixes issues online

hotspot mitigation  
re-balancing  
dynamic packing





KEEP  
CALM  
AND  
CONSOLIDATE  
AS HELL

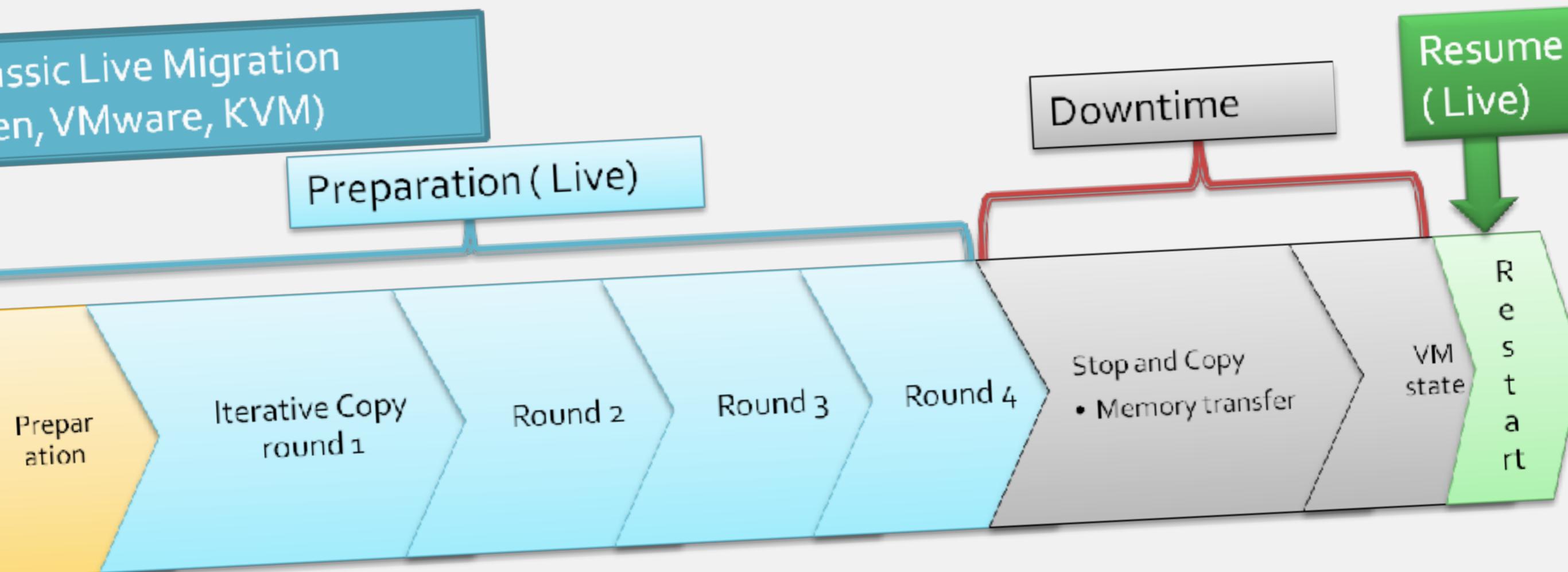
1 node =



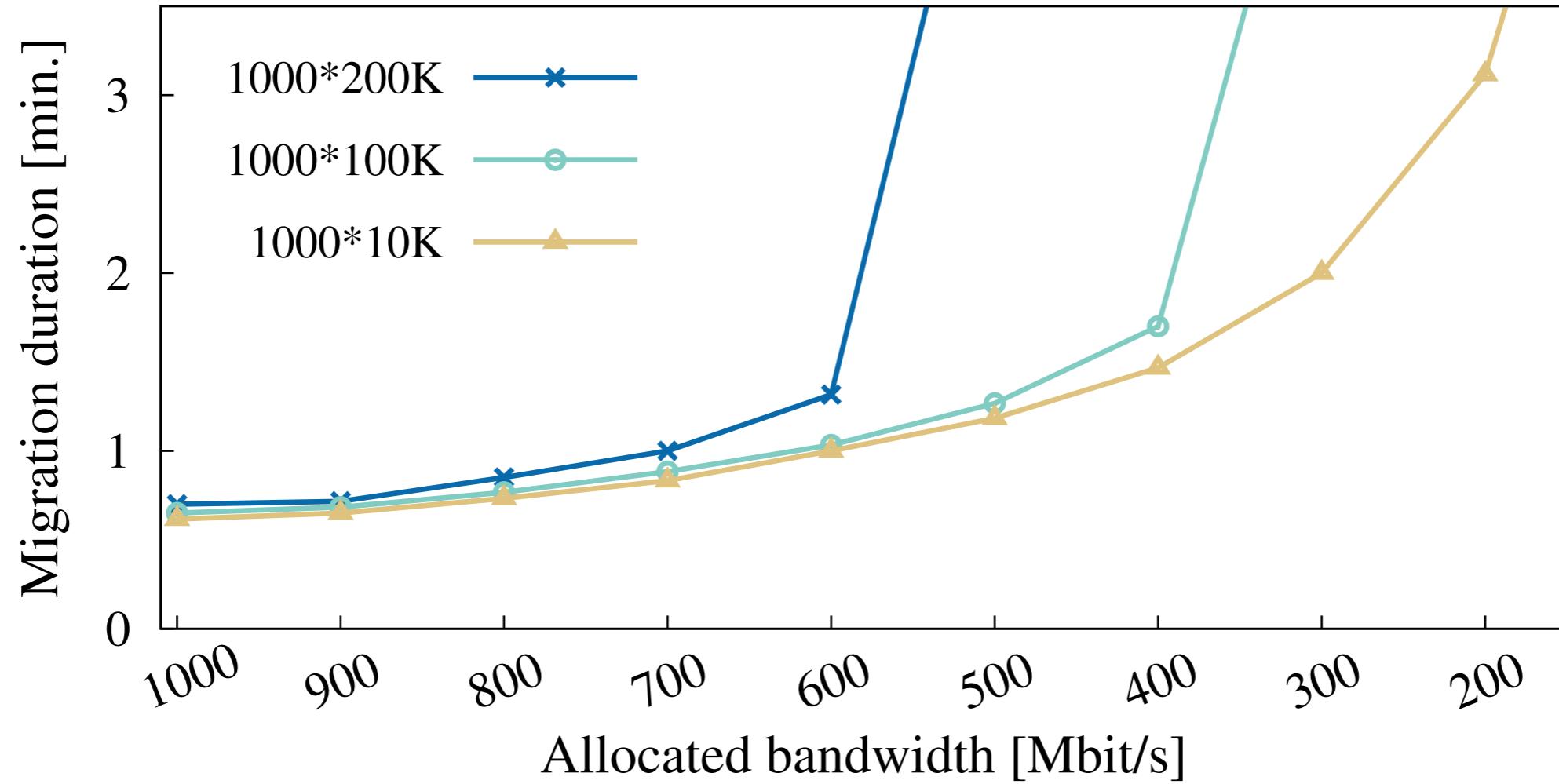
VDI workload:  
12+ vCPU/1 pCPU

100+ VMs / server

# 2005: Live-migration of VMs



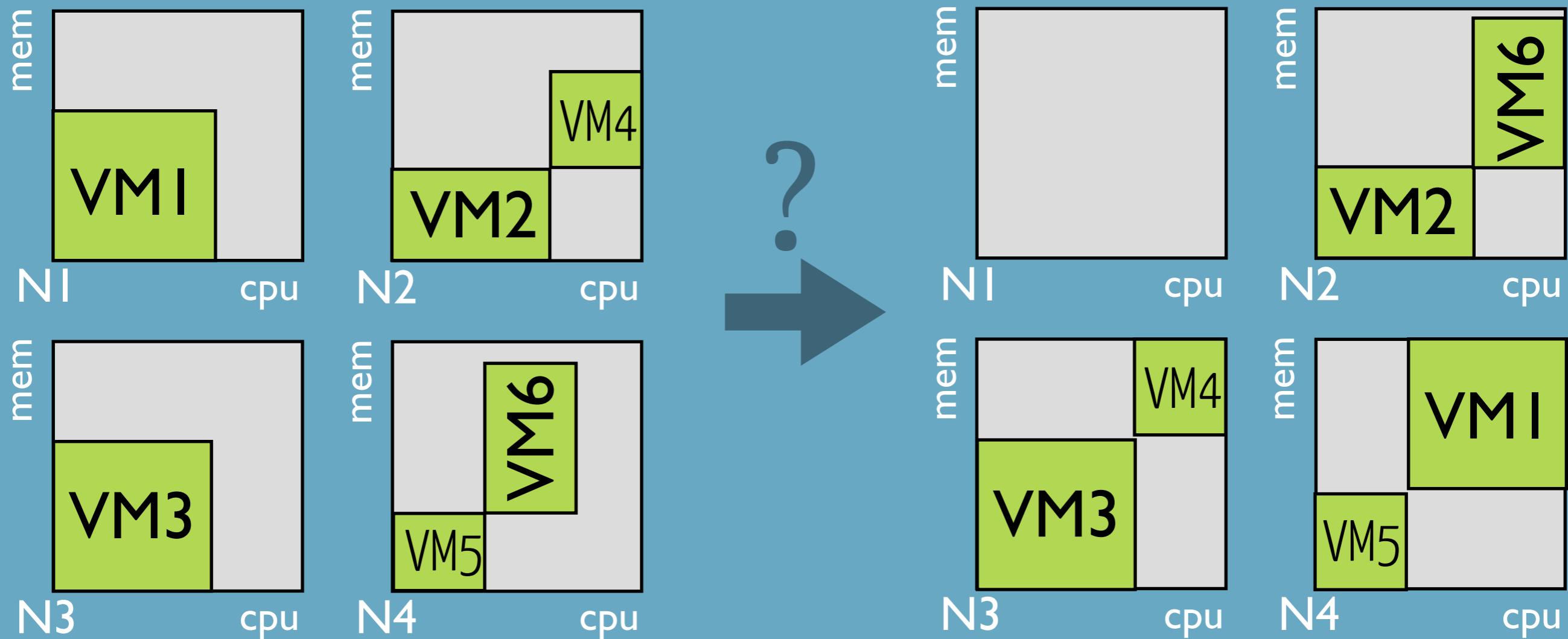
temporary,  
resources are used on the source and  
the destination nodes



Migrations are costly

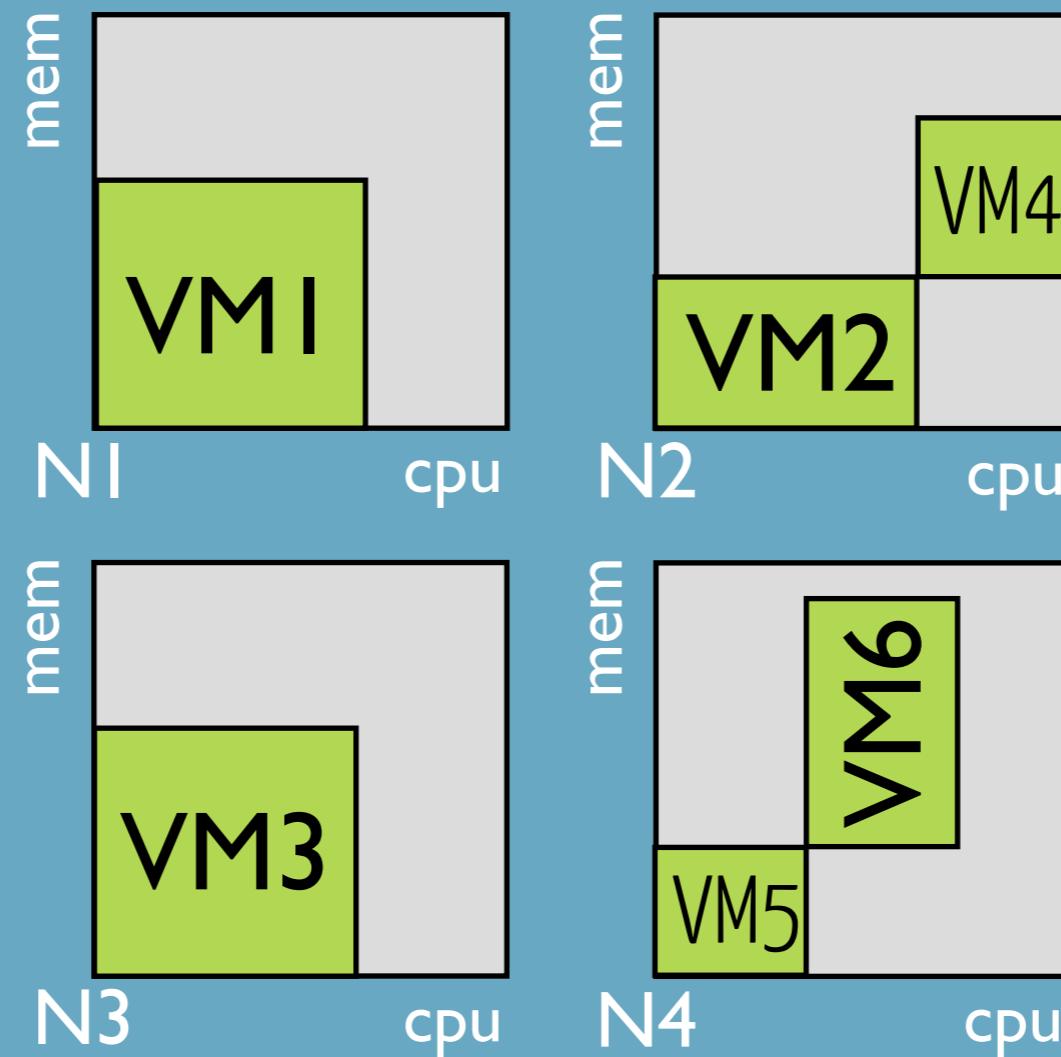
# dynamic schedulers

dependency management



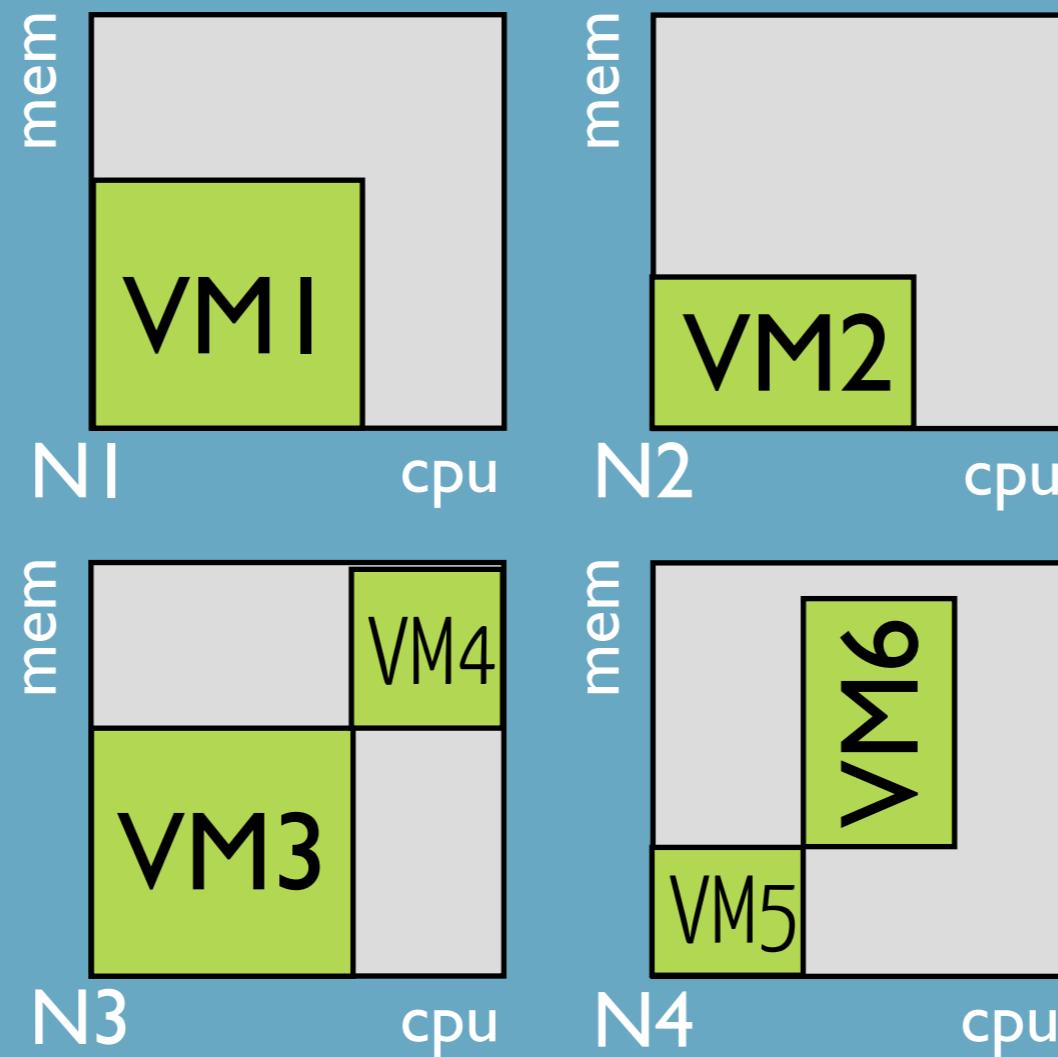
# dynamic schedulers

dependency management



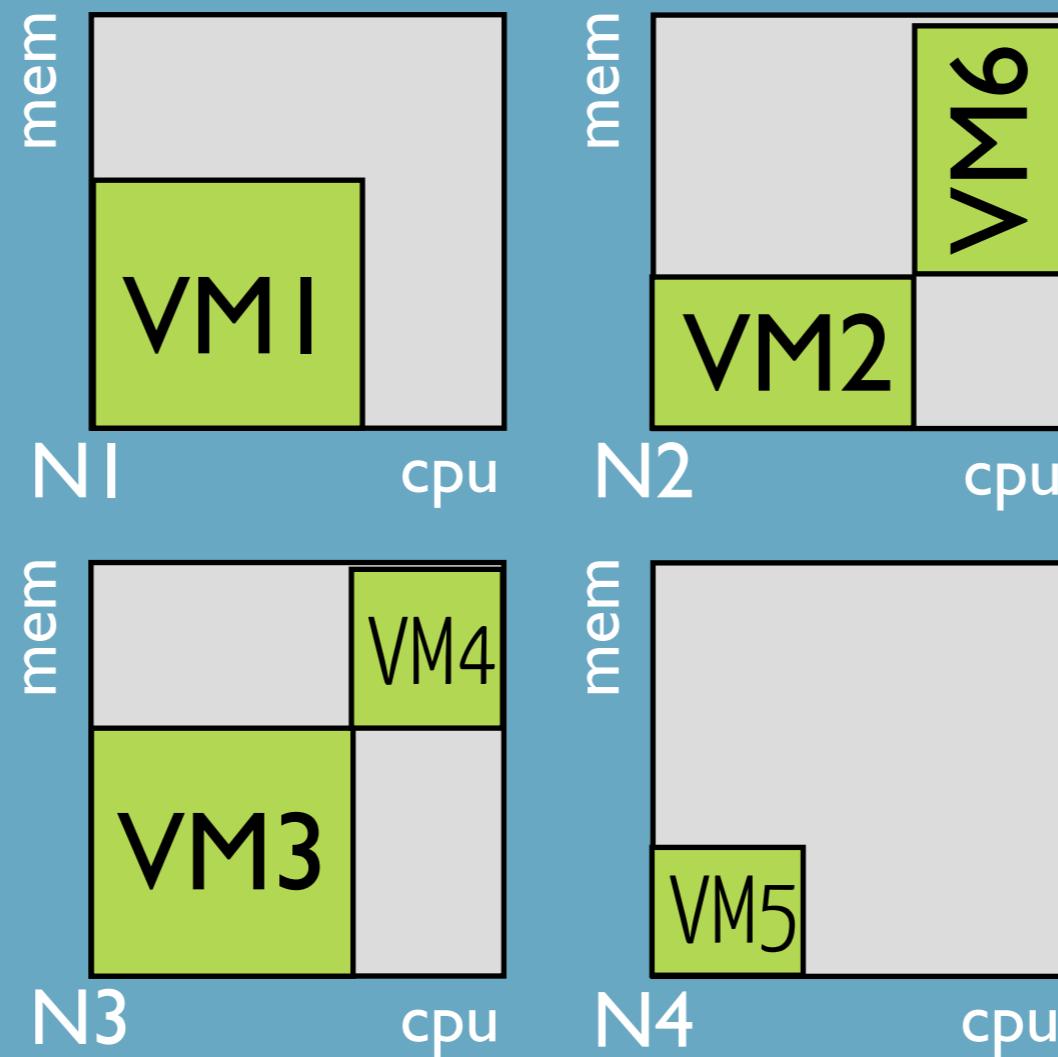
# dynamic schedulers

dependency management



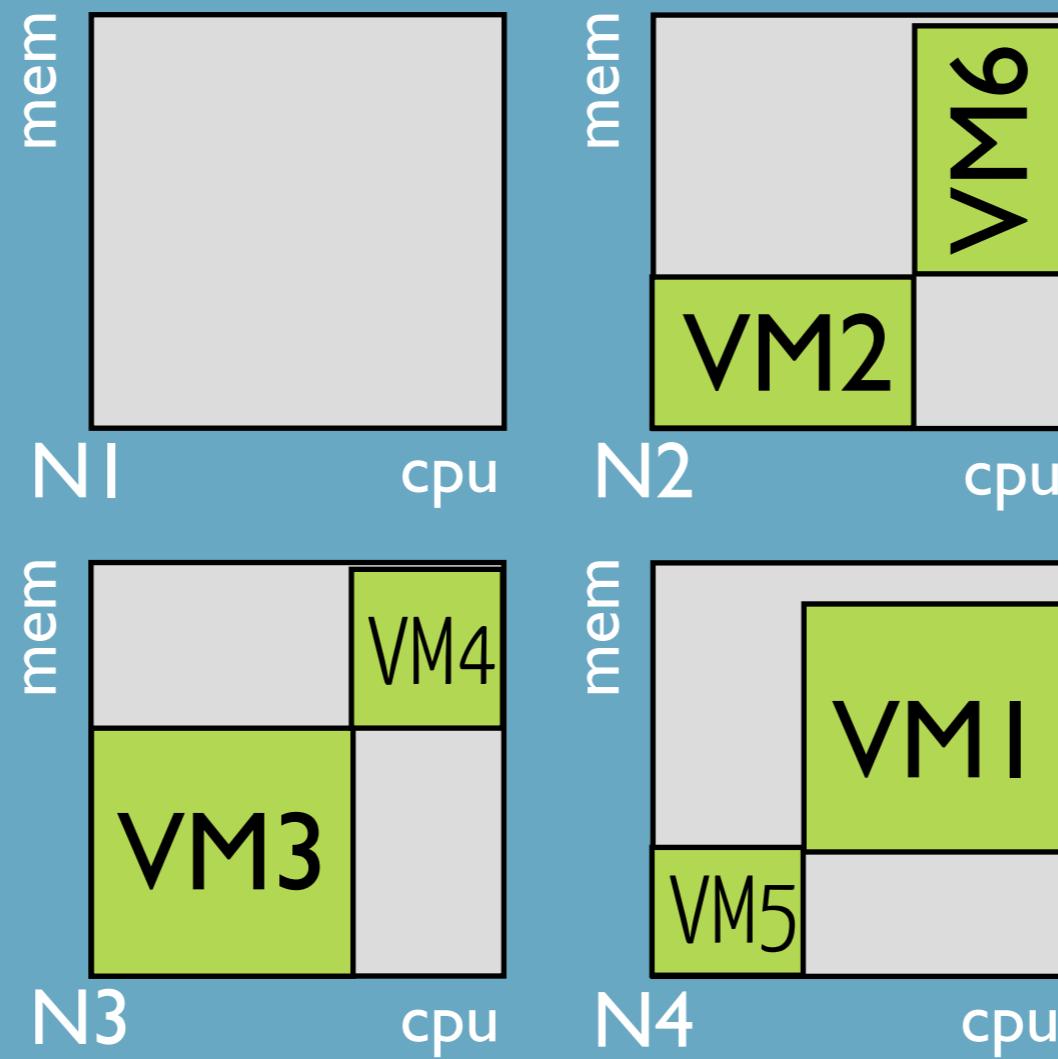
# dynamic schedulers

dependency management



# dynamic schedulers

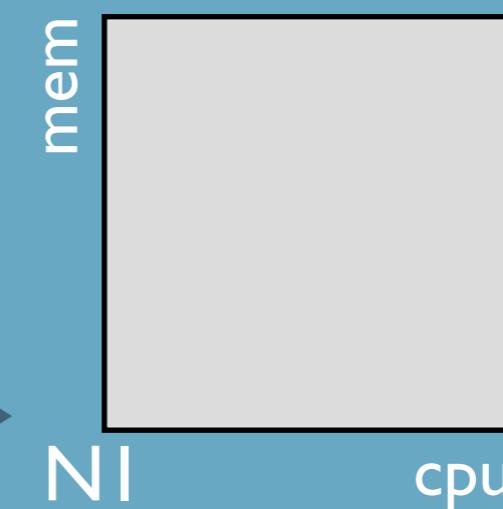
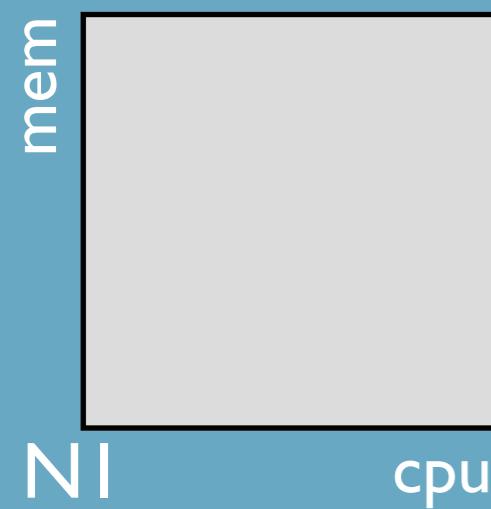
dependency management



