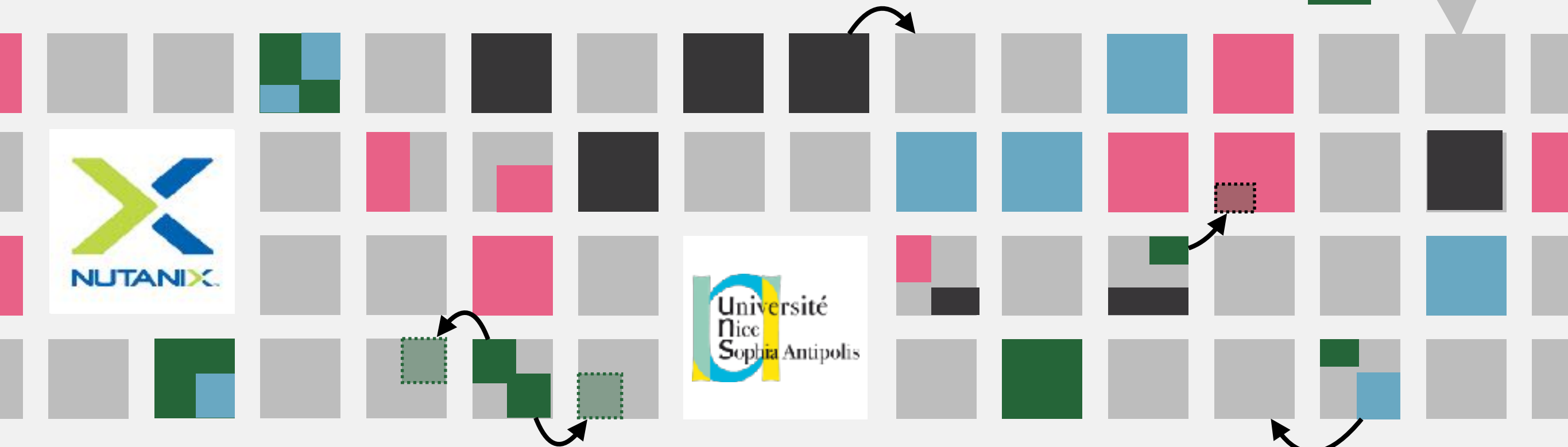
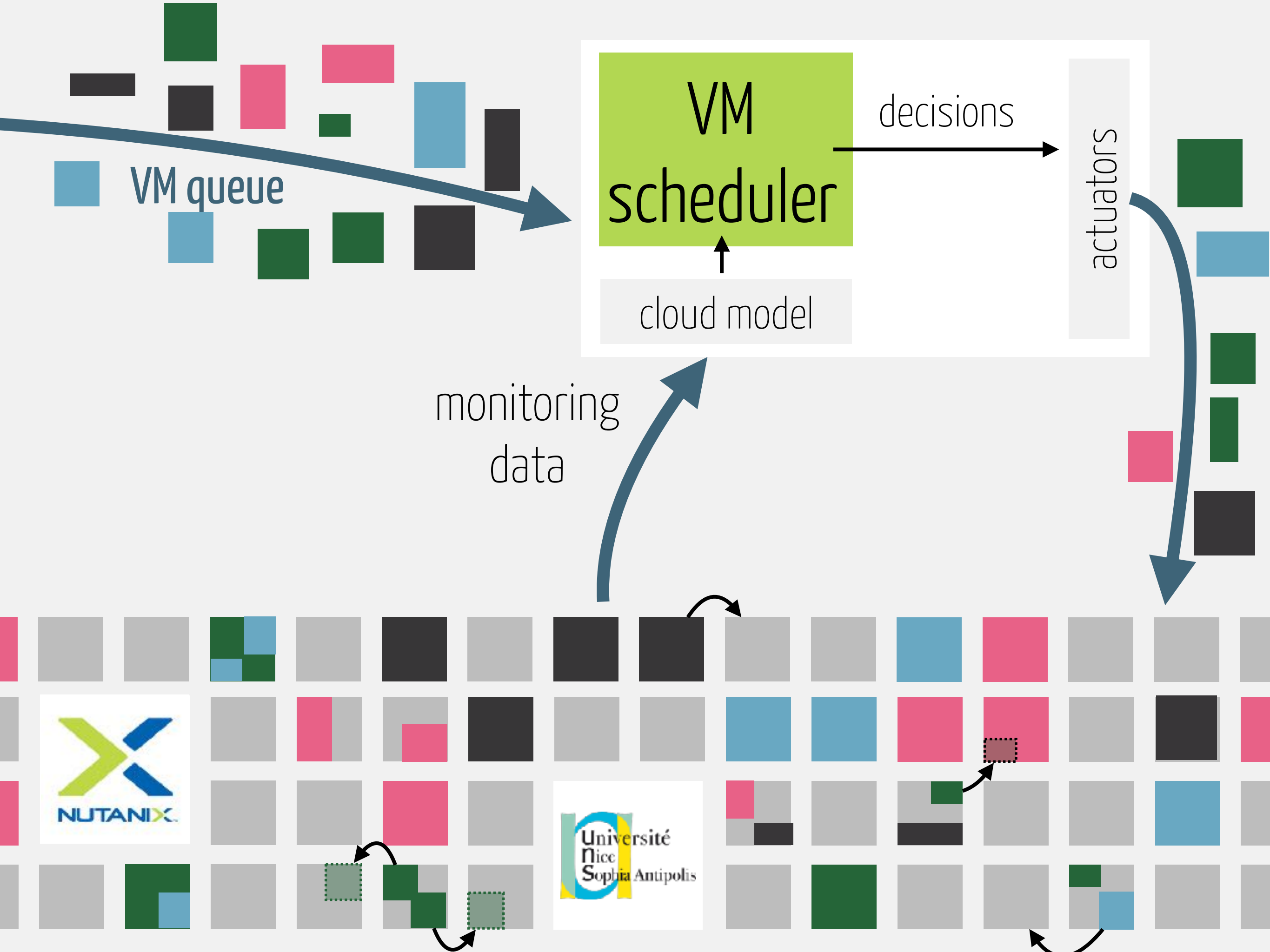