# dynamic schedulers

cyclic dependencies

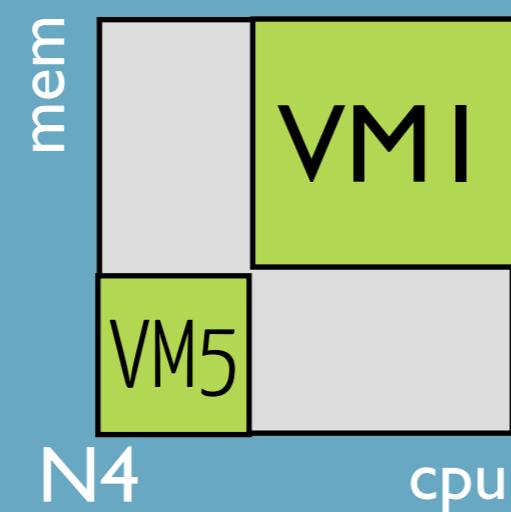
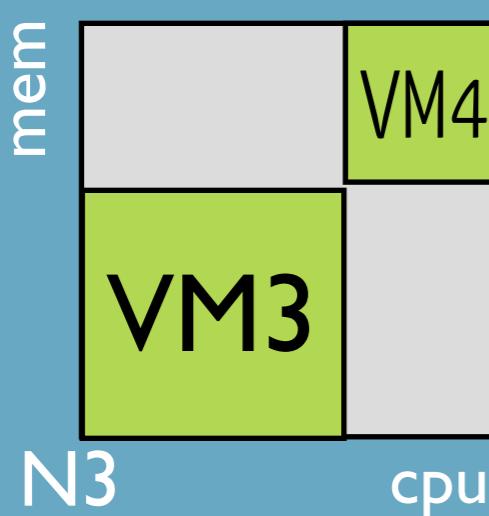
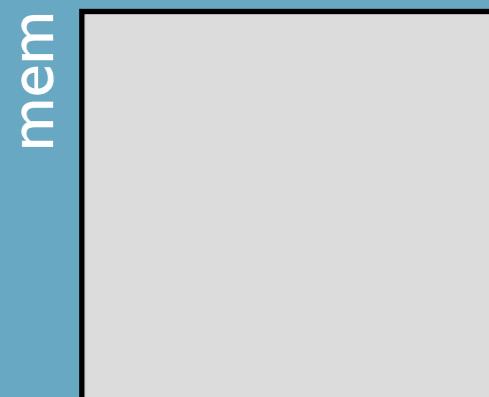
anti-affinity(VM3,VM4)  
min(#onlineNodes)



anti-affinity(VM3,VM4)  
min(#onlineNodes)

# dynamic schedulers

cyclic dependencies

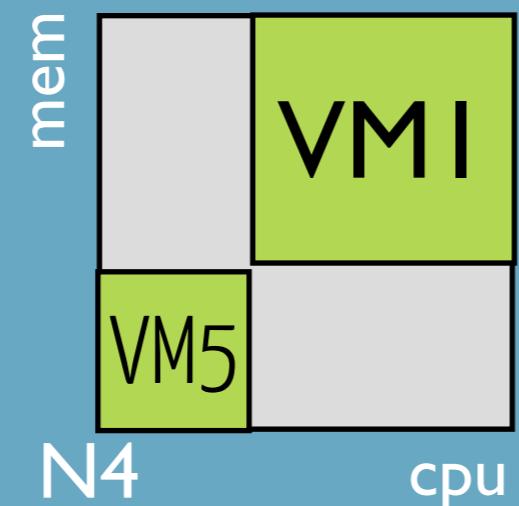
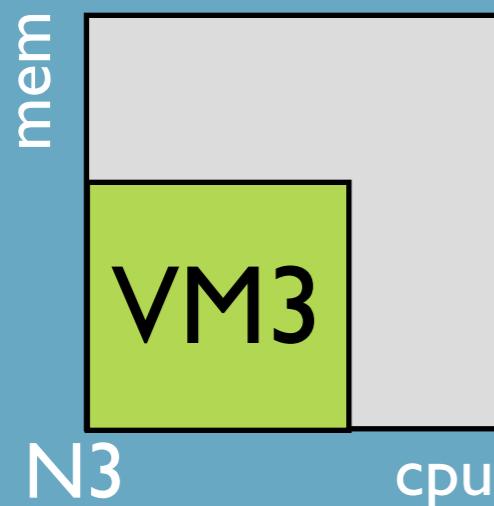
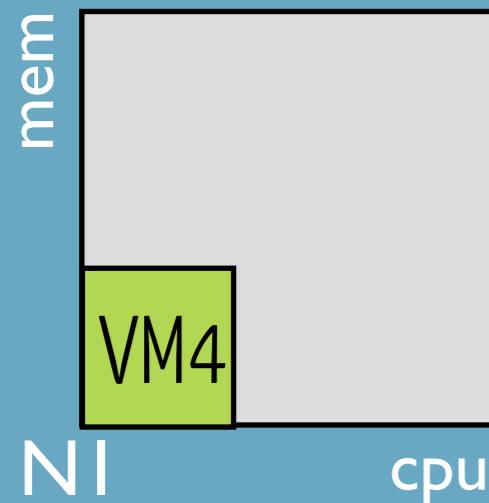


a pivot to break the cycle

fix or prevent the situation ?

# dynamic schedulers

cyclic dependencies

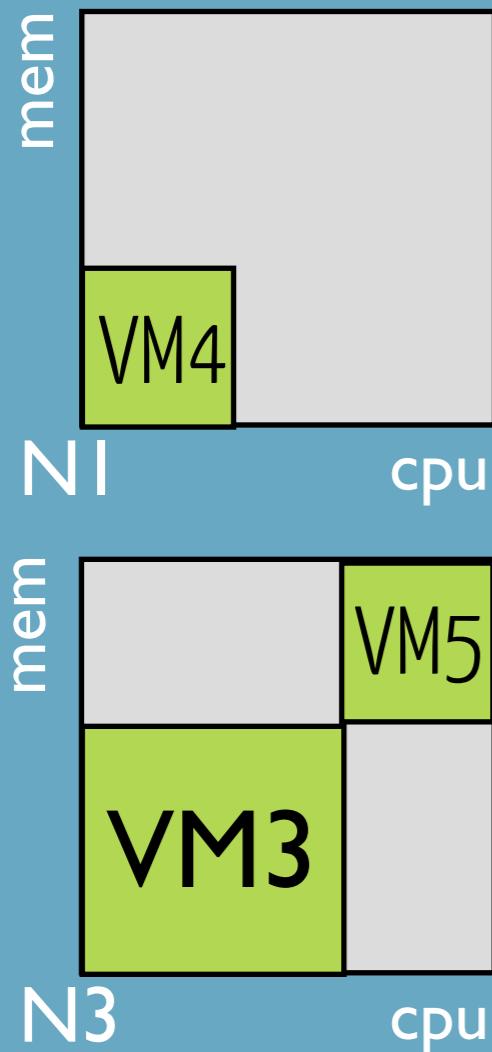


a pivot to break the cycle

fix or prevent the situation ?

# dynamic schedulers

cyclic dependencies

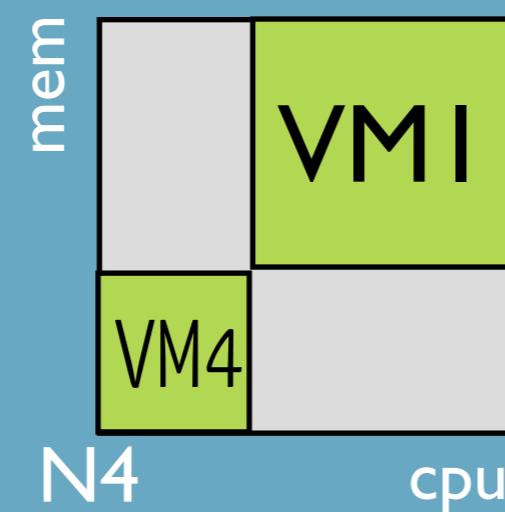
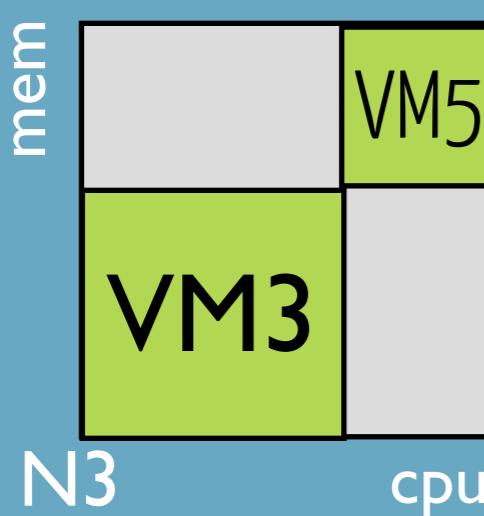
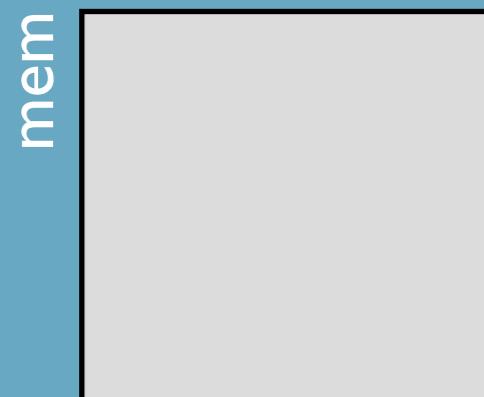


a pivot to break the cycle

fix or prevent the situation ?

# dynamic schedulers

cyclic dependencies

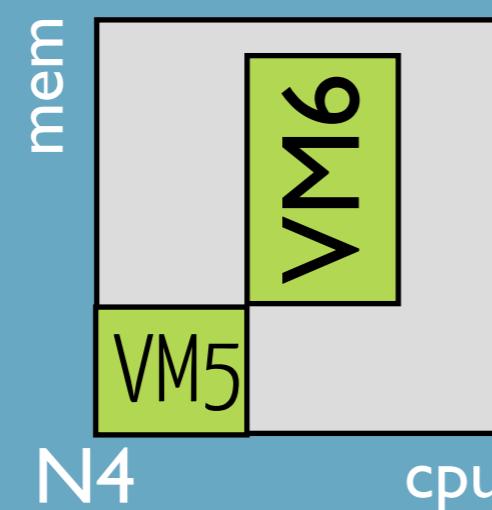
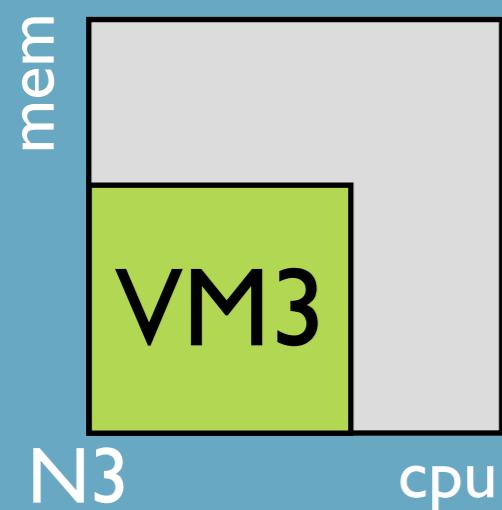
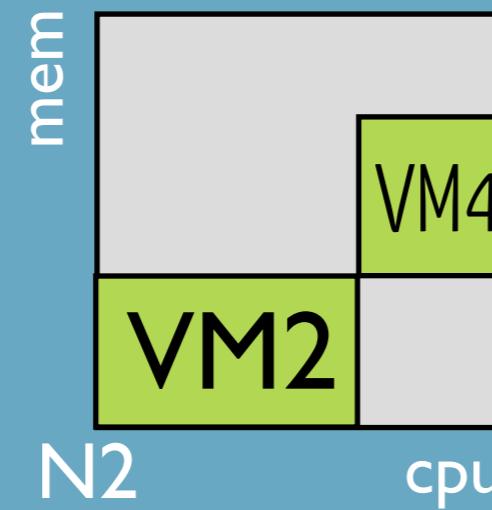
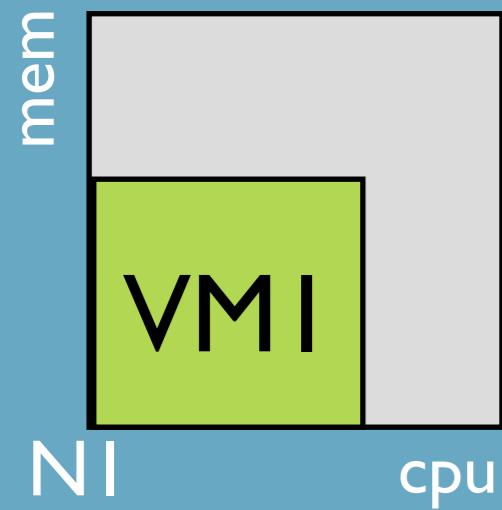


a pivot to break the cycle

fix or prevent the situation ?

# dynamic schedulers

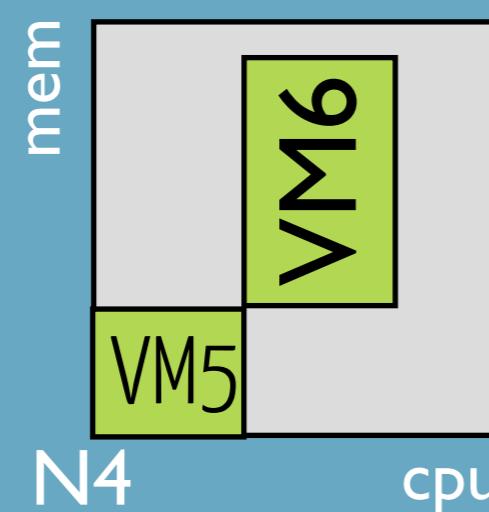
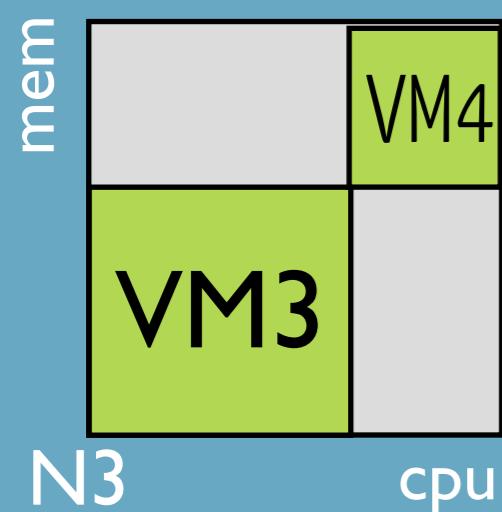
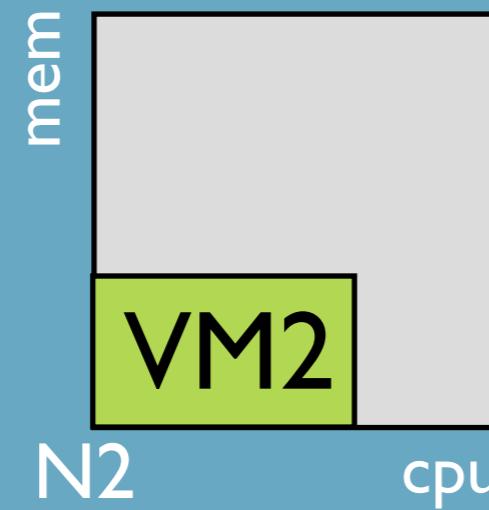
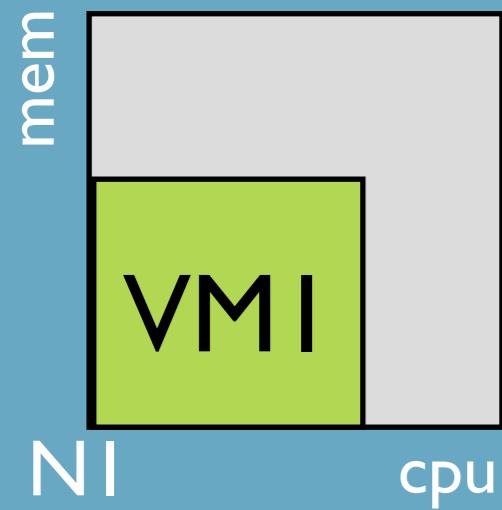
quality at a price



$$\min(\#\text{onlineNodes}) = 3$$

# dynamic schedulers

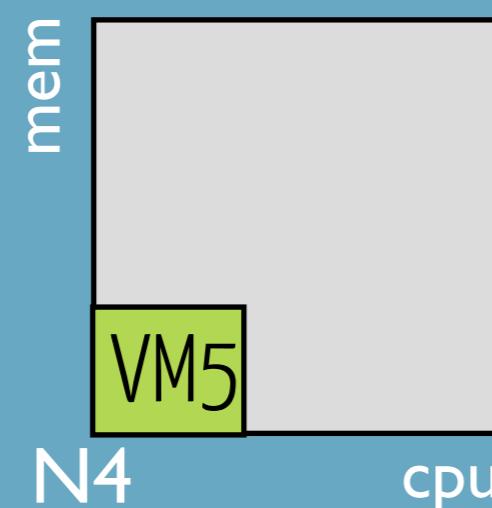
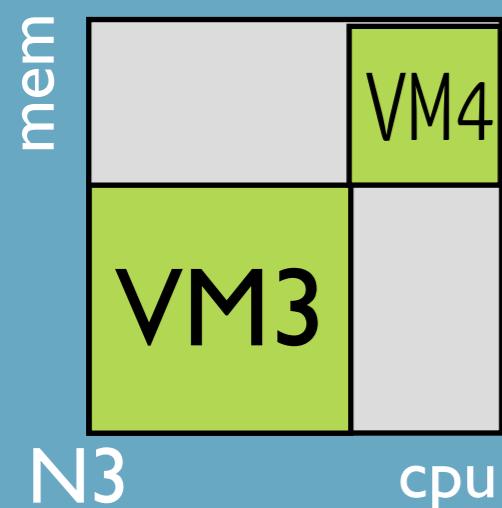
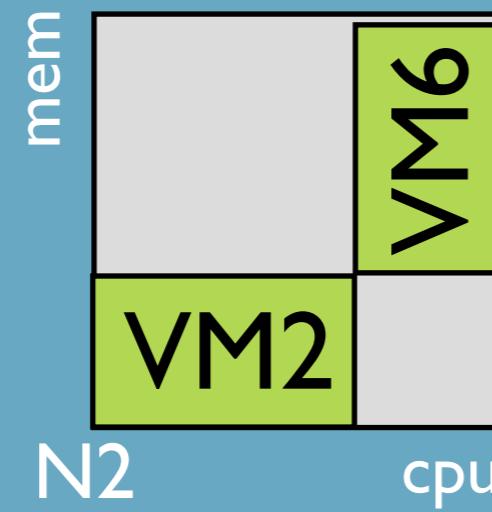
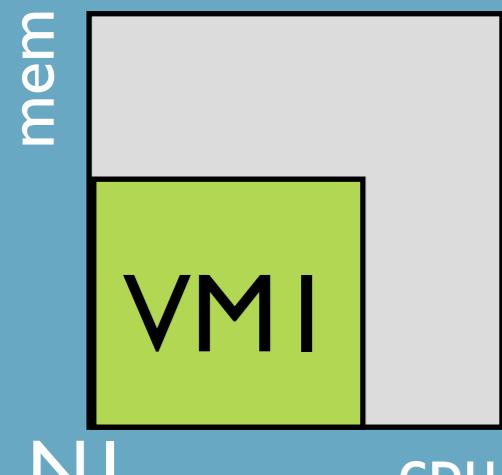
quality at a price