Trustable VM Scheduling in a Cloud

Fabien Hermenier, Ludovic Henrio





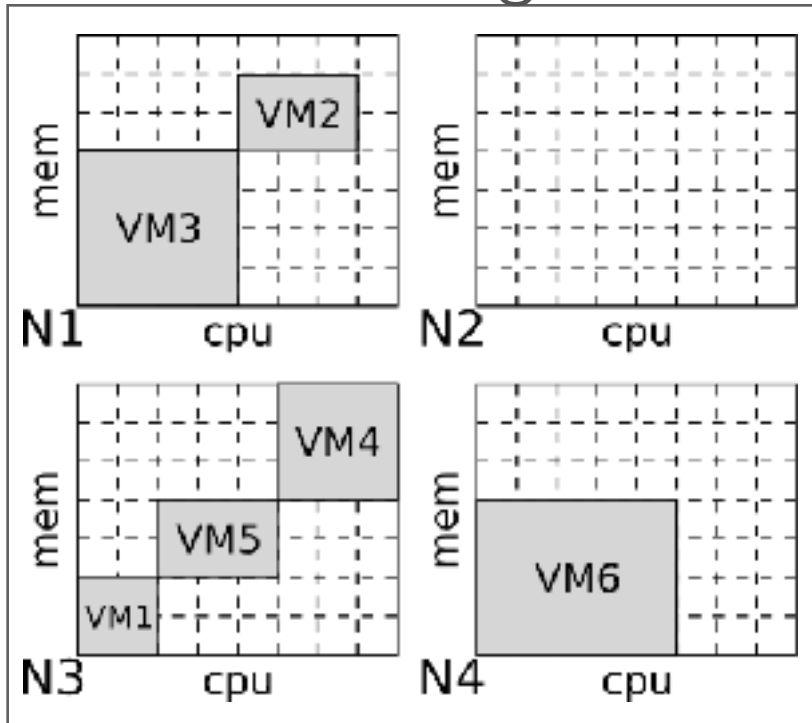


cloudstack



Microsoft
Hyper-V

current configuration

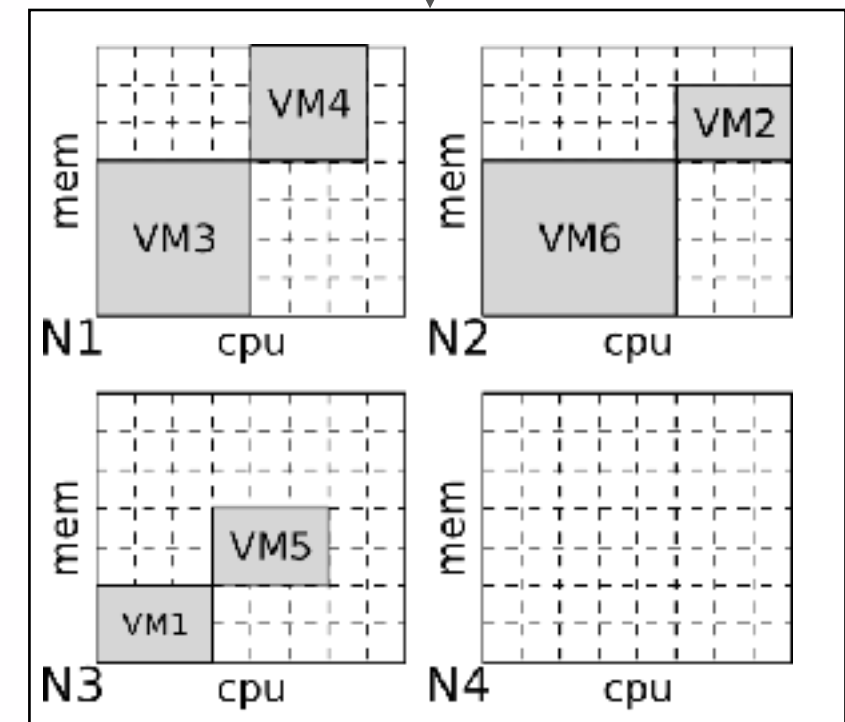


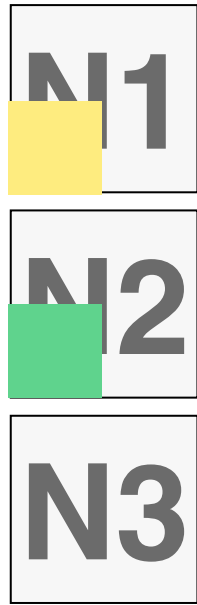
constraints

```
anti-affinity(VM[2..3]);  
allocate({VM1}, 'ucpu', 3);  
offline(@N4);
```

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)
```

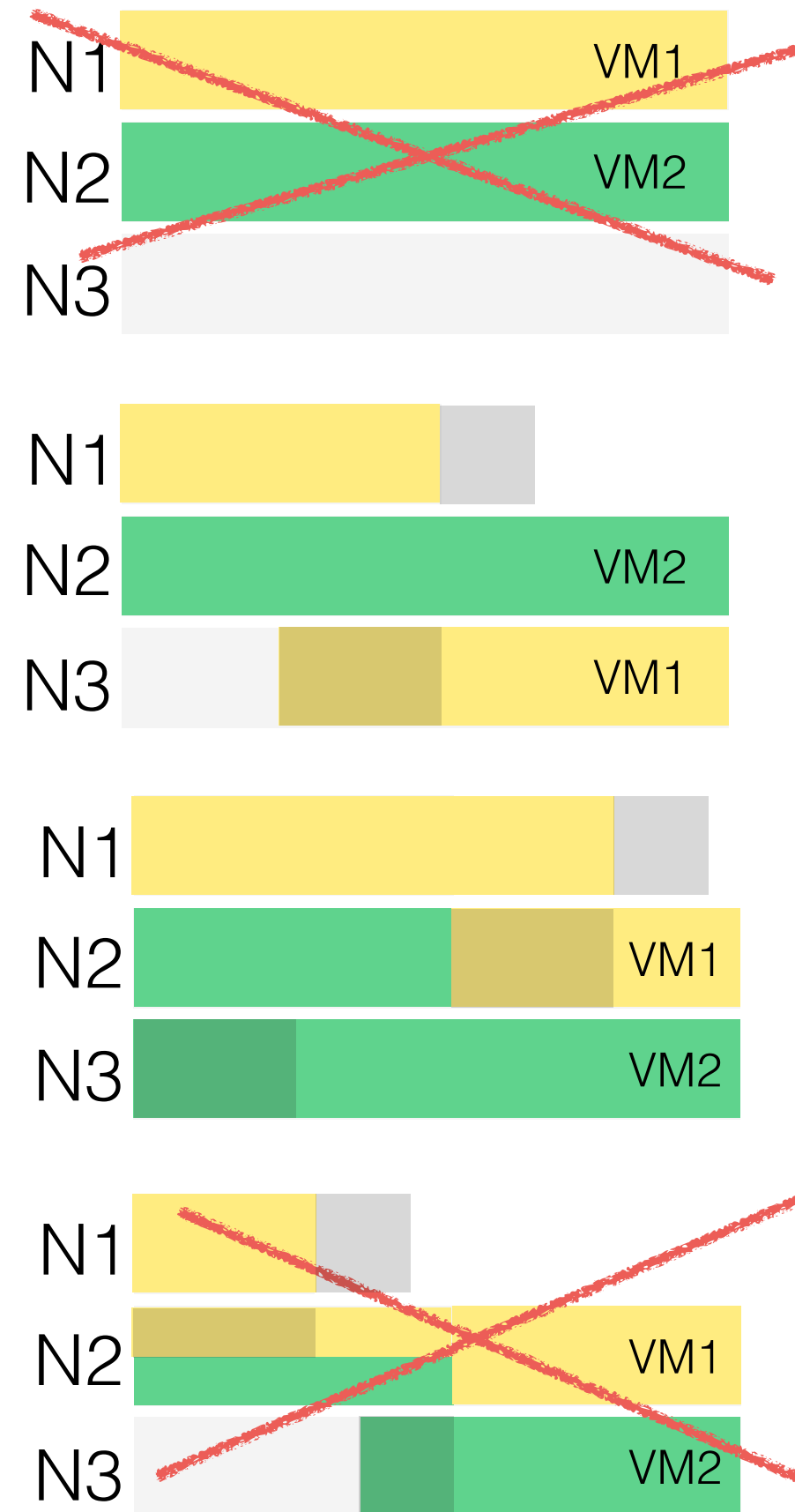




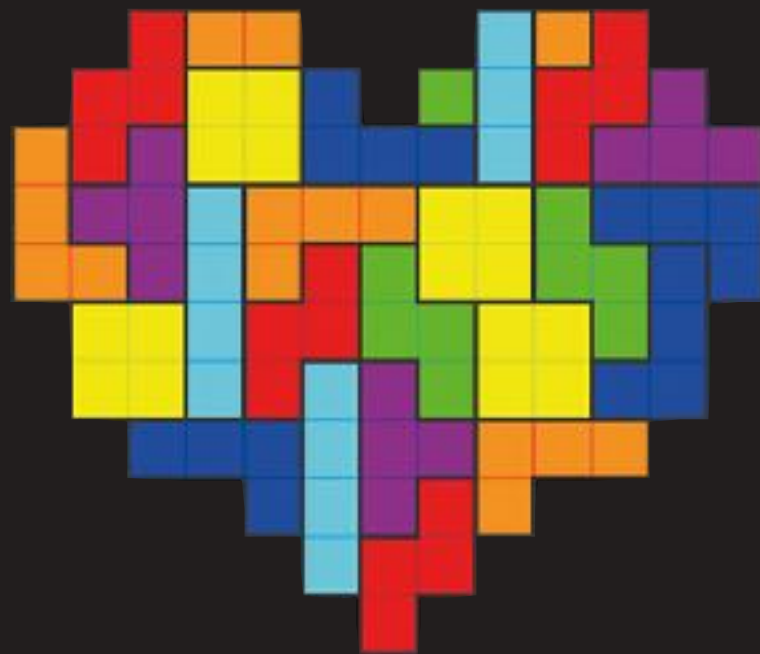
$$\begin{array}{l}
 + \\
 \text{running(VM1)} \\
 \text{running(VM2)} \\
 \text{isolate(VM2)} \\
 \text{offline(N1)}
 \end{array}
 = ?$$

computing solutions is filtering
out non-viable decisions