$$\min(\#\text{onlineNodes}) = 3$$

# dynamic schedulers

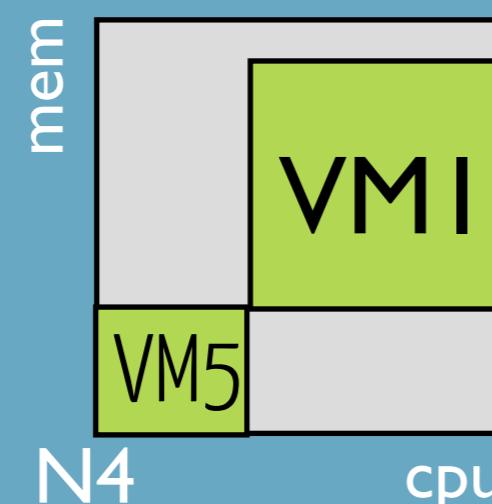
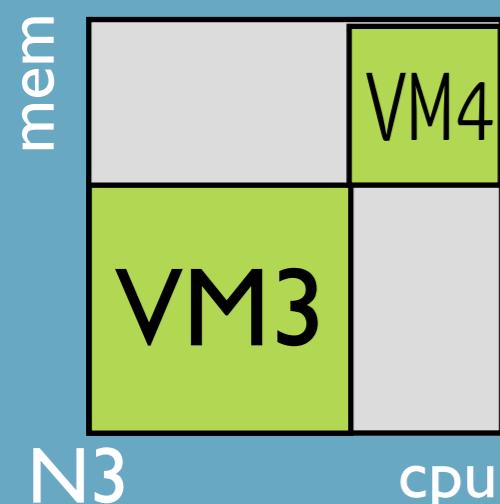
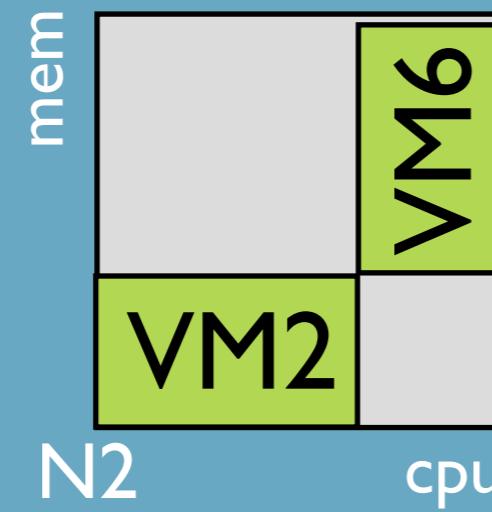
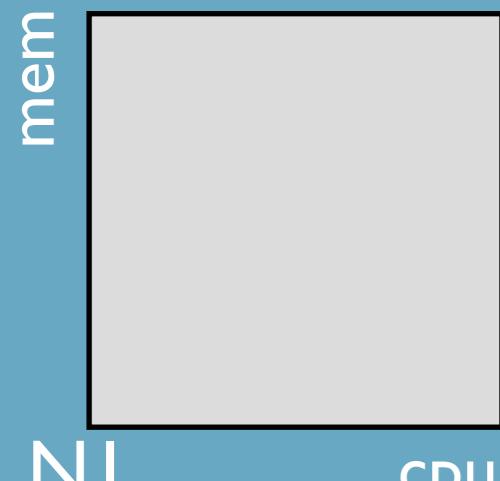
quality at a price



$$\min(\#\text{onlineNodes}) = 3$$

# dynamic schedulers

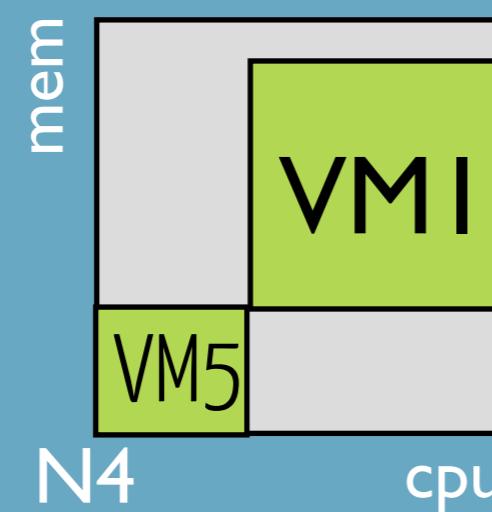
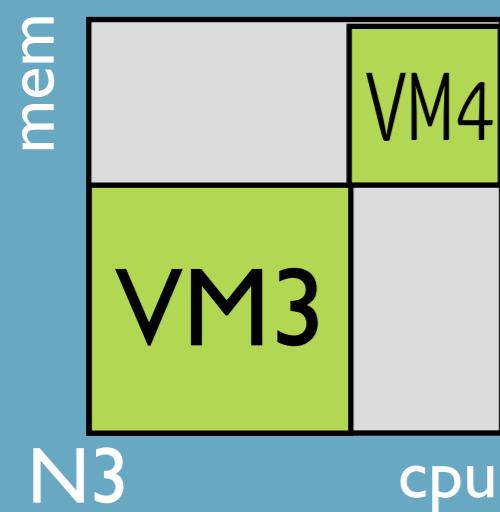
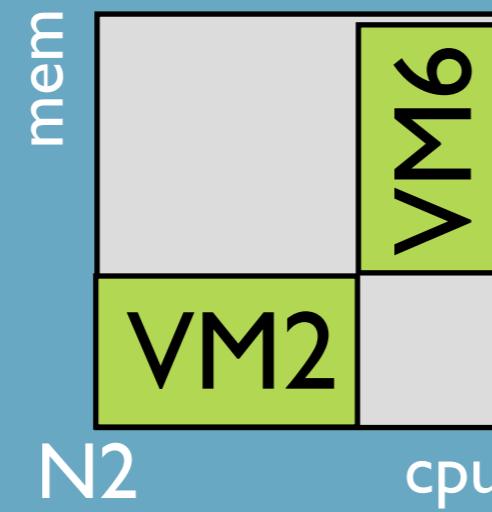
quality at a price



$$\min(\#\text{onlineNodes}) = 3$$

# dynamic schedulers

quality at a price

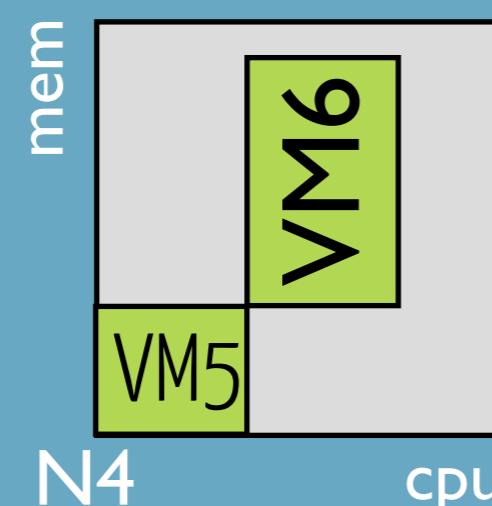
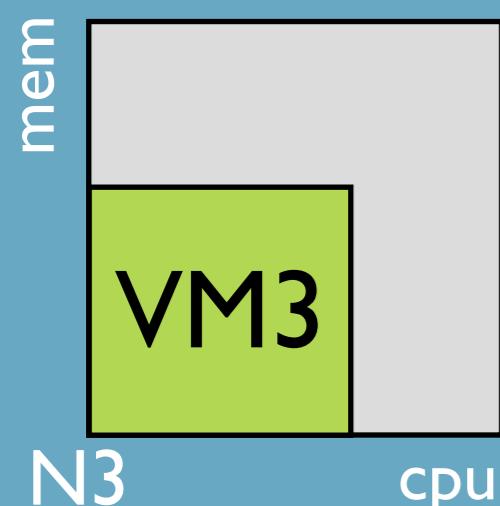
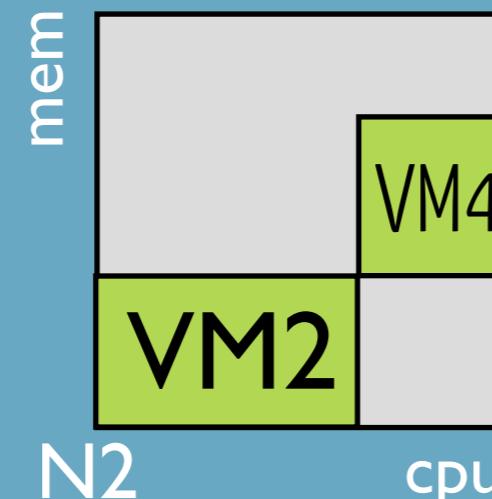
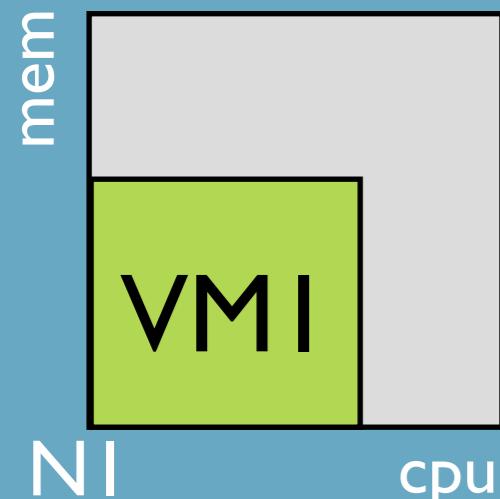


Sol #1: 1m,2m,2m

$$\min(\#\text{onlineNodes}) = 3$$

# dynamic schedulers

quality at a price

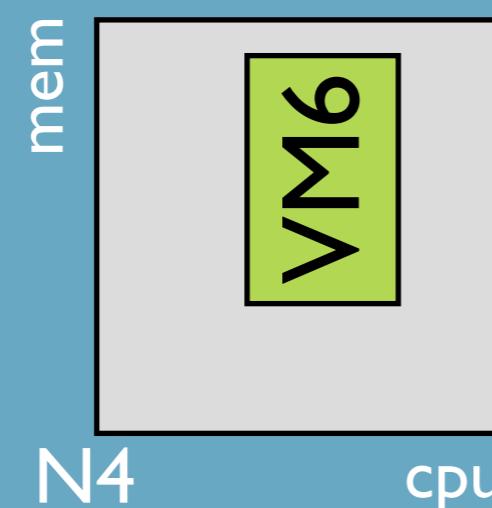
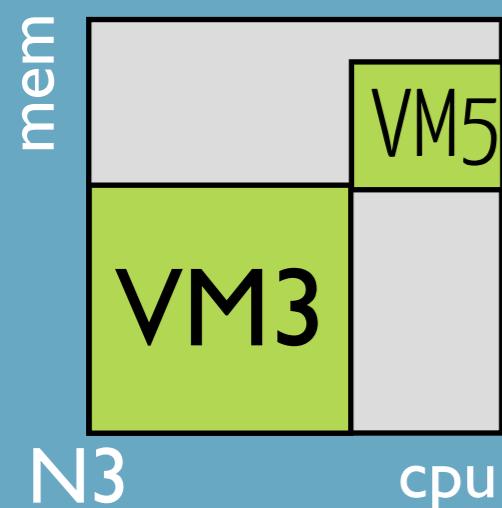
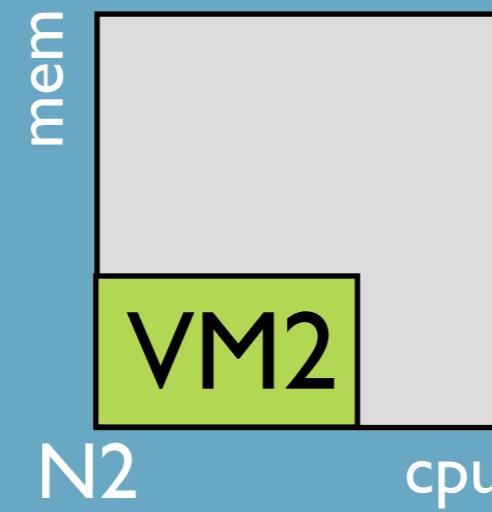
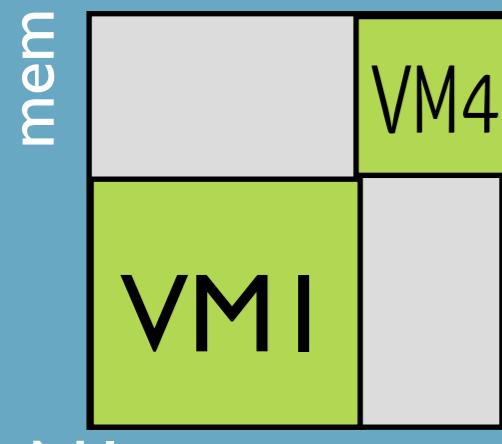


Sol #1: 1m,2m,2m

$$\min(\#\text{onlineNodes}) = 3$$

# dynamic schedulers

quality at a price

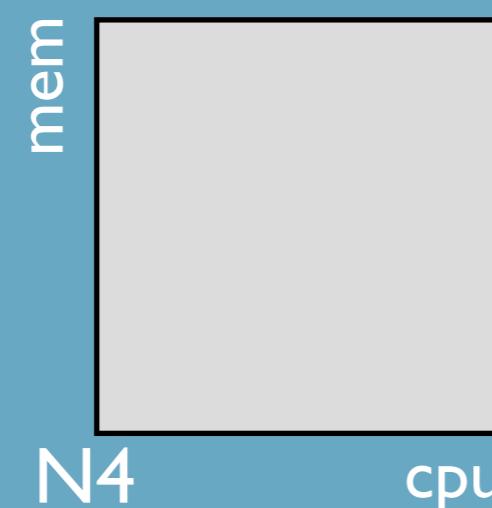
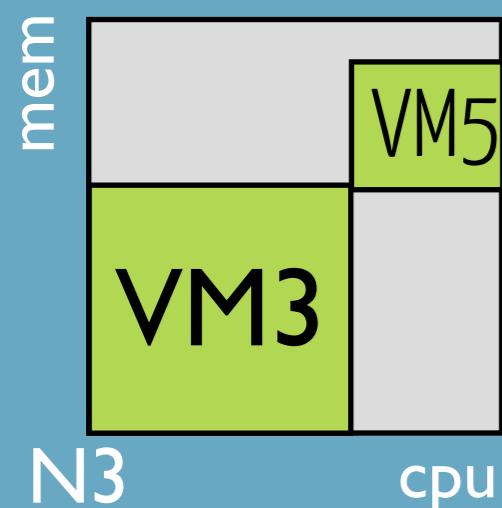
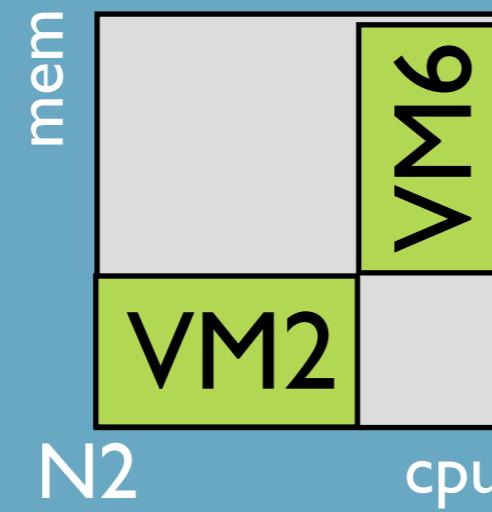
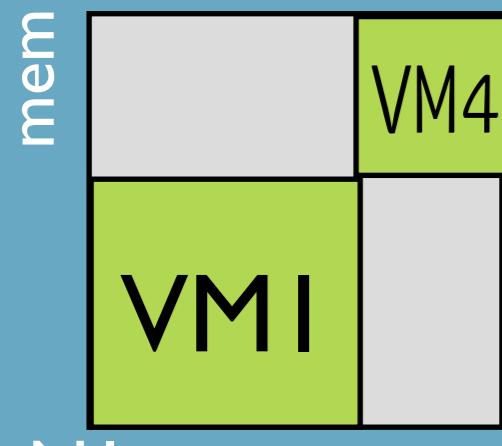


Sol #1: 1m,2m,2m

$$\min(\#\text{onlineNodes}) = 3$$

# dynamic schedulers

quality at a price

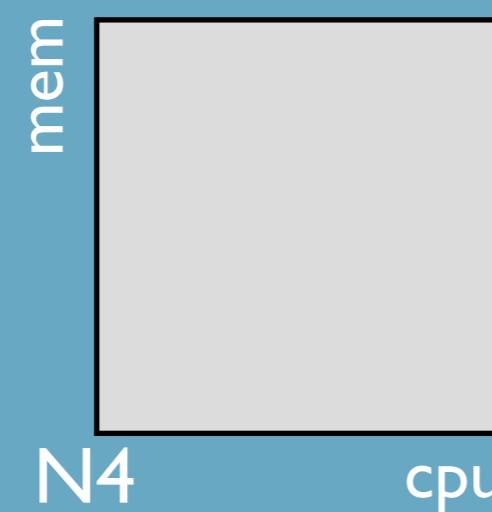
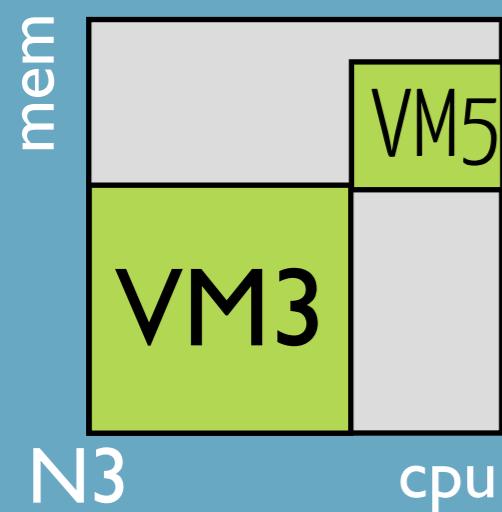
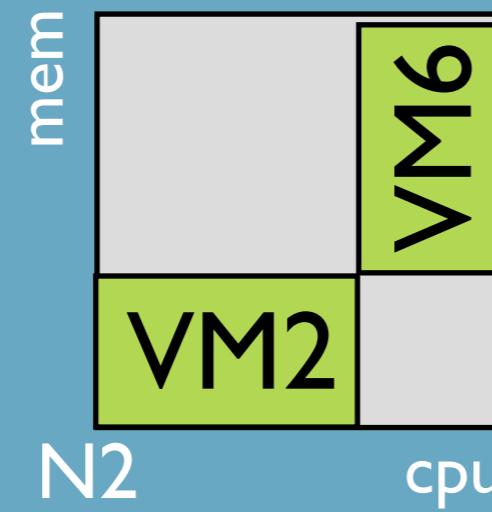
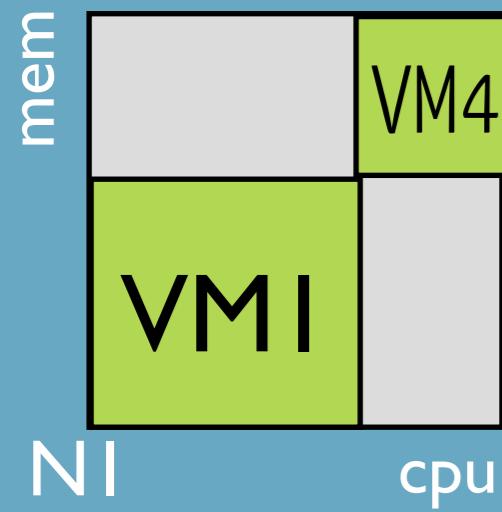


Sol #1: 1m,2m,2m

$$\min(\#\text{onlineNodes}) = 3$$

# dynamic schedulers

quality at a price



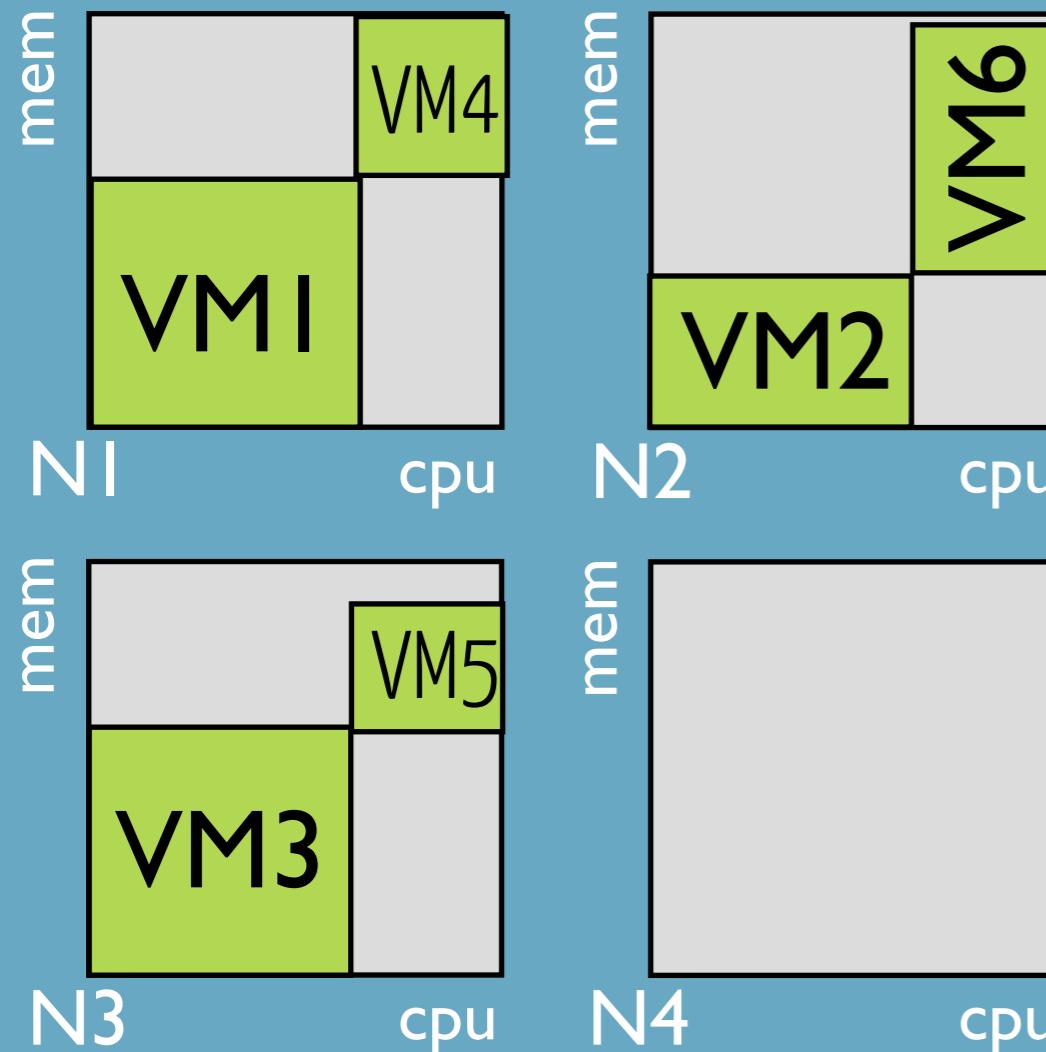
sol #1: 1m,2m,2m

sol #2: 1m,2m  
1m

$$\min(\#\text{onlineNodes}) = 3$$

# dynamic schedulers

quality at a price



sol #1: 1m,2m,2m

sol #2: 1m,2m  
1m

lower MTTR  
(faster)

$$\min(\#\text{onlineNodes}) = 3$$

# static schedulers

consider the VM queue

deployed everywhere

fragmentation issues

# dynamic schedulers

live-migrations to  
address fragmentation

Costly  
(storage, migration latency)

thousands of articles

over-hyped ?

but used in private clouds  
(steady workloads ?)

# Placement constraints

customer or provider side

various concerns

performance, security, power efficiency,  
legal agreements, high-availability,  
fault-tolerance ...

dimension

spatial (placement) or temporal (scheduling)

enforcement level

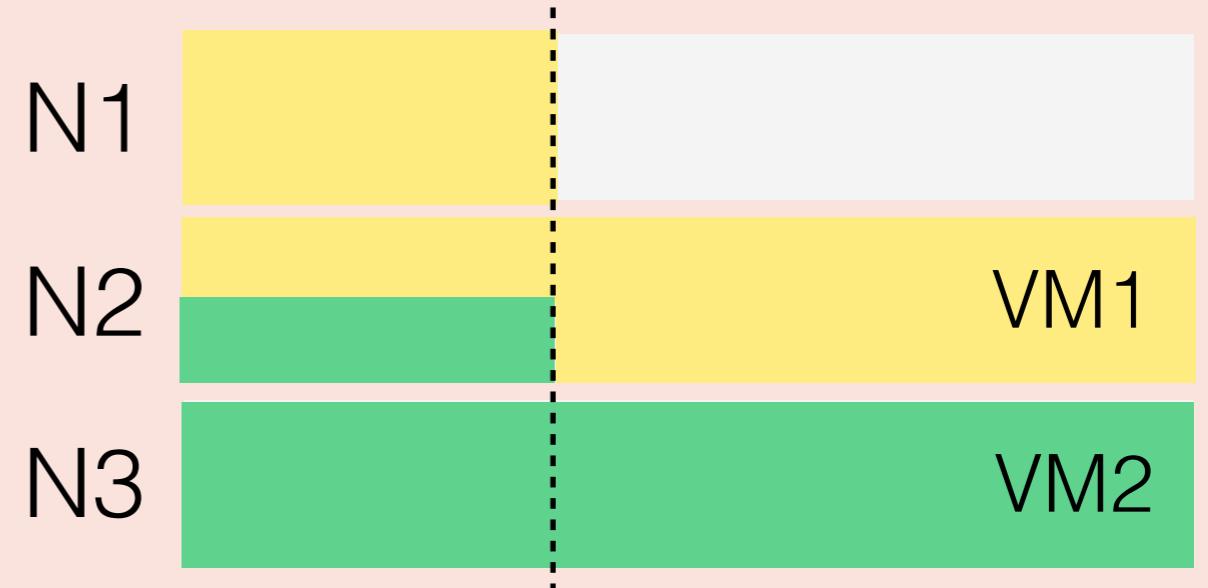
hard or soft  
discrete or continuous

manipulated  
concepts

state, placement, resource allocation,  
action schedule, counters, etc.

# discrete constraints

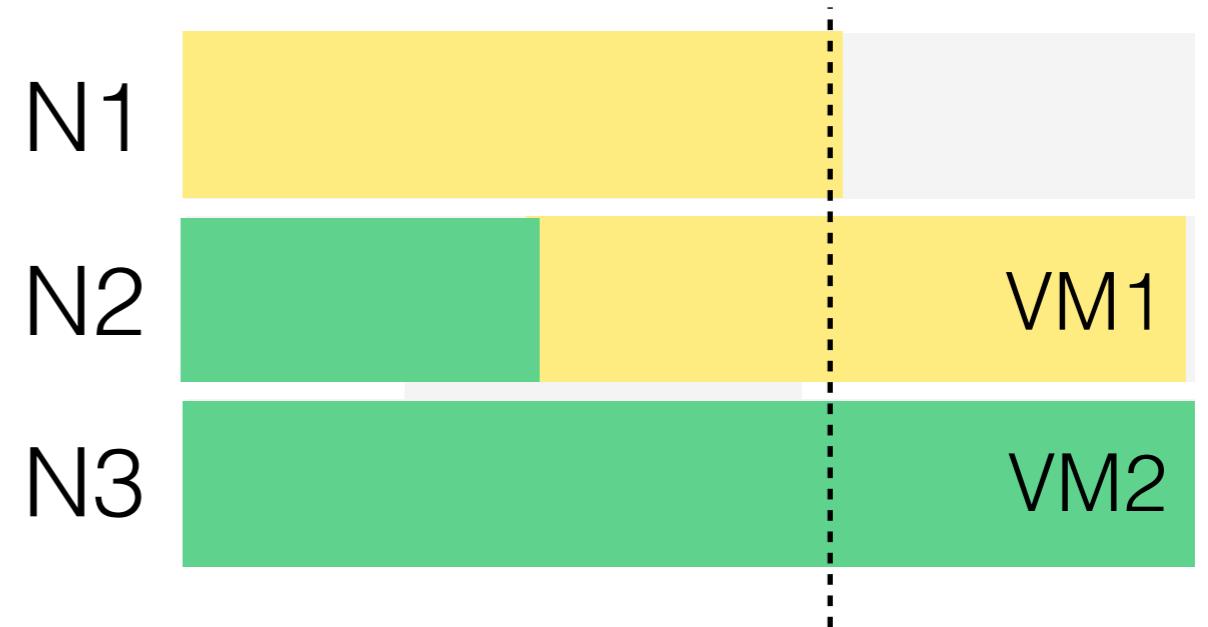
```
>>spread(VM[1,2])  
ban(VM1, N1)  
ban(VM2, N2)
```



“simple” spatial problem

# continuous constraints

```
spread(VM[1,2])  
ban(VM1, N1)  
ban(VM2, N2)
```



harder scheduling problem  
(think about actions interleaving)

# hard constraints

`spread(VM[1..50])`

must be satisfied  
all or nothing approach  
not always meaningful

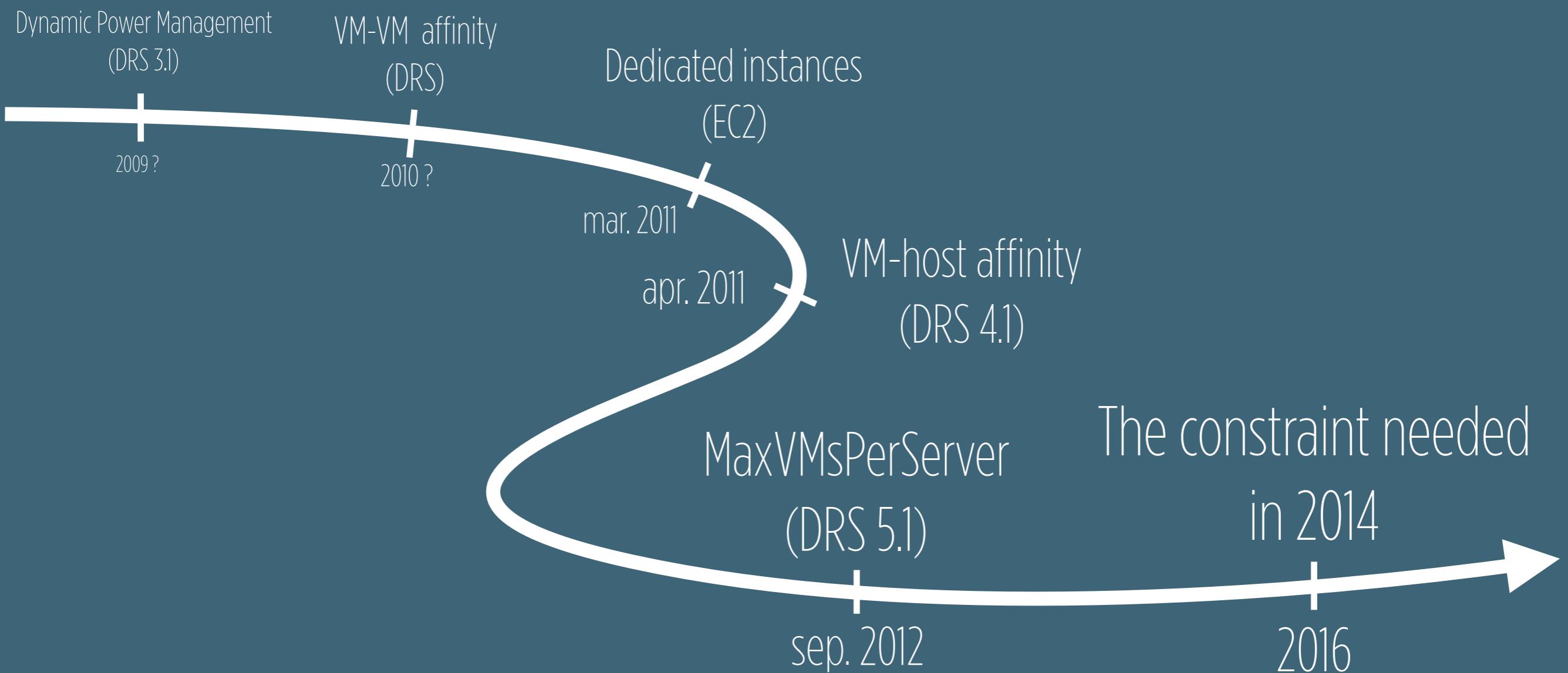
# soft constraints

`mostlySpread(VM[1..50], 4, 6)`

satisfiable or not  
internal or external penalty model

harder to implement/scale  
hard to standardise ?

# The constraint catalog evolves



the  objective

provider side

$\min(x)$  or  $\max(x)$

# atomic objectives

$\min(\text{penalties})$

$\min(\text{Total Cost Ownership})$

$\min(\text{unbalance})$

...

# composite objectives using weights

$$\min(\alpha x + \beta y)$$

How to estimate coefficients ?  
useful to model sth. you don't understand ?

$$\min(\alpha \text{ TCO} + \beta \text{ VIOLATIONS})$$

€ as a common quantifier:

$$\max(\text{REVENUES})$$

# Optimize or satisfy ?

min(...) or max(...)

easy to say

hardly provable

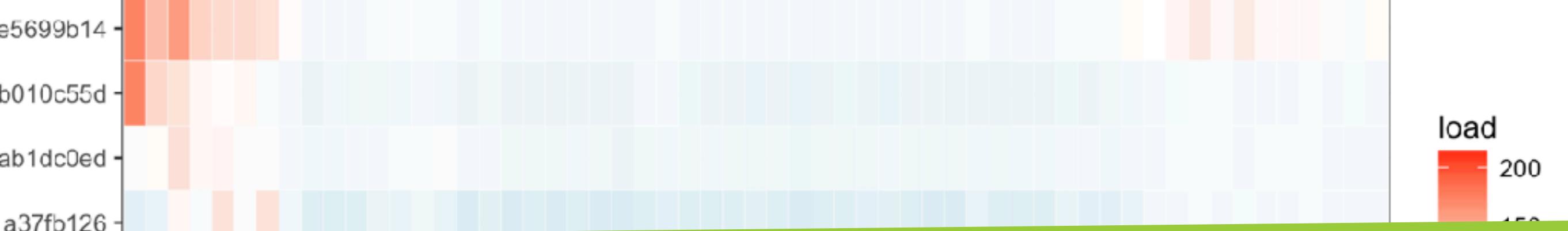
composable through  
weighting magic

threshold based

domain specific expertise

verifiable

composable



# Acropolis Dynamic Scheduler mitigates hotspot

Trigger



Thresholds

85%

CPU  
storage-CPU

Maintain

affinity constraints

Resource demand  
(from machine learning)

Minimize

$\sum$  mig.  
cost

30 sec. timeout

128MB RAM max

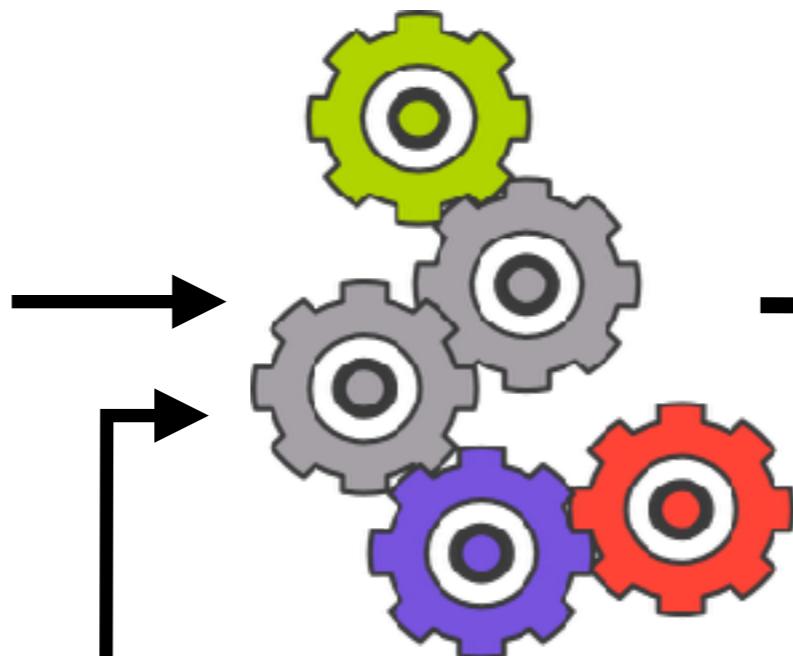
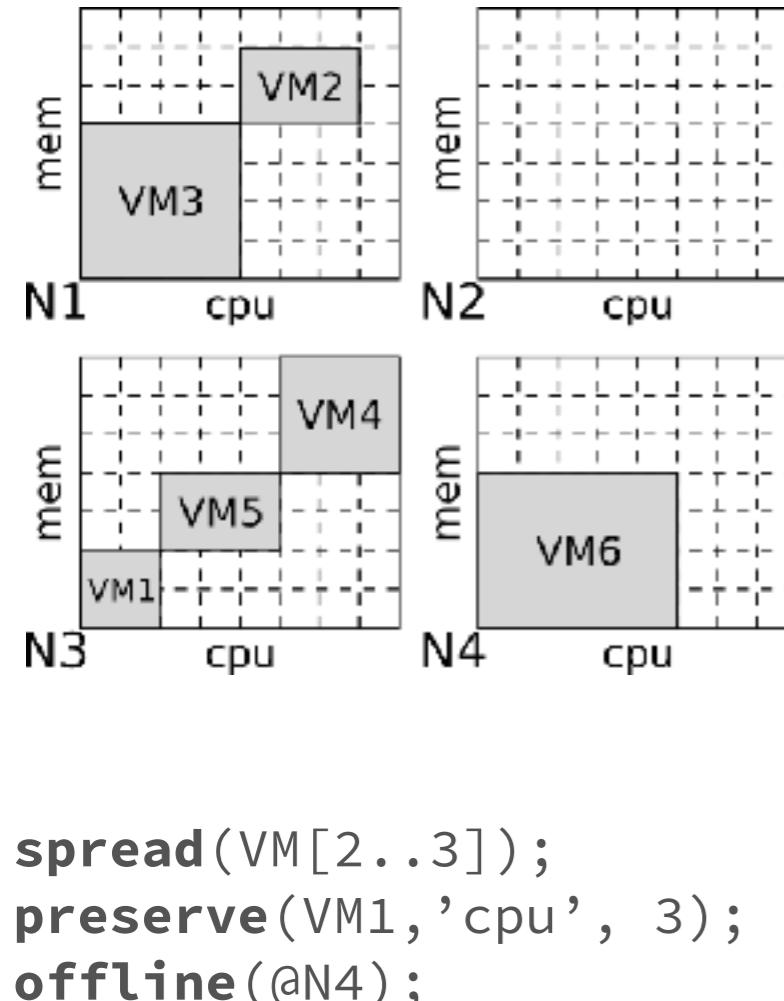


# BtrPlace

adapt the VM placement depending on  
pluggable expectations

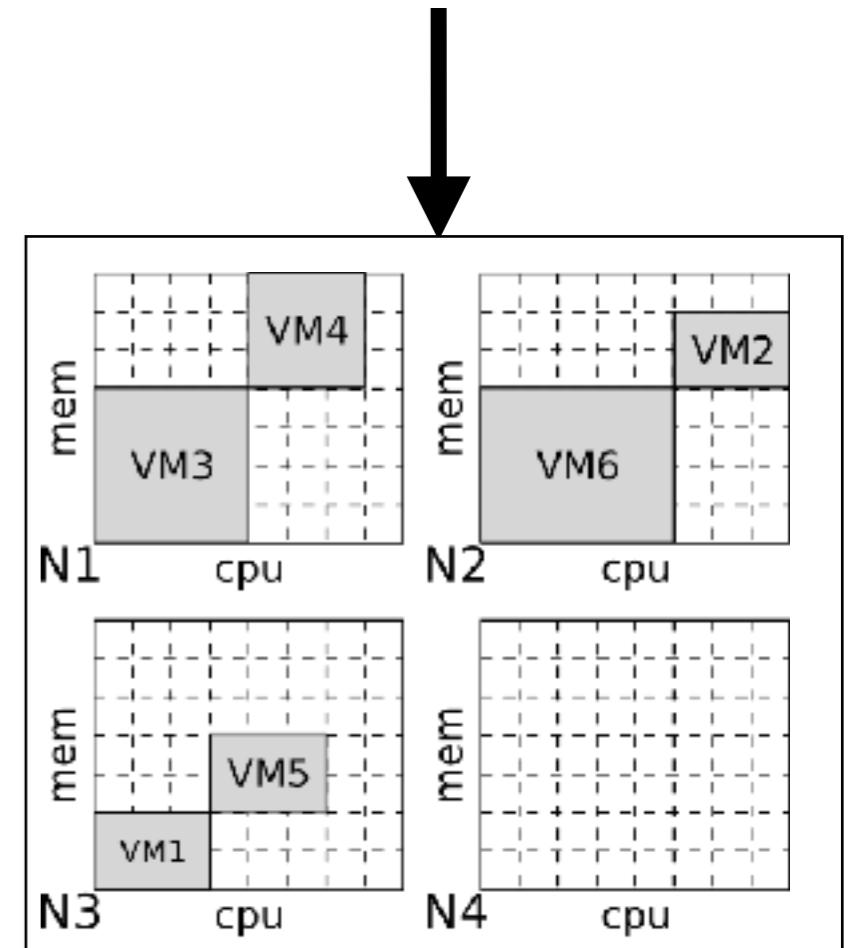
network and memory-aware migration scheduler, VM-(VM|PM) affinities, resource matchmaking, node state manipulation, counter based restrictions, energy efficiency, discrete or continuous restrictions

interaction through a DSL, an API or JSON messages



## The reconfiguration plan

- 0'00 to 0'02: relocate(VM2, N2)
- 0'00 to 0'04: relocate(VM6, N2)
- 0'02 to 0'05: relocate(VM4, N1)
- 0'04 to 0'08: shutdown(N4)
- 0'05 to 0'06: allocate(VM1, 'cpu', 3)

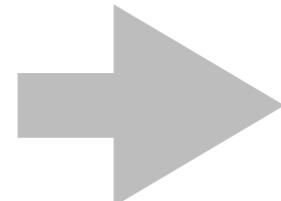


# P.S: From a theoretical to a practical schedule

duration may be longer  
convert to an event based schedule

/!\ possible loss of quality but no alternative yet

0:3 - migrate VM4  
0:3 - migrate VM5  
0:4 - migrate VM2  
3:8 - migrate VM7  
4:8 - shutdown(N2)

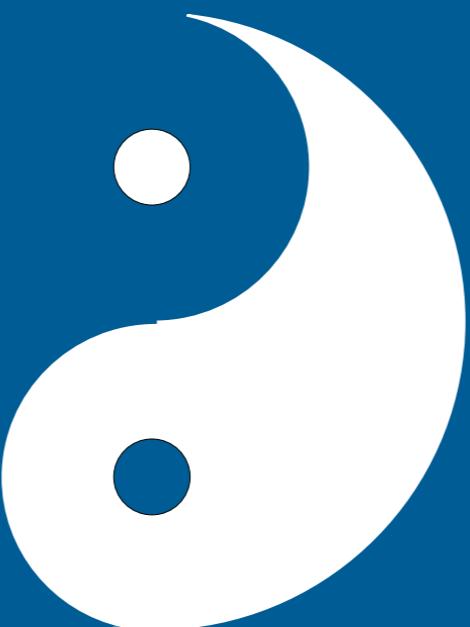


- : migrate VM4  
- : migrate VM5  
- : migrate VM2  
!migrate(VM2) & !migrate(VM4): shutdown(N2)  
!migrate(VM5): migrate VM7



An Open-Source java library  
for constraint programming

deterministic composition  
high-level constraints



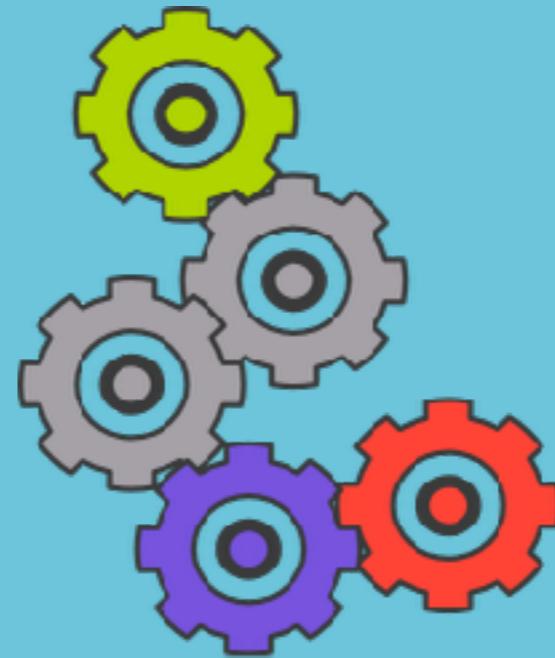
the right model  
for the right problem

$$\begin{aligned}\mathcal{X} &= \{x_1, x_2, x_3\} \\ \mathcal{D}(x_i) &= [0, 2], \forall x_i \in \mathcal{X} \\ \mathcal{C} &= \left\{ \begin{array}{l} c_1 : x_1 < x_2 \\ c_2 : x_1 + x_2 \geq 2 \\ c_3 : x_1 < x_3 \end{array} \right.\end{aligned}$$

# Entropy

<http://entropy.gforge.inria.fr>

2006



# BtrPlace

2011

2017

From how it worked to how it works  
Consequences of trying to understand

2006 - 2009



<http://entropy.gforge.inria.fr>

No knowledge in CP: Collaboration with the Nantes team

2 phases approach: 1 problem for the placement (VMPP),  
1 for the “schedule” (VMRP)

(Faster & better than 1 phase + weighting magic)

[vee 2009]

2006 - 2009



<http://entropy.gforge.inria.fr>

## The placement problem (VMPP)

VMs are idle or burning the CPUs

Old-school bin-packing

$H_i$  bit vectors + bool\_channelling

$$\mathcal{R}_p \cdot H_i \leq \mathcal{C}_p(n_i) \quad \forall n_i \in \mathcal{N}$$

$$\mathcal{R}_m \cdot H_i \leq \mathcal{C}_m(n_i) \quad \forall n_i \in \mathcal{N}$$

Multi-knapsack constraint + dynamic programming  
(Thanks Xavier Lorca & Hadrien Cambazard)

Symmetry breaking for items not fitting

Minimize #nodes using atmost\_nvalue  
(green computing hype)

2006 - 2009



<http://entropy.gforge.inria.fr>

## The replacement problem (VMRP)

VMPP variation:

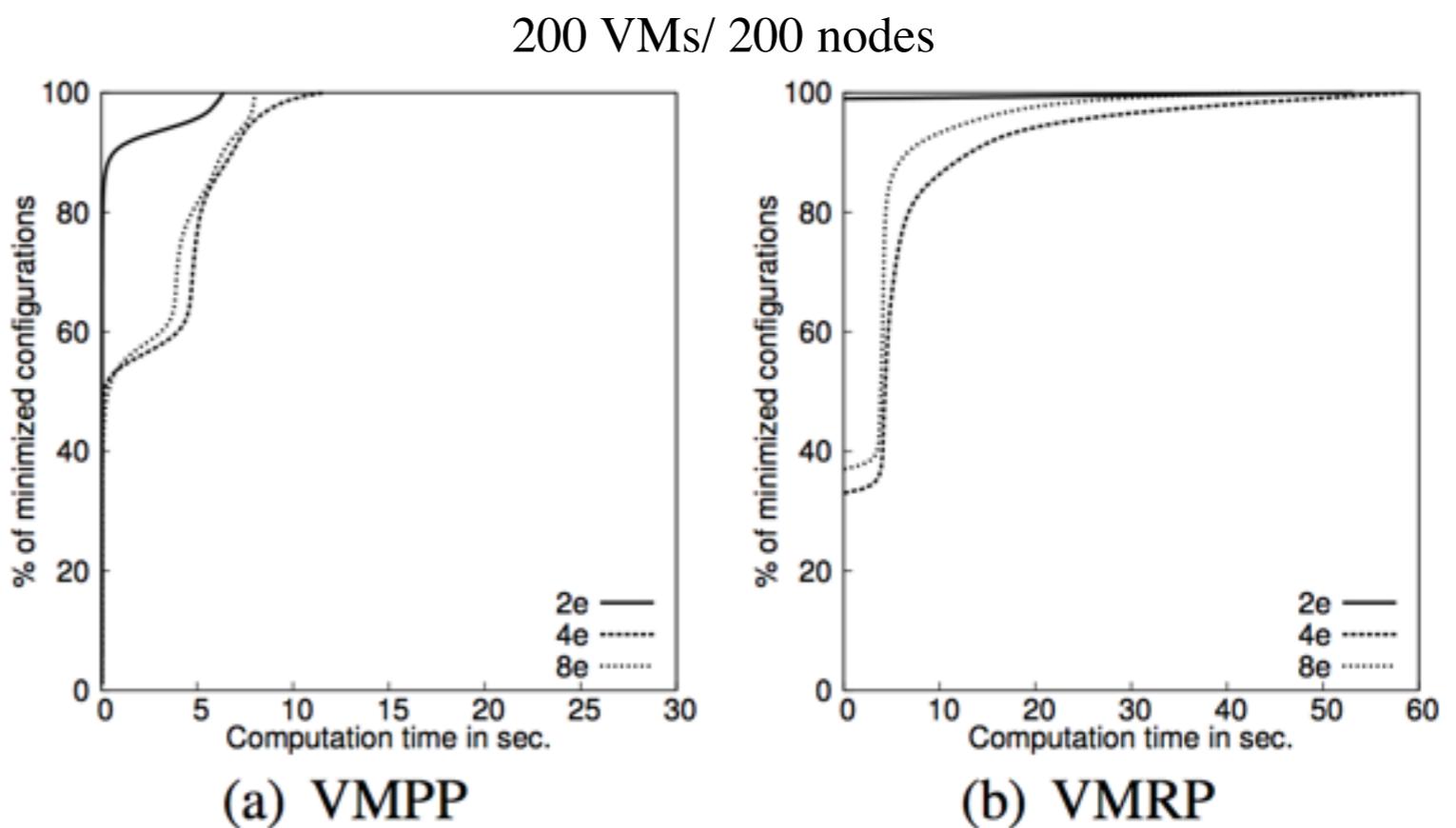
atmost\_nvalue, #nodes as a constant

migration\_duration: [0, K], 0 iff stays

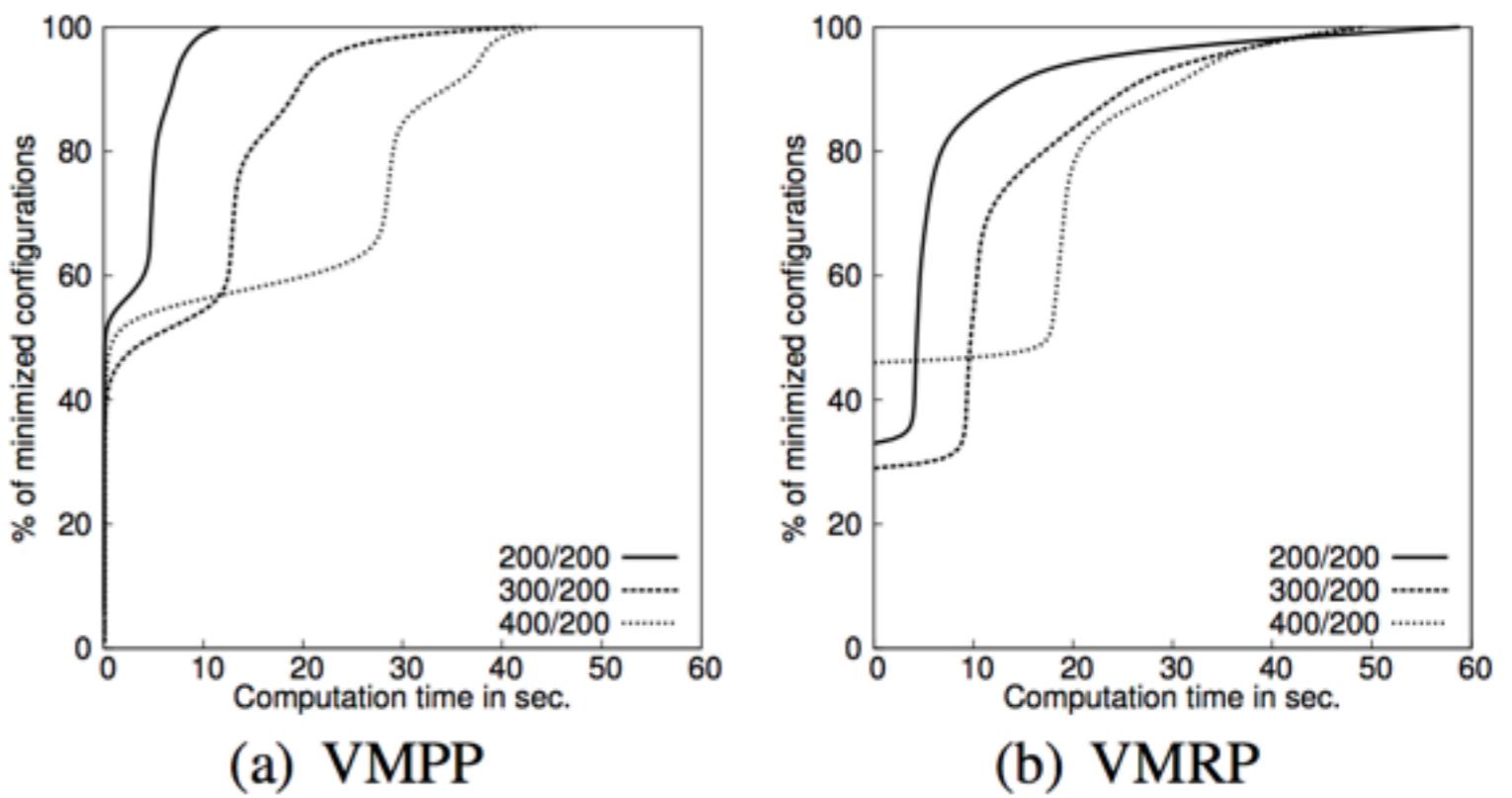
dependency\_management & cost function:  
hard coded inside a propagator

min (sum of the end moment of migrations)

Symmetry breaking  
50% faster



Global perf  
2 minutes for a local opt.



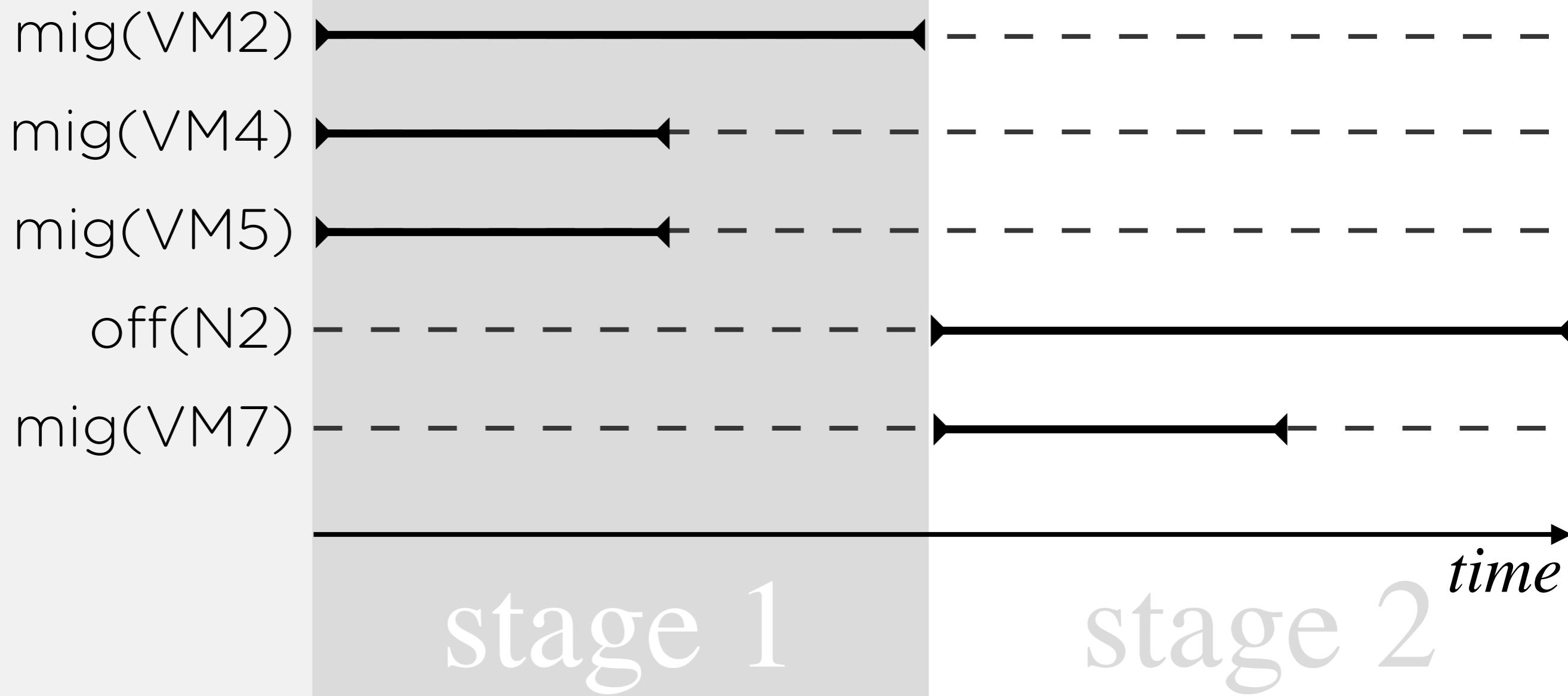
# Reviewing Entropy design

Scheduling heuristic prevents control  
hard-coded but vital heuristic inside a propagator  
not optimisation friendly

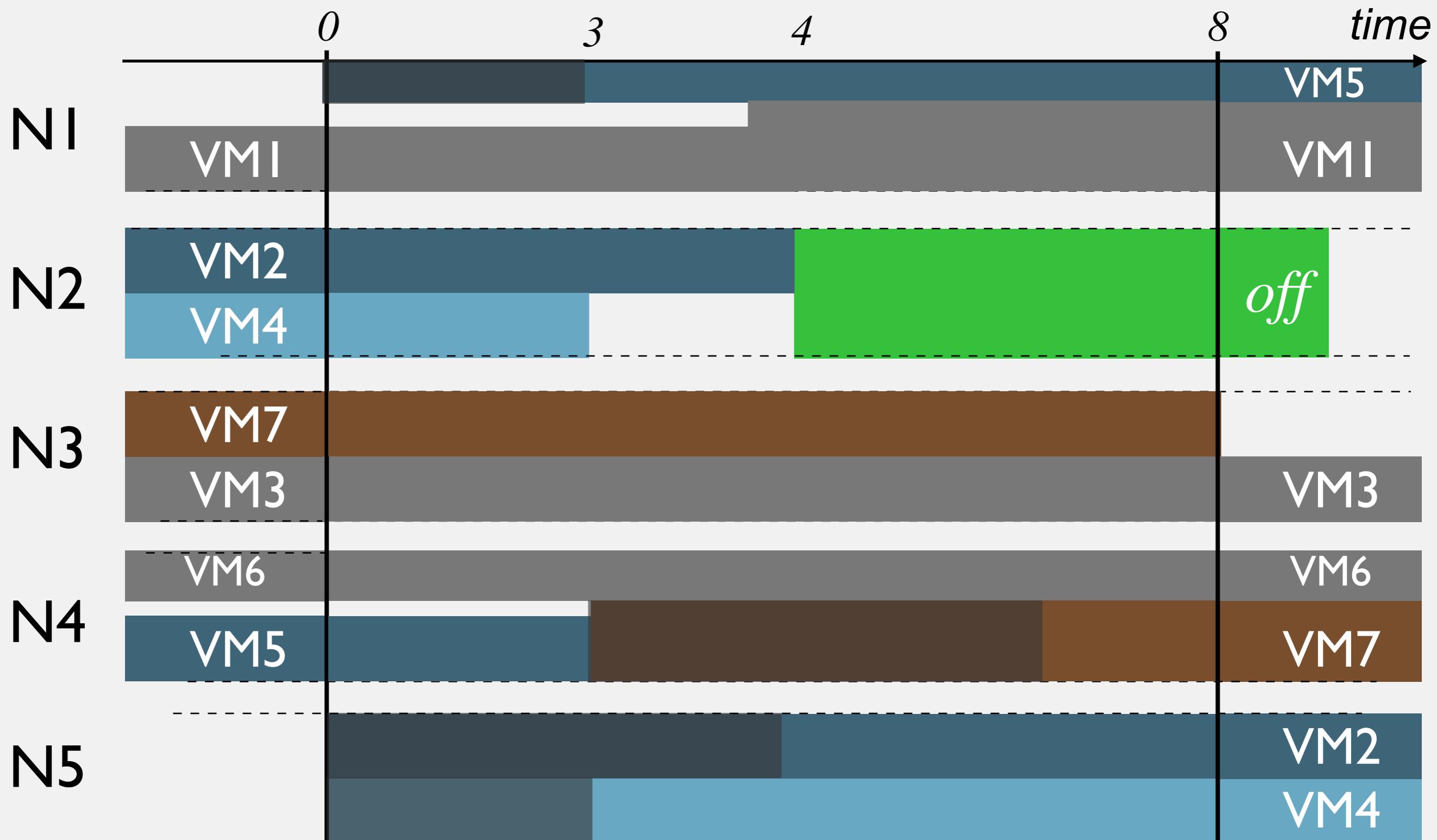
Scalability issue  
model-level:  $n*m$  binary variables, 5k VMs/100 servers: 500k bool vars  
memory level: see above (4GB RAM in 2006)

Objective issue  
No one really cares about  $\min(\#nodes)$   
 $\min(\#migration)$  dominates

# coarse grain staging delay actions



# 2010+ : let schedule properly



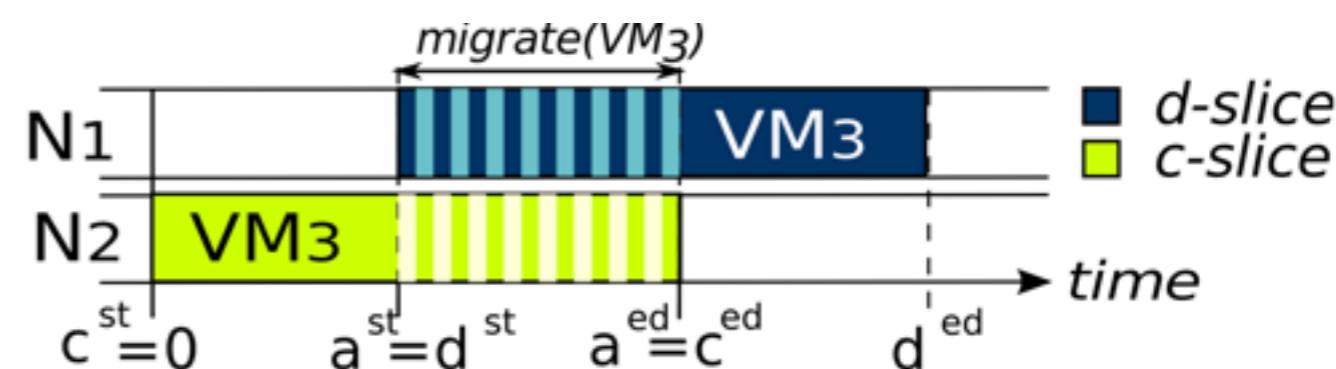
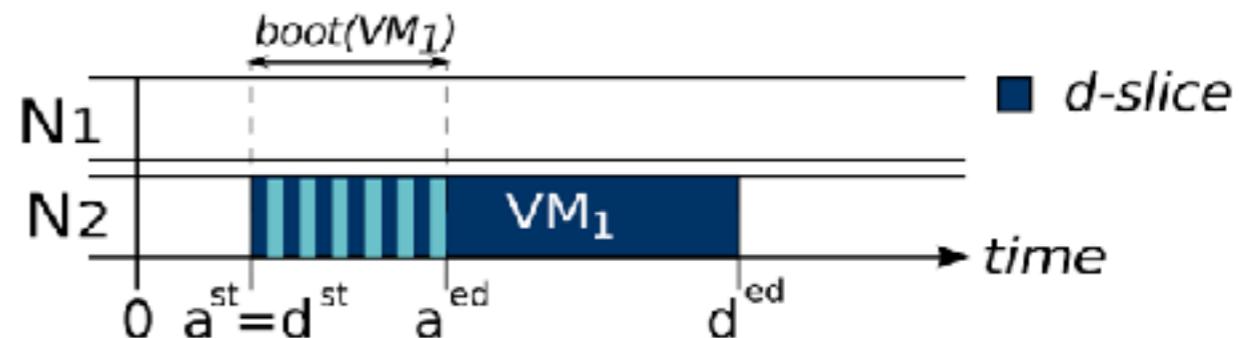
# Resource-Constrained Project Scheduling Problem

Tasks to model resource consumption

Every action modelled from its impact over resource

Booting a VM:  
Place a task on the node

Migrating a VM:  
2 tasks (source and destination)  
Tasks co-locate if the VM stays  
(impact the overlapping period)



# Original implementation

Using Choco “cumulativeS” constraint (1 per resource)  
(Thanks Arnaud Malapert)

Regular cumulative constraints + binary variables to state  
if the tasks is in.

Conditional non-overlapping modeled  
outside the constraints



No more cyclic dependencies

Prevent the situation in place of fixing it. Should not happen anyway

Enough perf for a Poc. Not enough for being good enough  
First implementation falls with more than 10 nodes

# From cumulativeS to *myCumulative*

Problem: overkill filtering

2 kind of tasks: starts at 0 (c-task) or ends at makespan (d-task)

Problem particularities:

A VM migration is modelled using 1 c- and 1 d-task

no overlap when there is no migration

Moving to a home-made implementation

(thanks Sophie Demassey)

active only once the items are placed

# From myCumulative to myCumulative 2

Booting || halting the node:  
No longer modelled using tasks

First implementation by my own  
hackie, incorrect, big fail

Second implementation by Sophie Demassey  
big win

Quick improvement by my side  
semi-win: bug found 2 years later (over-filtering)

# 2011+ : let pack properly

From 1 bin packing per dimension  
to one xD vector packing

Originally:

- same filtering but less events, less memory
- get rid of the knapsack filtering from profiling observation
- no more bool vector. Useless.

Now:

- knapsack is back again (see later).
- start to consider cardinality dimension as a pivot

Un-effective tryout: static or dynamic big-items

# BtrPlace becomes a CP solver dedicated to VM management

A primary problem to model a reconfiguration  
Extensions to bring additional concerns  
Constraints to manipulate

$$\begin{aligned}
boot(v \in V) &\triangleq & D(v) &\in \mathbb{N} \\
&& st(v) &= [0, H - D(v)] \\
&& ed(v) &= st(v) + D(v) \\
&& d(v) &= ed(v) - st(v) \\
&& d(v) &= D(v) \\
&& ed(v) &< H \\
&& d(v) &< H \\
&& h(v) &\in \{0, \dots, |N| - 1\}
\end{aligned}$$

# BtrPlace core CSP

models a reconfiguration plan  
 1 model of transition per element  
 action durations as constants \*

$$\begin{aligned}
relocatable(v \in V) &\triangleq \dots \\
shutdown(v \in V) &\triangleq \dots \\
suspend(v \in V) &\triangleq \dots \\
resume(v \in V) &\triangleq \dots \\
kill(v \in V) &\triangleq \dots \\
bootable(n \in N) &\triangleq \dots \\
haltable(n \in N) &\triangleq \dots
\end{aligned}$$

Views bring additional concerns  
new variables and relations

**ShareableResource(r) ::=**

$$\forall n \in \mathcal{N}, \sum_{v \in \mathcal{V}, host(v)=n} cons(v, r) \leq capa(n, r)$$

**Network() ::= ...**

**Power() ::= ...**

**High-Availability() ::= ...**

$$spread(X \subseteq \mathcal{V}) \triangleq \forall(a, b) \in X, host(a) \neq host(b)$$
$$lonely(X \subseteq \mathcal{V}) \triangleq \bigcup_{v \in X} host(v) \not\subseteq \bigcup_{v \in \mathcal{V} \setminus X} host(v)$$

...

## Constraints state new relations

Constraint	Loc.	Constraint	Loc.	Constraint	Loc.
spread	50	root	11	preserve	10
among	40	lonely	17	overSubscription	40
ban	20	quarantine	40	offline	10
fence	58	capacity	64	noIdles	10
gather	11	splitAmong	31		

CP expressivity leads to concise code



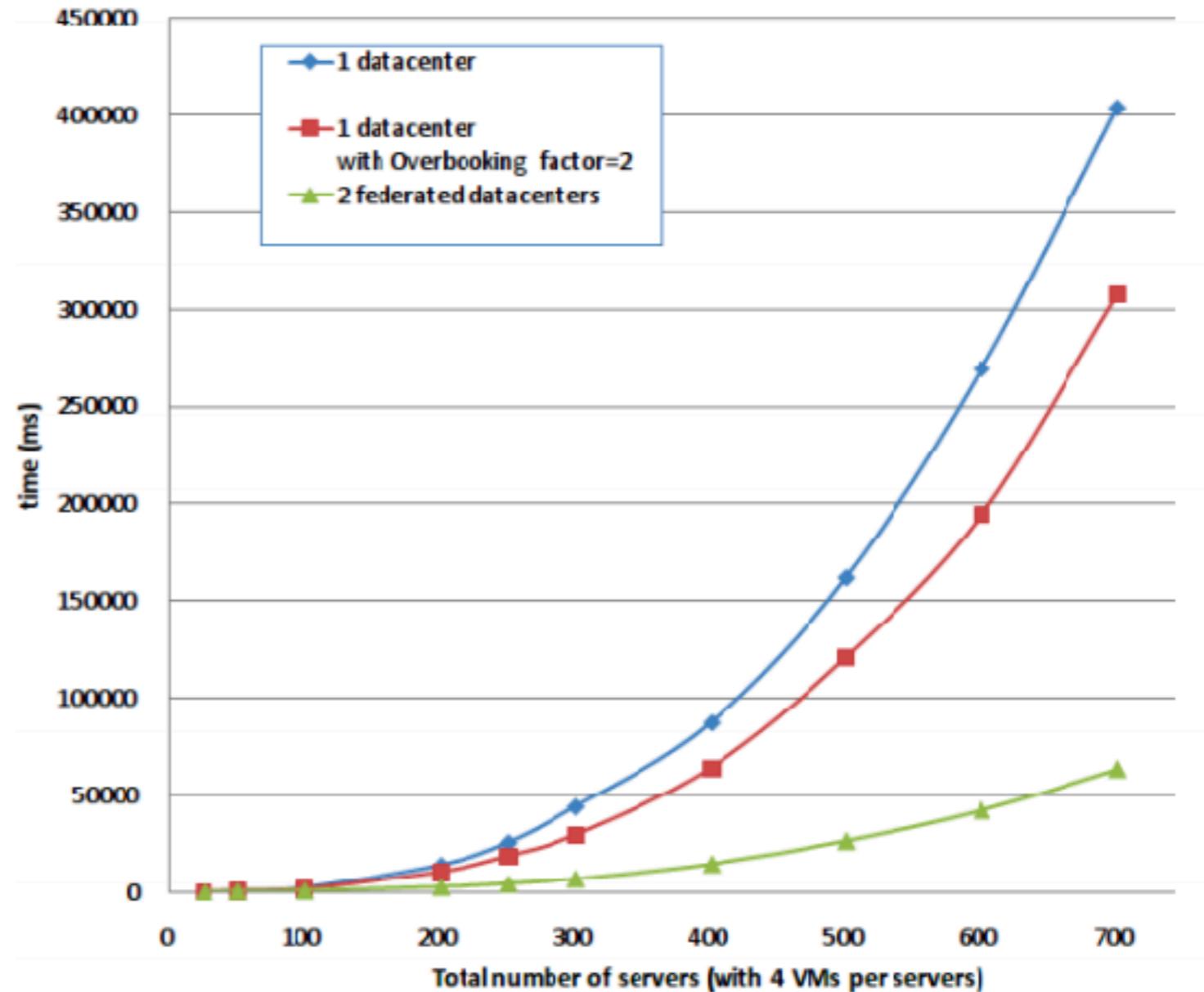
2009 - 2011

FIT4green aims at contributing to ICT energy reducing efforts by creating a energy-aware layer of plug-ins for data centre automation frameworks.