Explicit in OpenStack, CloudStack
Implicit in BtrPlace

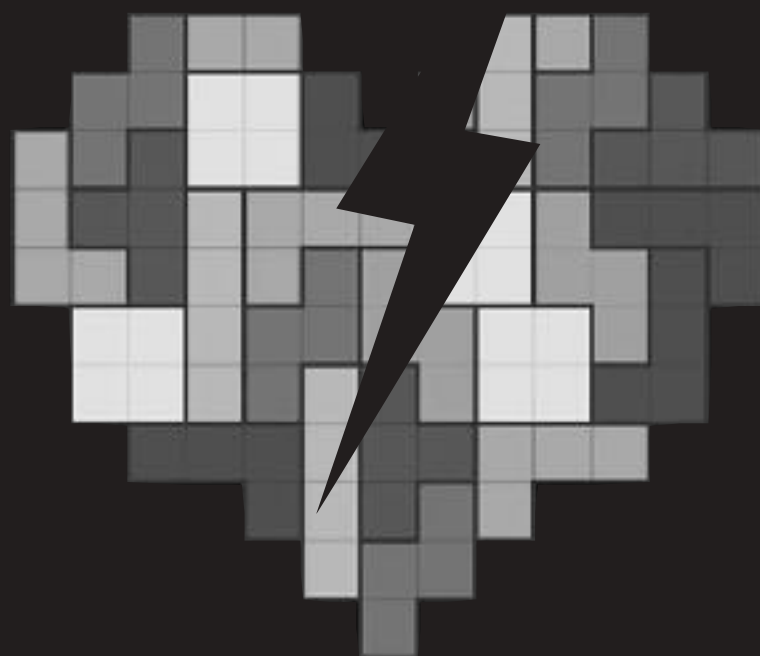


VM scheduler brings



high consolidation, performance,
trustworthy placements, valid schedules

Behind the scene



over-filtering

Filter-out viable decisions
Reduce the hosting capabilities

under-filtering

Let non-viable decisions
Break SLA & user confidence

crash

...