2010: switch from their heuristic to Entropy/BtrPlace.  
homemade (very fine grain) objective  
Involved as a contributor

[e-energy 12]

2.8k VMs, 700 servers  
400 sec. to the 1st solution



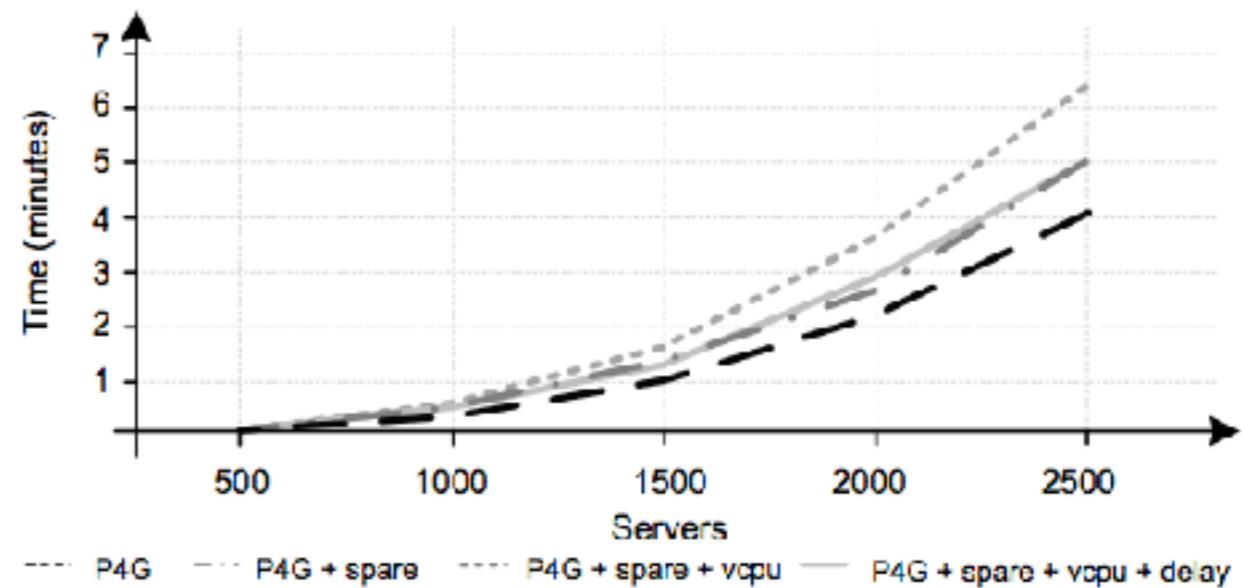
“Houston (Fabien), we got a problem, CP is sloowww



ej-TECHNOLOGIES'  
**JProfiler**

95%+ in search heuristic  
no value gettable O(1), O(n) at least

One cache later:  
from 400 to 20 sec.  
[adhoc network 12]



## Lesson learned

- 1/ CP was not slow, they did not try to understand the issue
- 2/ The fix is not good/disruptive/innovative. Normal due to the context
- 3/ Their opinion did not change as they forgot about the fix.

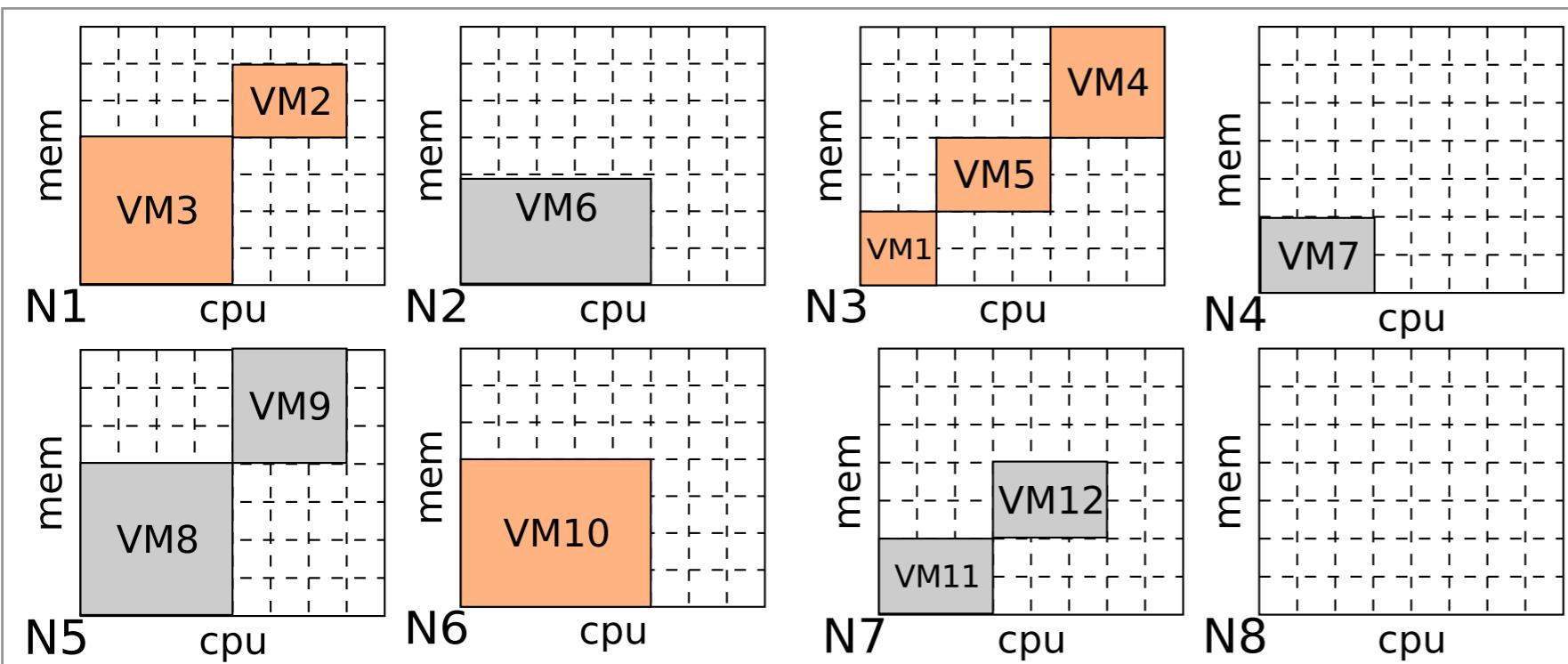
# 2011: better knowledge of the problem

There is no anarchy in a data center  
load spike, failures: local issue

The data center is not a single space  
Technical limitations create *autonomous regions*

Static instance analysis to the rescue

# static model analysis 101



```
spread({VM3,VM2,VM8});  
lonely({VM7});  
preserve({VM1}, 'ucpu', 3);  
offline(@N6);  
ban($ALL_VMS, @N8);  
fence(VM[1..7], @N[1..4]);  
fence(VM[8..12], @N[5..8]);
```

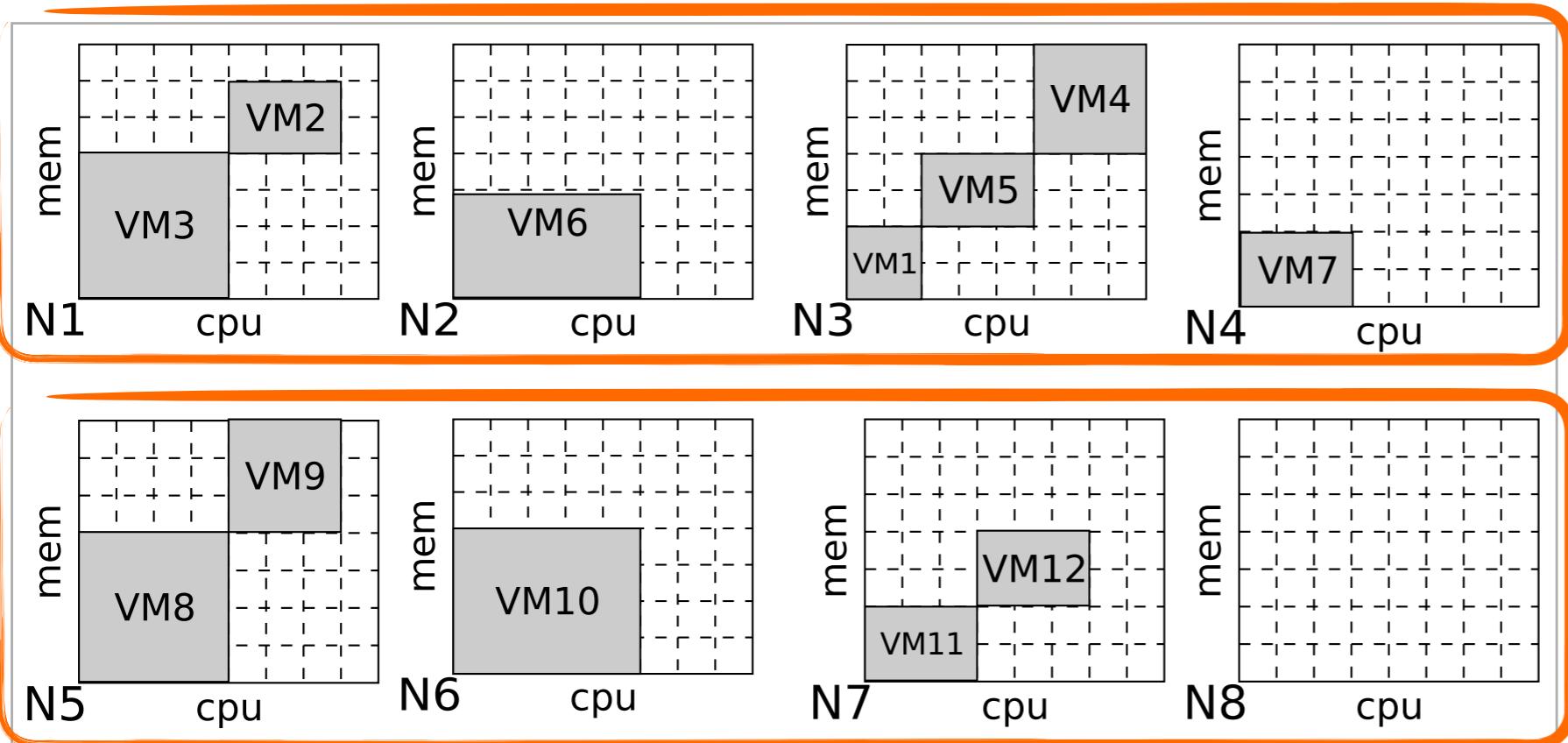
`scheduler.doRepair(true)`

manage only supposed mis-placed VMs

Pre-place “well placed VM”

beware of under estimations !

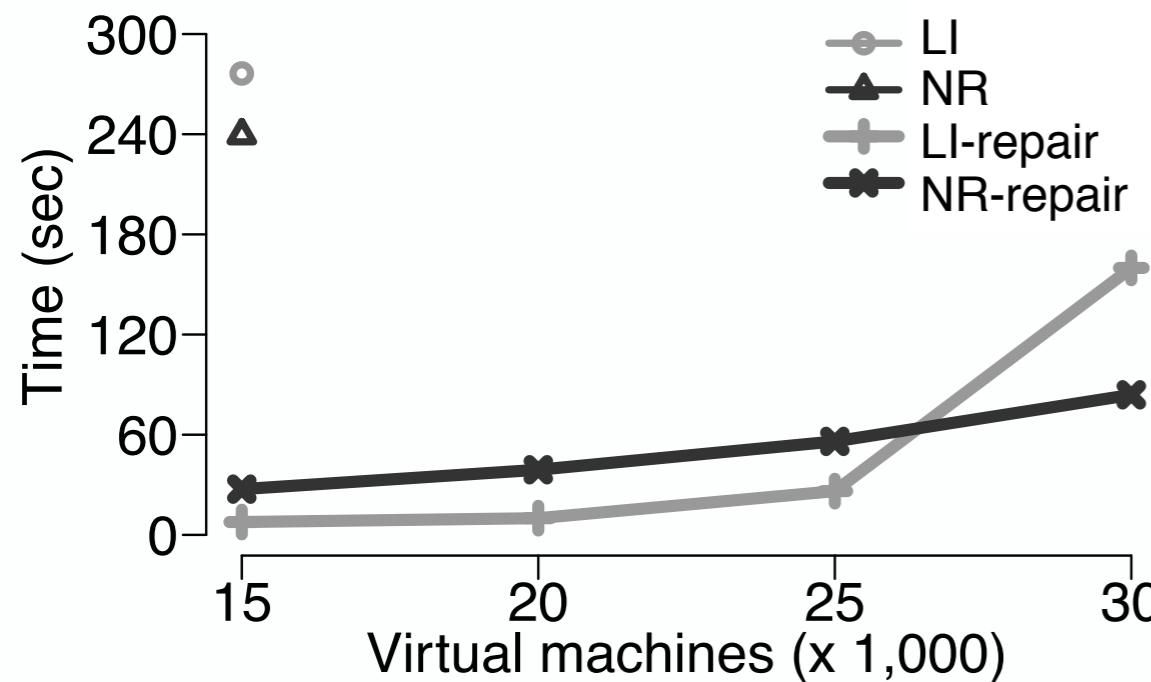
# static model analysis 201



```
spread({VM3,VM2,VM8});  
lonely({VM7});  
preserve({VM1}, 'ucpu', 3);  
offline(@N6);  
ban($ALL_VMS, @N8);  
fence(VM[1..7],@N[1..4]);  
fence(VM[8..12],@N[5..8]);
```

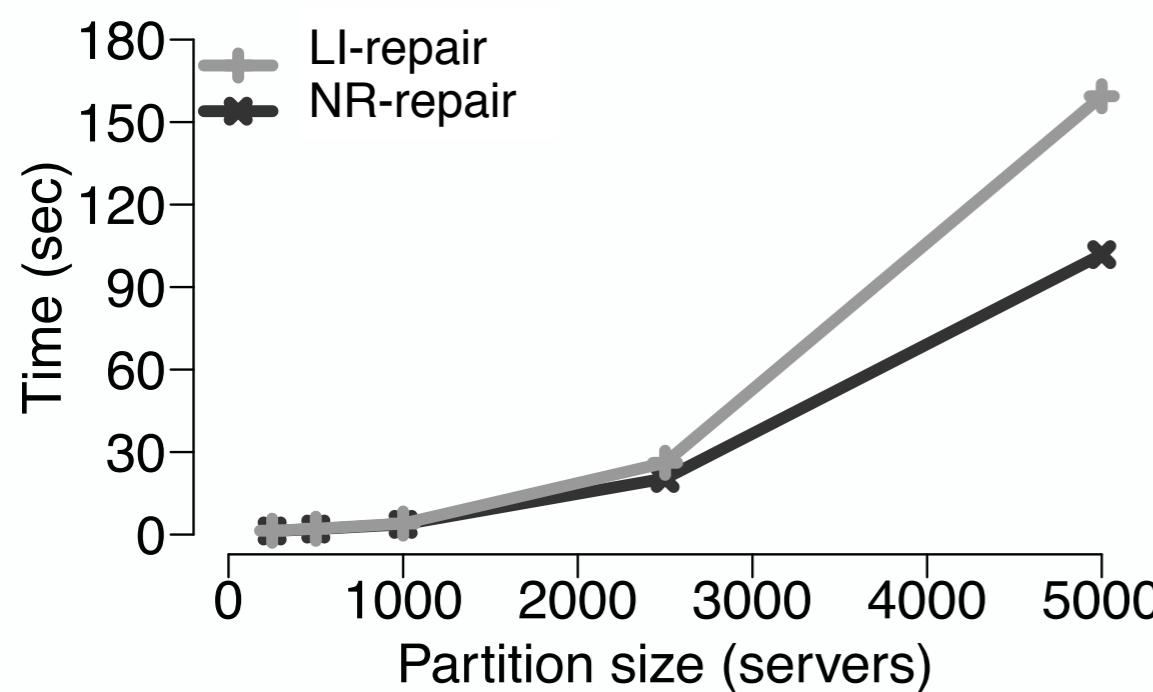
```
s.setInstanceSolver(  
    new StaticPartitioning())
```

independent sub-problems solved in parallel  
beware of resource fragmentation !



## Repair benefits

- 5 times less VMs to consider
- 10 times faster at least
- no impact on the solution quality



## Partitioning benefits

- 2 times smaller, 4 times faster
- no impact on the solution quality

# 2011: The hype of set variables (Better knowledge of the model & the solver)

## In choco bin packing

In the user-API to state the items in every bin

Embed a channeling to the placement variables

## In side constraints

Lonely: 2 sets of VMs must be on disjoint set of nodes.

using the *set\_disjoint* constraint

From 2 to 6k VMs on 1k servers

Table 1: Impact of the consolidation ratio on the solving process.

Ratio	Rebuild Mode					Repair Mode				
	solved	obj	nodes	fails	time	solved	obj	nodes	fails	time
2:1	100	452	2034	352	42.2	100	381	163	0	3.5
3:1	94	1264	3119	3645	75.2	100	749	394	0	8.4
4:1	65	3213	4574	11476	129.3	100	1349	836	0	18.7
5:1	10	7475	6878	47590	241.2	100	2312	1585	44	37.7
6:1	0	-	-	-	-	86	4092	2884	2863	71.5

From 2k5 VMs/500 servers to 10kVMs/2k servers

Table 2: Impact of the datacenter size on the solving process (repair mode).

Set	#servers	#VMs	solved	obj	nodes	fails	time
x1	500	2,500	100	1160	805	13	7.0
x2	1,000	5,000	99	2321	1594	17	36.2
x3	1,500	7,500	99	3476	2374	43	105.5
x4	2,000	10,000	100	4635	3171	15	217.0

# [RR report: “CP 2011” back to work]

## In bin packing

set variables used to iterate only. Overkill  
switch to internal bitsets + channelling with the  
placement variables

## In side constraints

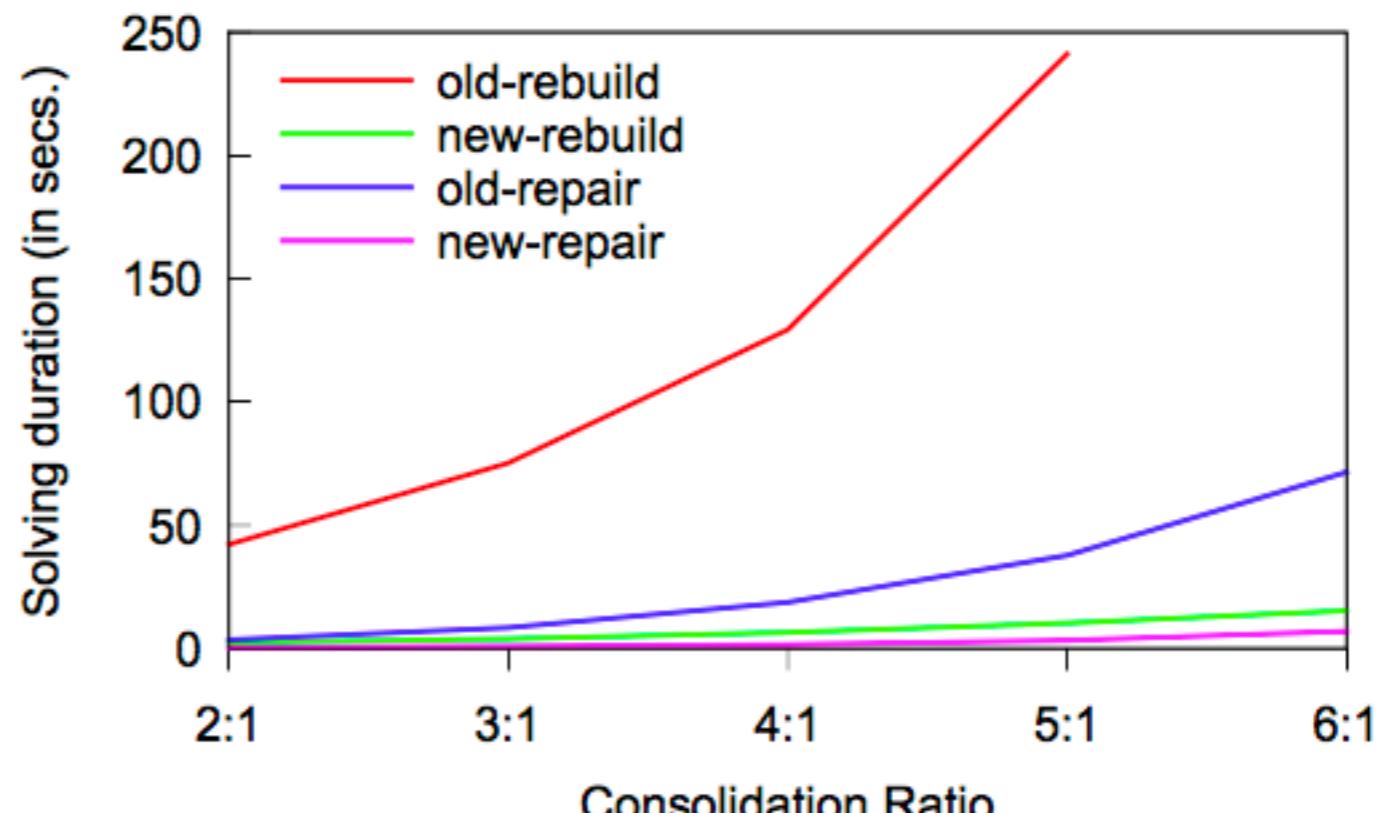
set variables are overkill as the list of VMs is known  
switch to 2 list of placement variables

## Side change in a heuristic

Fix a bug that ignored the first-fail

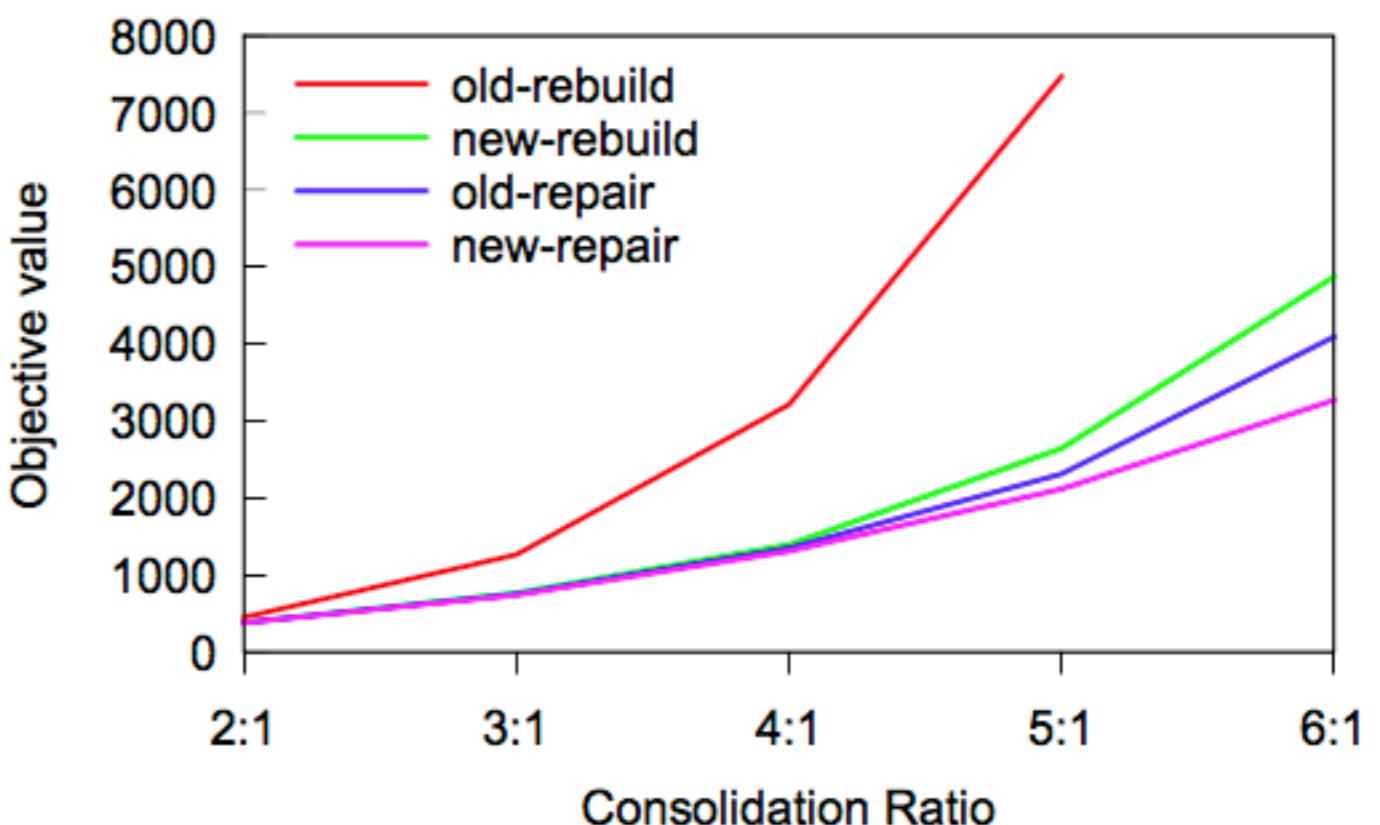
Improve the scheduling heuristic to branch  
on VMs going to leaf node

Up to a x20 speedup  
No more backtracks  
At least 1 sol for every instance  
(Better score)



Nothing interesting for a publication (at least in my domain)

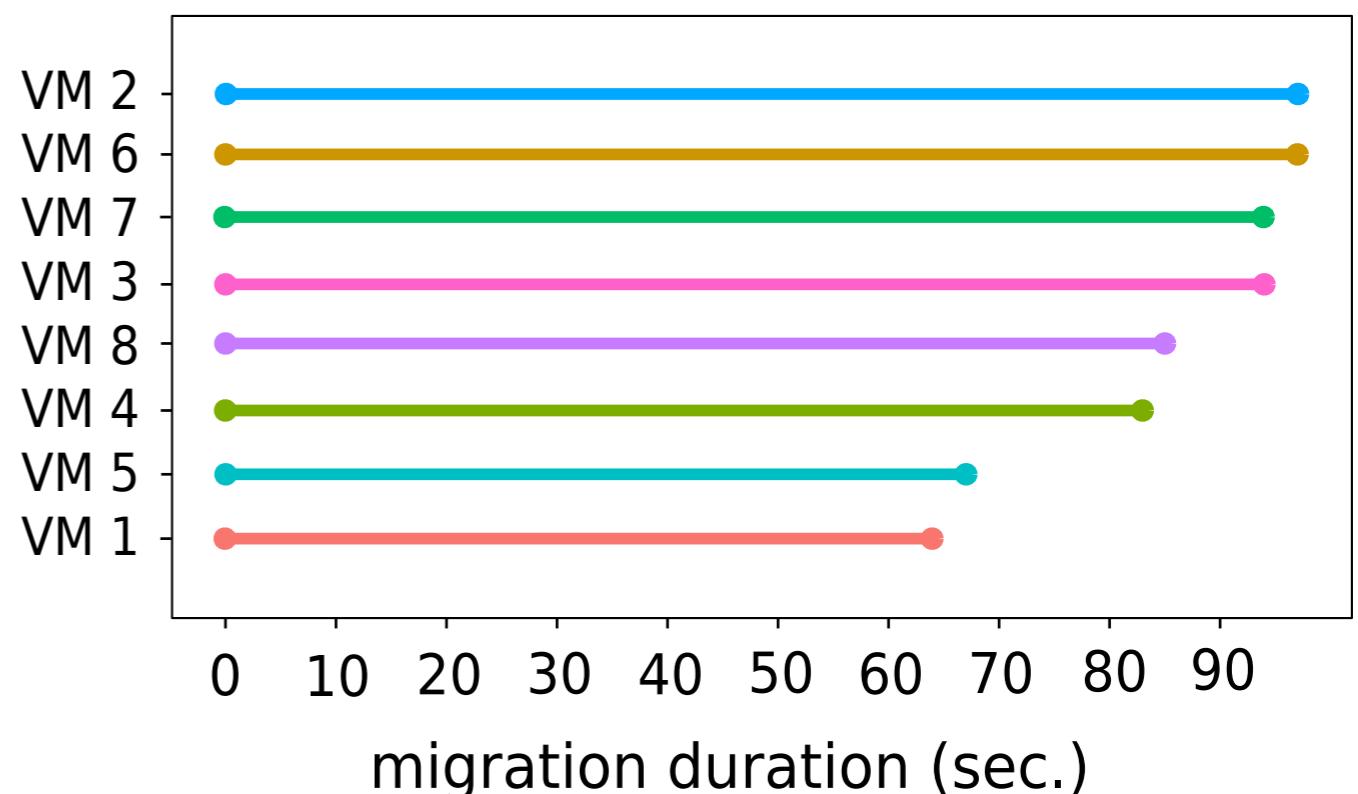
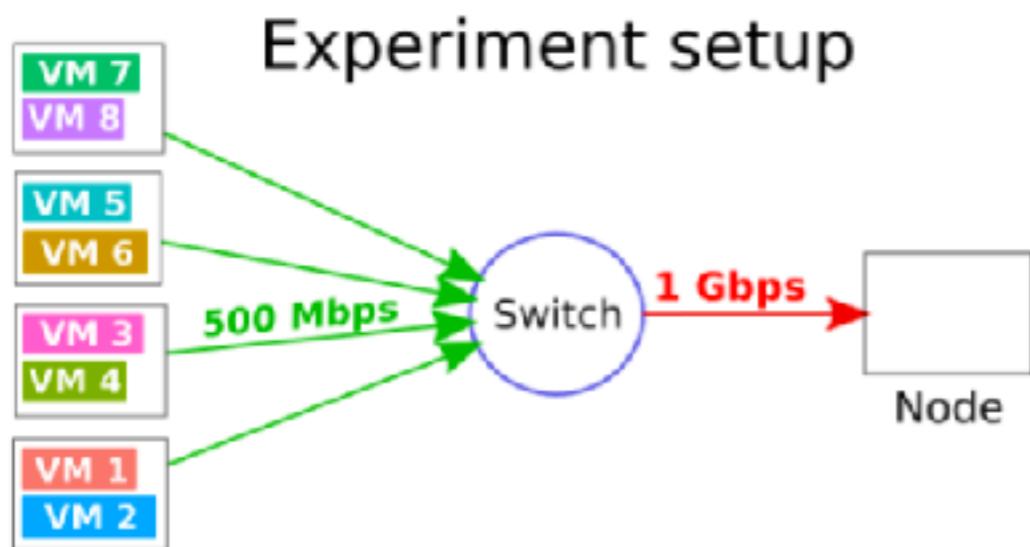
Before, the model was overkill



# 2013-2016

looking for a better migration scheduler  
[Vincent Kherbache work]

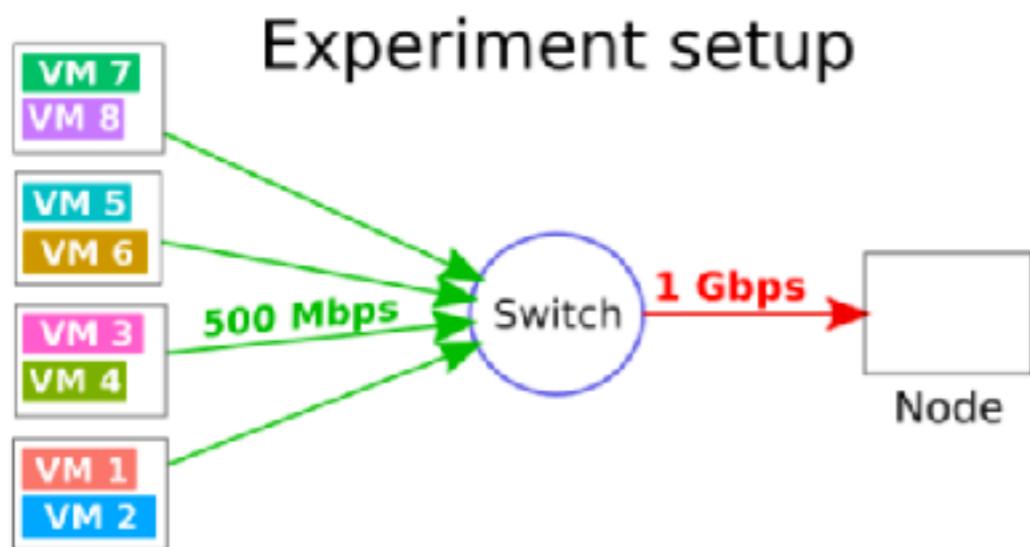
[btrplace vanilla, entropy, cloudsim, ...]



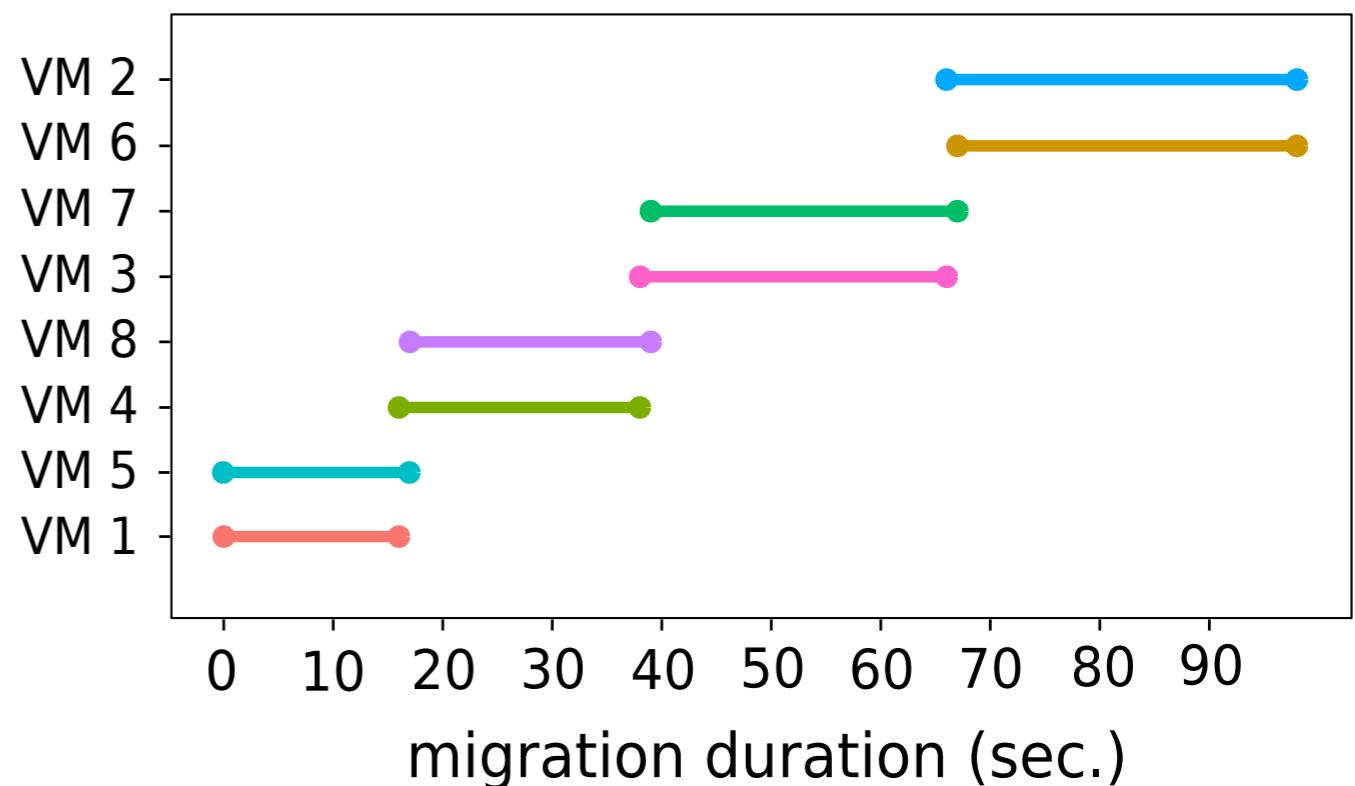
network and workload blind

# 2013-2016

looking for a better migration scheduler



btrplace + migration scheduler [UCC 15, TCC 17]



network and workload aware

# Coding effort

Network Model

heterogeneous network

*1 cumulative constraint per network element;  
+/- 300 sloc.*

Migration Model

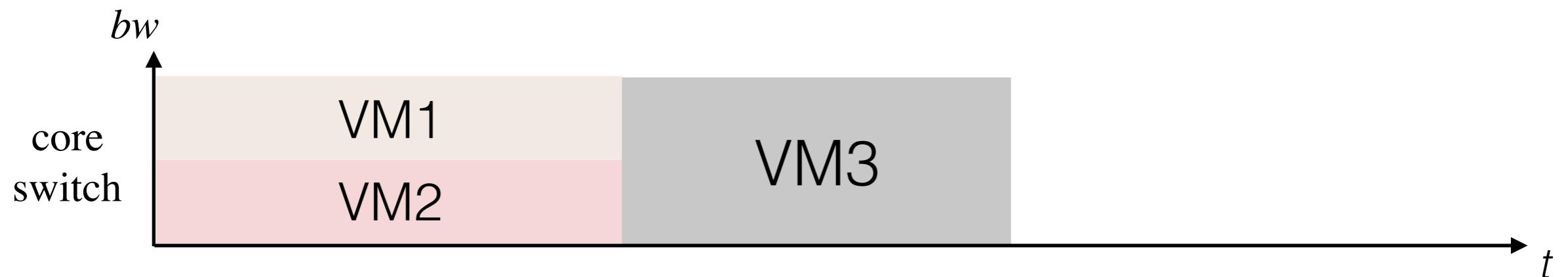
memory and network aware

*+/- 200 sloc.*

Constraints Model

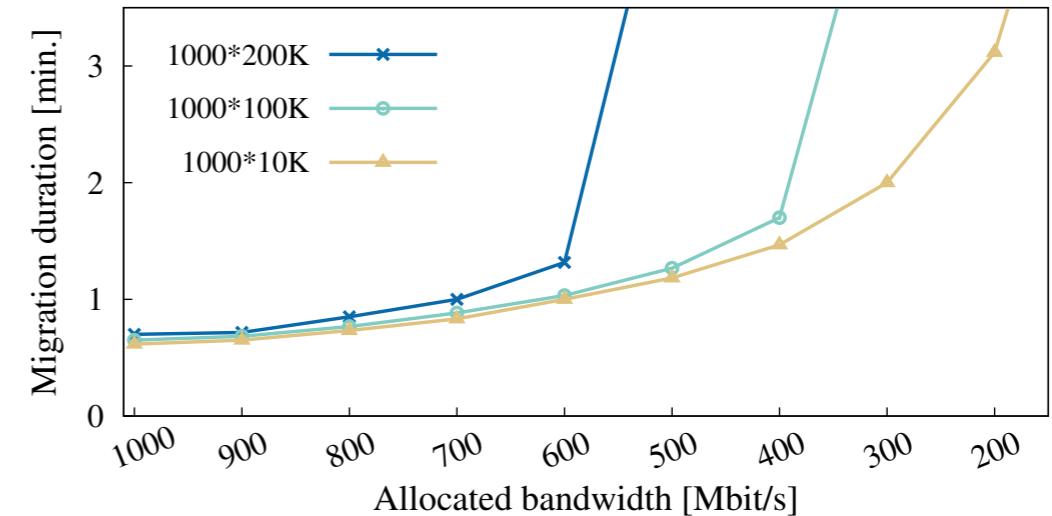
restrict the migration models

*+/- 100 sloc.*



# Migration Model

Phase 0 (6 mo.)  
observe, experiment



Phase 1 :  
mathematical model  
(Few months)

$$d_{CP}(m) = \frac{CP_s}{bw(m) - CP_r}$$

$$d_{HP}(m) = \frac{HP_s}{bw(m)} + \frac{HP_s - (D \times bw(m))}{bw(m) - HP_r}$$

$$d_{CP}(m) + d_{HP}(m) = \frac{CP_s - ((D \times bw(m)) - HP_s)}{bw(m) - CP_r} + \frac{HP_s}{bw(m)}$$

$$d(m) = d_{min}(m) + d_{CP}(m) + d_{HP}(m) + D$$

# Phase 2 : CSP models

Try 1 - maths to CSP 101

Division, truncation issues  
does not scale

Try 2: chunk based bandwidth allocation

e.g. : 250, 500, 7 50, 1Gb

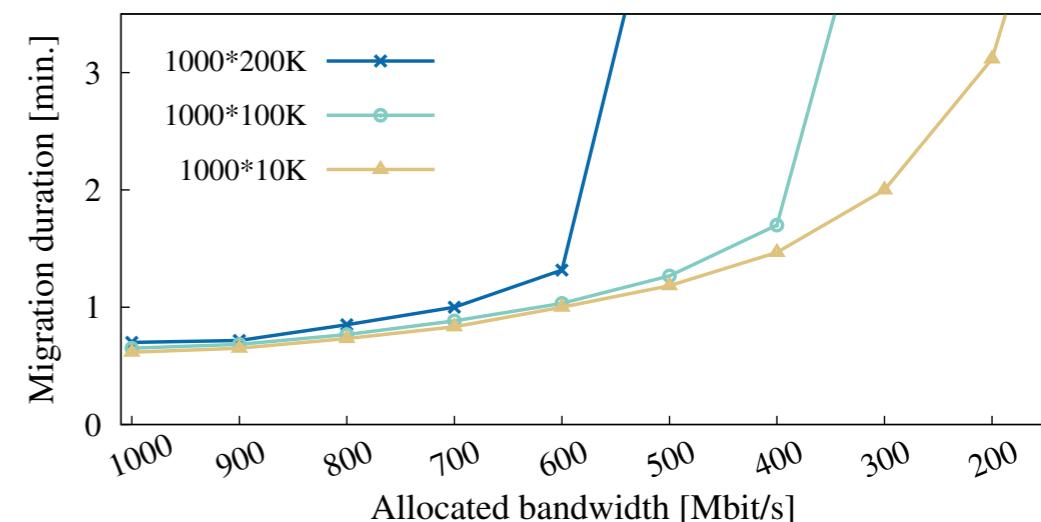
table constraint

better scalability

Try 3: ... why slowing down migrations ?