```
/* CAmongTest.java */
@Test public void testContinuousWithNotAlreadySatisfied() {...}
@Test public void testWithOnGroup() { ...}
@Test public void testWithGroupChange() {...}
@Test public void testWithNoSolution() {...}
@Test public void testContinuousWithAlreadySatisfied() {...}
```

A limited vision of the significant use cases
(specific state/transitions)

❏ ⓘ **Continuous among is too restrictive** bug
#44 opened on 29 Aug 2014 by fhermeni



fhermeni commented 6 minutes ago

Owner

In theory, there is a solution to this problem: <https://gist.github.com/fhermeni/76358167e5371ce6c128>
Only VM1 is running so it's ok to migrate it to the second partition, then to boot the other vms.



CloudStack / CLOUDSTACK-8896

Allocated percentage of storage can go beyond 100%

MEDIUM

FIX RELEASED

#1379451 anti-affinity policy only honored on boot

📄 OpenStack Compute (nova) 🔥 26 🔄

MEDIUM

CONFIRMED

#1012822 broken instances are considered to be consuming resources

📄 OpenStack Compute (nova) 🔥 34 🔄

A limited expertise in the theoretical foundations

discrete $\text{maxOnline}(N[1..10], 7) ::=$

$$\sum_{i=1}^{10} n_i^q \leq 7$$

ZERØ

to HERO★

continuous $\text{maxOnline}(N[1..10], 7) ::=$

$$\forall i \in [1, 10], \quad n_i^{on} = \begin{cases} 0 & \text{if } n_i^q = 1 \\ a_i^{start} & \text{otherwise} \end{cases}$$

$$n_i^{off} = \begin{cases} \max(T) & \text{if } n_i^q = 0 \\ a_i^{end} & \text{otherwise} \end{cases}$$

$$\forall t \in T, \text{card}(\{i | n_i^{on} \geq t \wedge n_i^{off}\}) \leq 7$$

continuous constraints [hotdep'13]

**unit tests,
smoke testing,
peer review
cannot address
reasoning issues**



a specification language

to state the awaited VM scheduler behaviour

fuzz testing + simulator

to exhibit reasoning issues

applied to BtrPlace

The specification language

First order logic

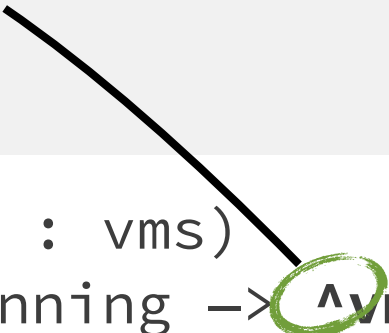
Business functions in native code (Java)

Added dynamically through reflections

Provide extensibility

*temporal call,
Refers to the initial state*

Core constraints reflect element lifecycle
Must always be satisfied