Force max bandwidth

The duration as a constant

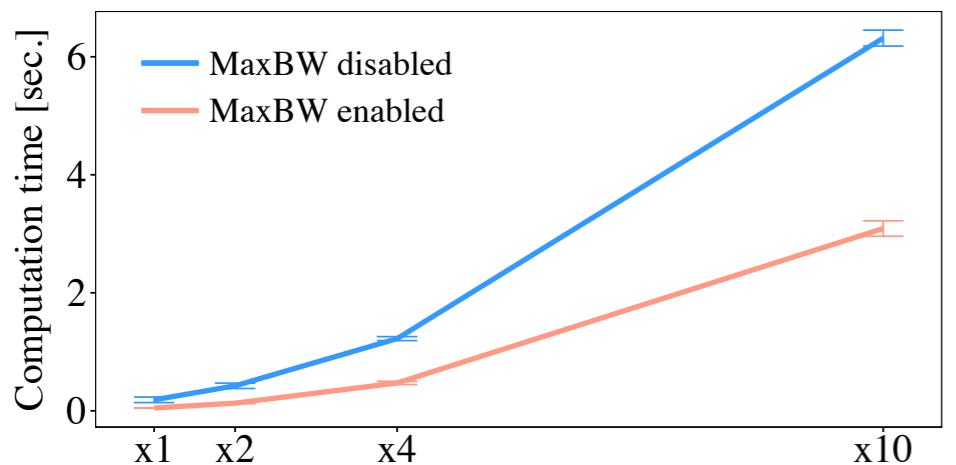


# Evaluating the benefits of “max bandwidth”

<b>Scale</b>	<b><i>MaxBandwidth</i> option disabled</b>	<b><i>MaxBandwidth</i> enabled</b>
x1	65%	100%
x2	22%	100%
x4	83%	100%
x10	62%	100%

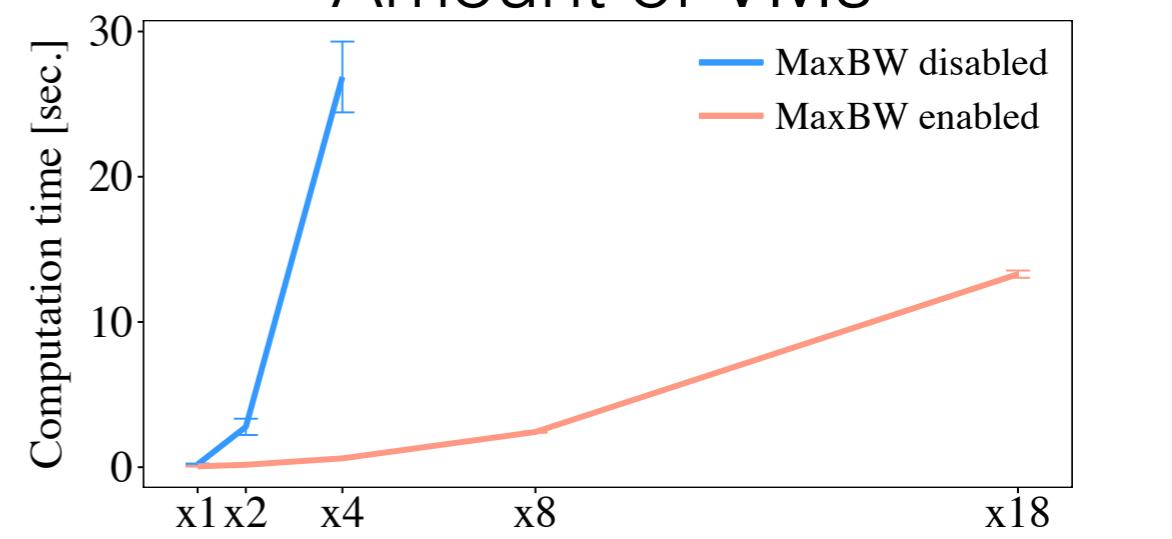
<b>Scale</b>	<b><i>MaxBandwidth</i> option disabled</b>	<b><i>MaxBandwidth</i> enabled</b>
x1	66%	100%
x2	57%	100%
x4	47%	100%
x8	0%	100%
x18	0%	100%

Infrastructure size



960 VMs,  
20x24 nodes

Amount of VMs



1728 VMs,  
2x24 nodes

# Placement + migration scheduler

Back to 2006 solving workflow :D

Phase 1: placement & old schedule

Phase 2: new scheduler (requires the route)

Not sure about the need to consolidate the phase

Can't saturate the network

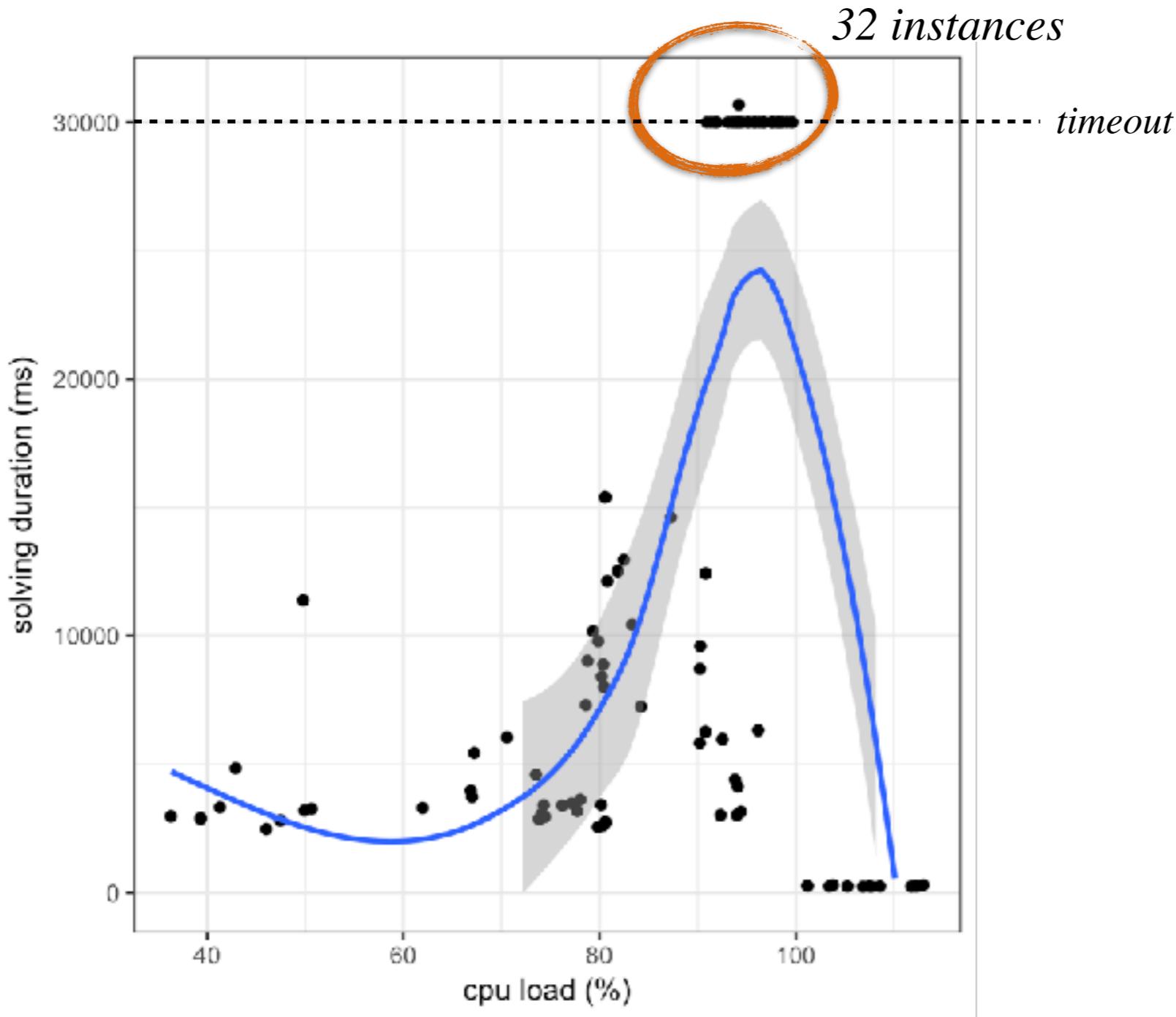
Workarounds exist

# 2017

## The story of the knapsack



# The right filtering algorithm for the right workload



very high load  
small but hard instances

ok when non-solvable  
but no evidence

# knapsack filtering

simple to understand, to develop

*“Iterate over the candidate items and filter out the oversized”*

called after every assignment

$O(n)$  worst case complexity

dsc. sorting items helps ...

but costly with several dimensions

Sort items per dimension inside the constraint

Indirection tables to ensure compatibility with the core model

# knapsack filtering

Strong filtering iff packing objective  
high load  
big items

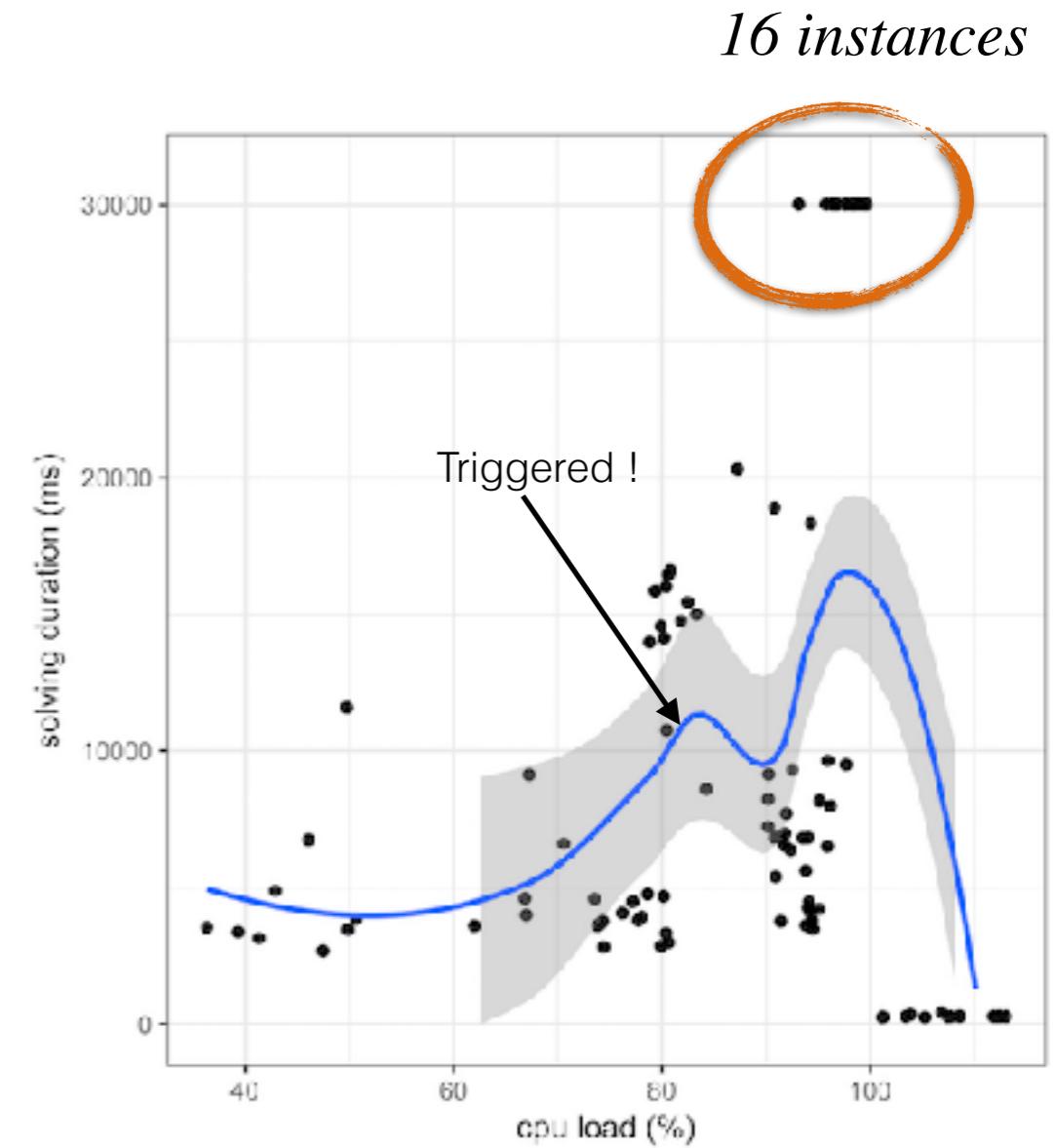
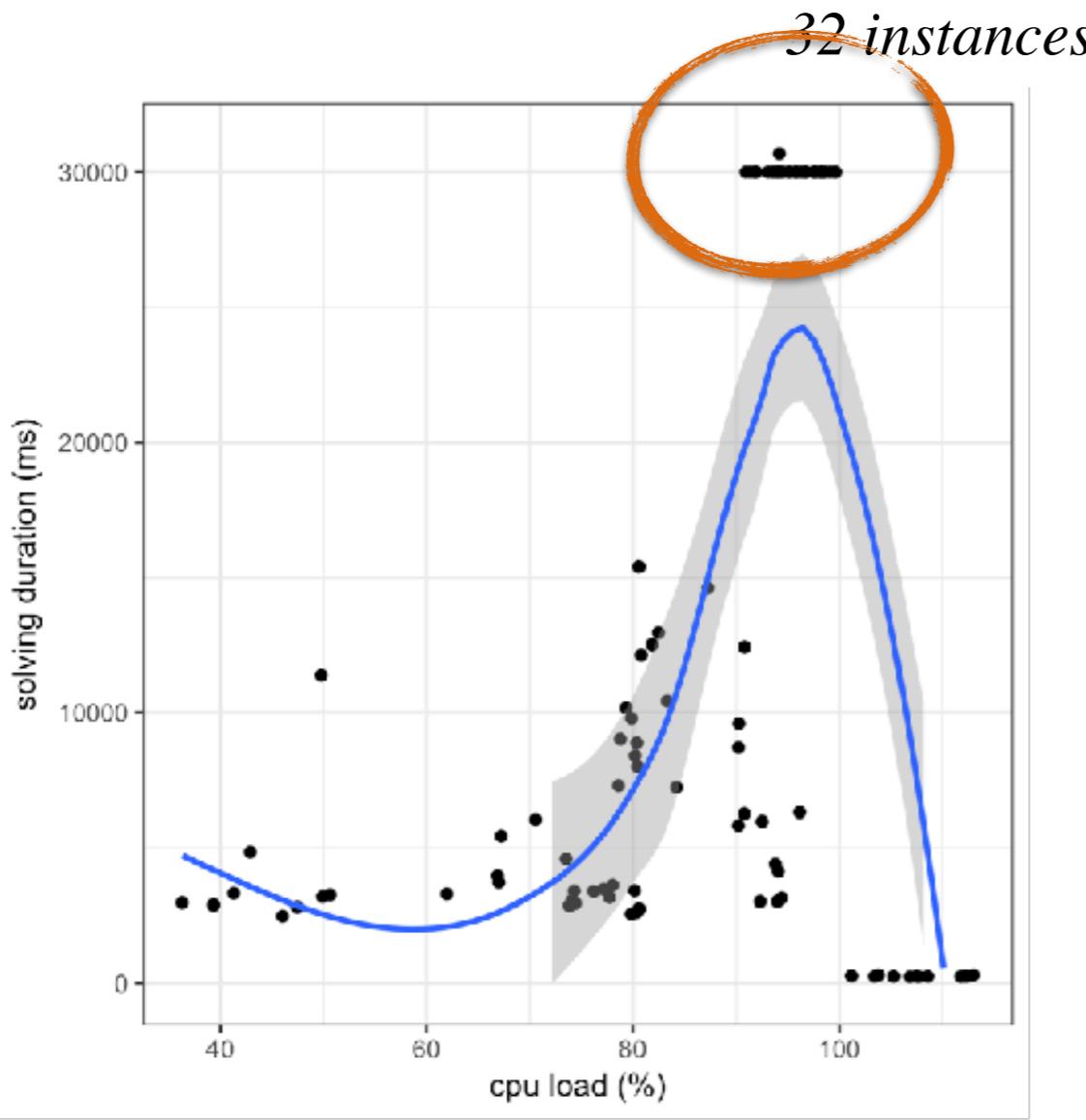
Low filtering when balance objective  
low load  
small items

With sorted item, memory usage  
increases with the dimensions

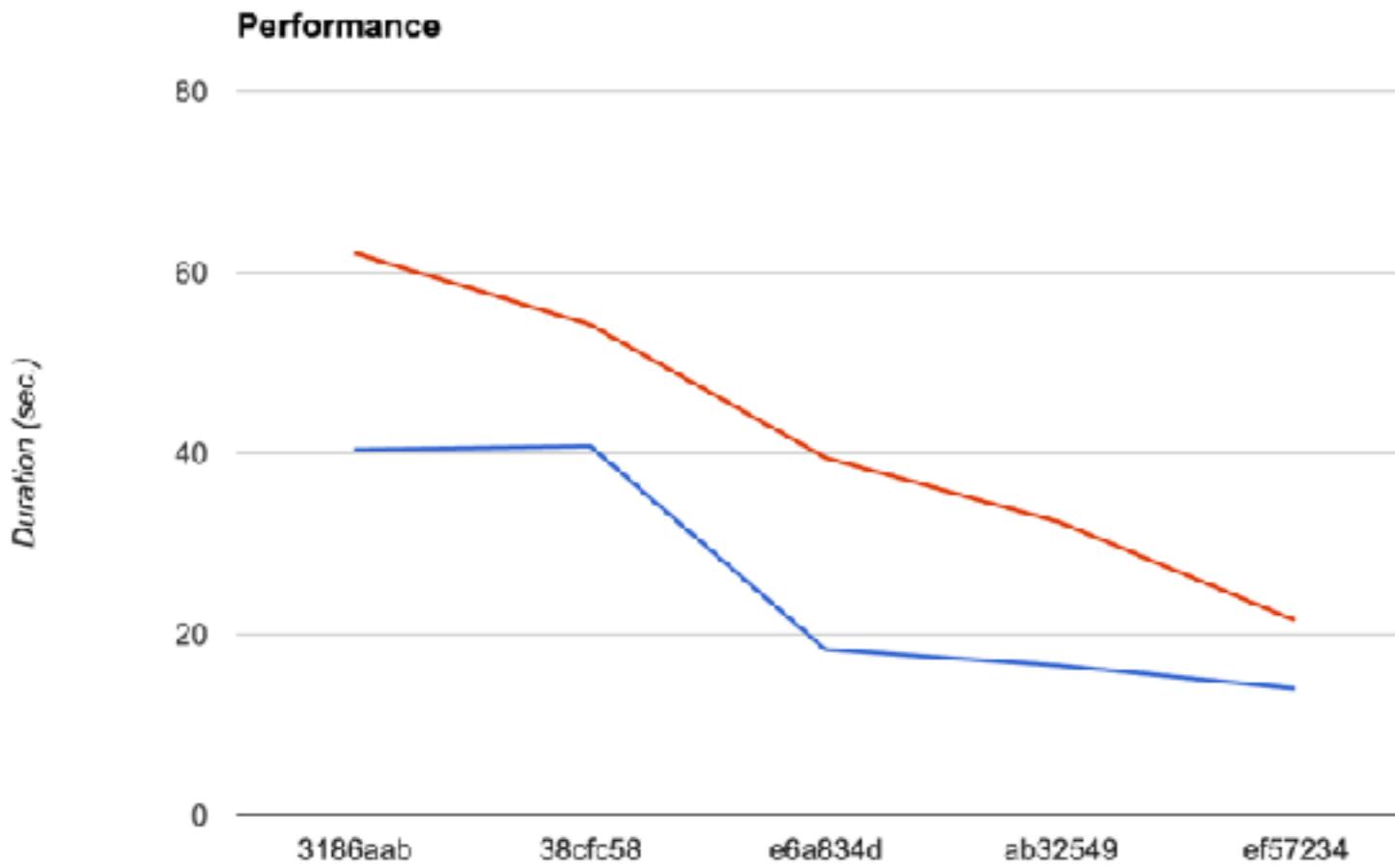
Overhead > benefits

# Triggered Knapsack

Triggered if the residual < biggest item  
bigger constants



# Review everything



understand the workload,  
tune the model  
tune the solver  
tune the heuristics

38cf58: [Choco.4 FirstFail] heuristic from the  $O(n)$  avg. legacy first fail to a  $O(n)$  worst case  
(iterates from the last instantiated variable, stop when Dom size == 2)

e6a834d: [task scheduling] simplistic local entailment. Stop early if all variables are instantiated

ab32549: [vector packing] cap cardinalities wrt. resource usage

ef57234: [memory] allocate memory per chunk to prevent increase with copy

**STOP**

# RECAP

I am  
from the (distributed) system community  
Not  
from the CP community



# Composability is gold

Fit the way we think, we work, we upgrade  
Take care of flexibility overhead

# Multi expertise required

Domain expertise  
CSP / solver expertise  
Engineering expertise

# global constraints are effective

But re-usability has its limits  
*“The right model/filtering for the right problem”*

*“The granularity is good when going finer does not change the decisions”*

— Fabien Hermenier, right now

# A regular dev writing a constraint ?

Deterministic event-based programming

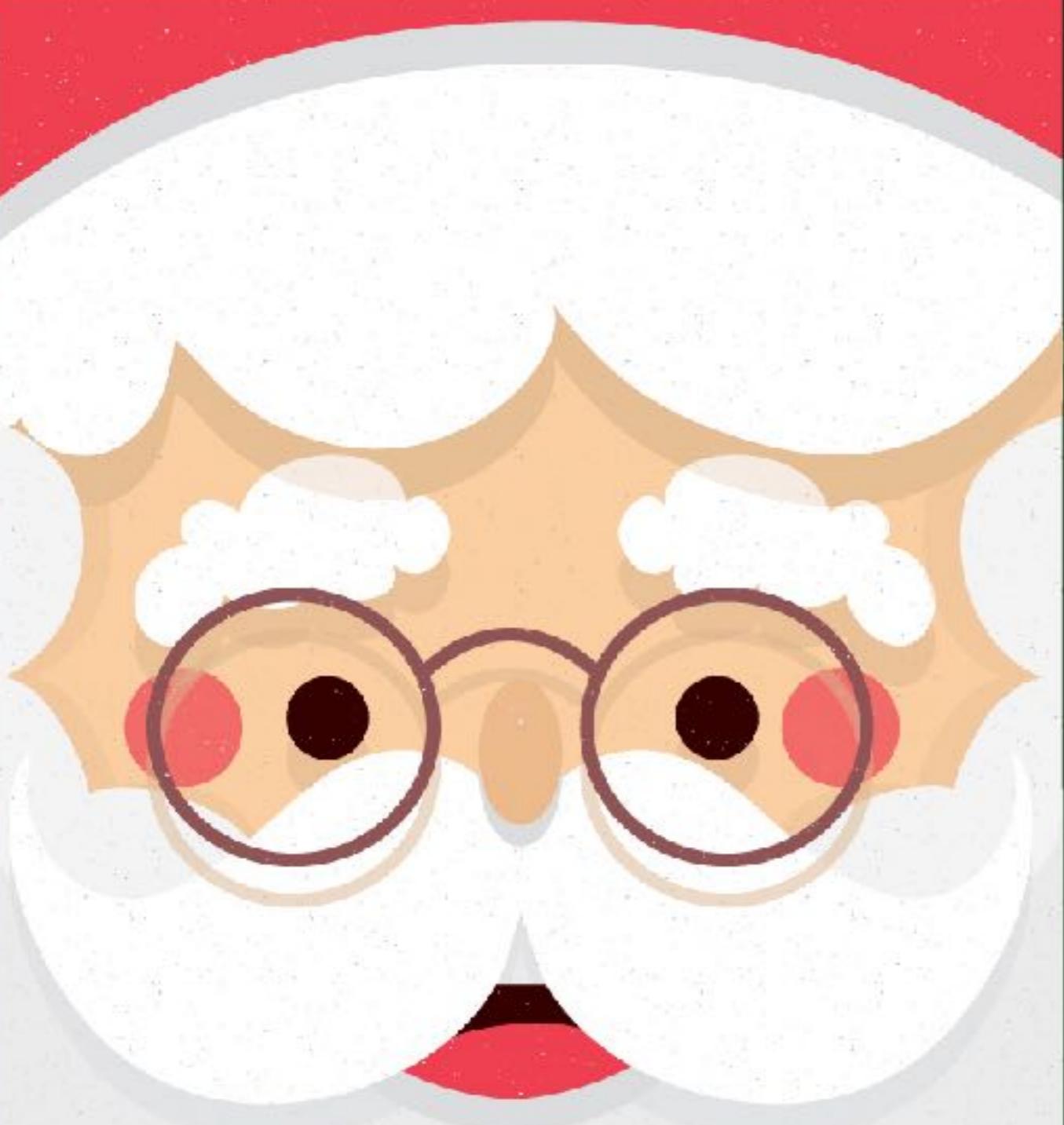
Error prone, no one like

Hard to debug

Jump to the 36th instance of constraint X, at the 217th search nodes, in the 47th. awakeOnRem

Unusual reasoning





Wish list of one  
CP enthusiast

Help me  
at  
modeling,  
developing,  
checking

redundancy checkers,  
effectiveness checkers,  
convenient programming model,  
debugger,  
...

Composability is gold

fit development model

uber-genericity can be costly

# Global constraints are great

Capture the problem.  
Limit math gymnastic

Be careful with swiss-knife constraints

*“The right constraint for the right problem”*

# The multi-expertise problem

domain-expertise  
solver-expertise  
programming expertise

Trade for the best

*“The right model granularity is when adding information does not change decisions”*

Over 10 years

solve problems 10 times bigger, 100 times faster while being more complex

A regular developer  
writing a constraint ?      **LOL**

Deterministic event-based programming

Error prone, no one like

Hard to debug

Jump to the 36th instance of constraint X, at the  
217 search nodes, in the 47th. awakeOnRem

# Wish list of a CP enthusiast

Help me at modelling, checking, debugging

Is there useless constraints ?

Do I use the appropriate constraint ?

How to check if my constraint is buggy (P.S: safePlace)



[http://\*\*BtrPlace\*\*.org](http://BtrPlace.org)

production ready

live demo

stable user API

documented

tutorials

issue tracker

support

chat room