```
toRunning ::= !(v : vms)
  vmState(v) = running -> ^vmState(v) : {ready, running, sleeping}

toReady ::=
  !(v : vms) vmState(v) = ready -> ^vmState(v) : {ready, running}

toSleeping ::=
  !(v : vms) vmState(v) = sleeping -> ^vmState(v) : {sleeping, running}

noVMsOnOfflineNodes ::=
  !(n : nodes) nodeState(n) /= online -> card(hosted(n)) = 0
```

The specification language

First order logic

Business functions in native code (Java)

Added dynamically through reflections

Provide extensibility

Side constraints are enabled on demand

```
RunningCapacity(ns <: nodes, nb : int) ::=
  sum([card(running(n)). n : ns]) <= nb  //array build on comprehension

MaxOnline(ns <: nodes, nb : int) ::=
  card({i. i:ns , nodeState(i)=online}) <= nb

ShareableResource(id : string) ::=
  !(n : nodes) sum([cons(v, id). v : host(n)]) <= capa(n, id)

Among(vs <: vms, parts <<: nodes) ::= //parts is a packing (set of sets)
  ?(g : parts) {host(i). i : vs, vmState(i) = running} <: g
```

The testing framework

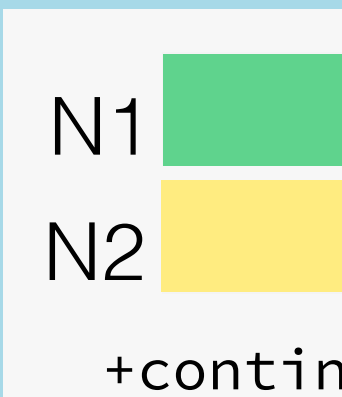
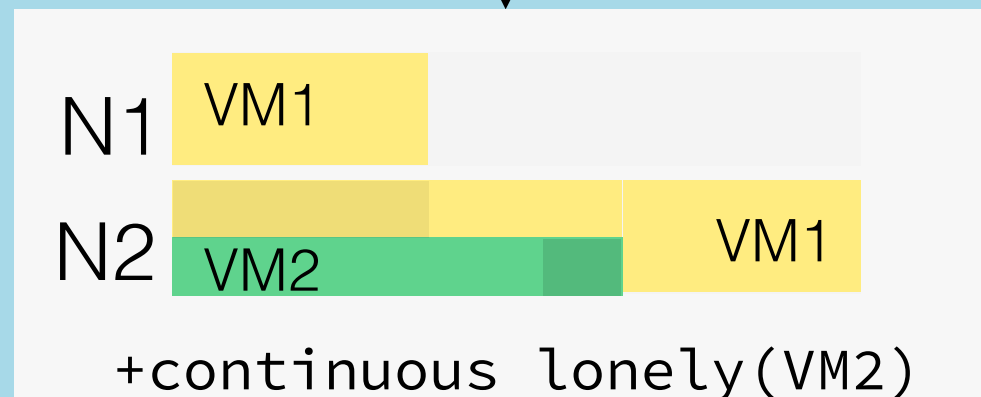
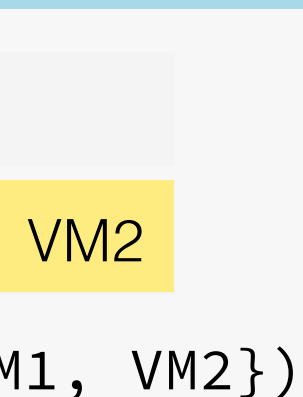
Test campaign

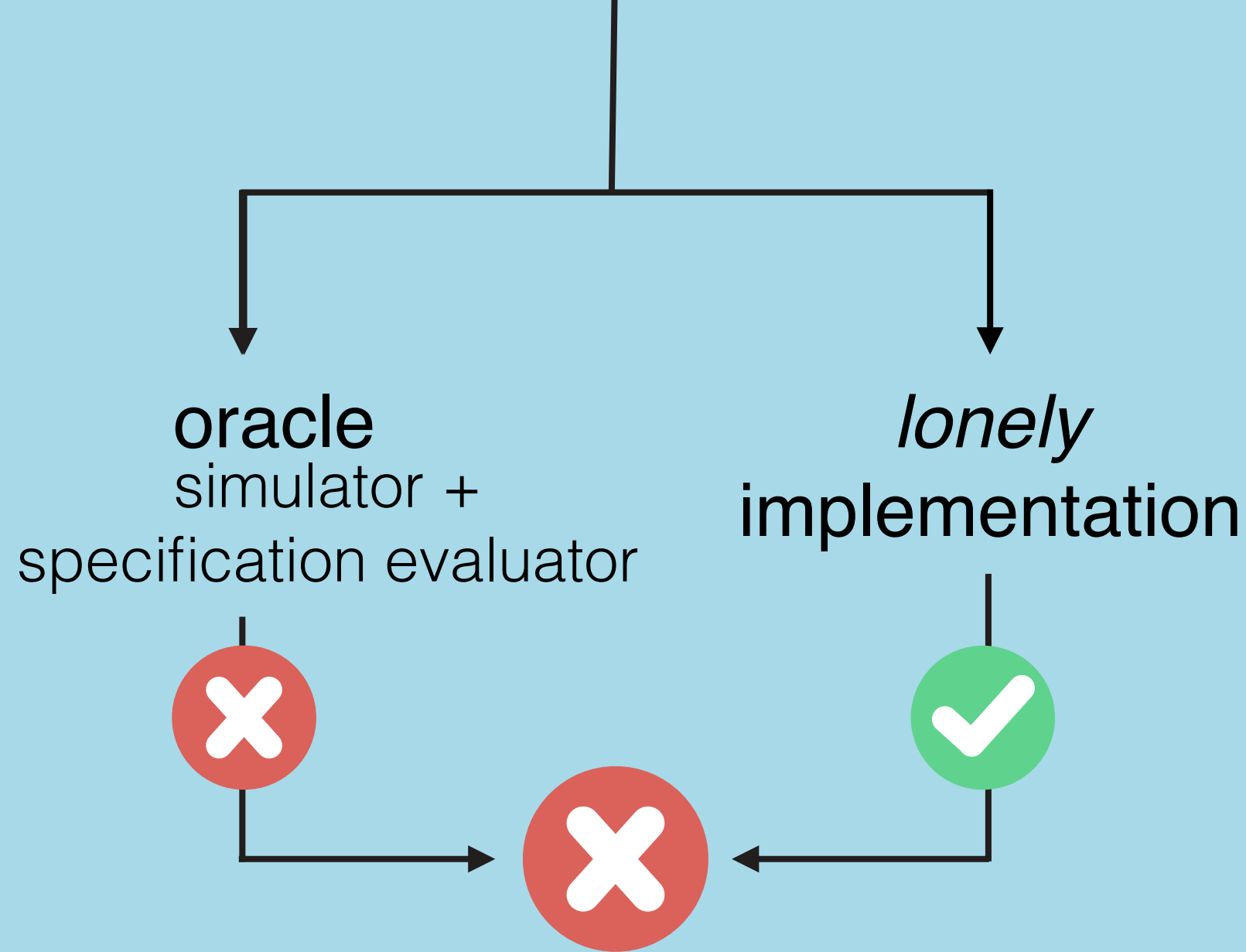
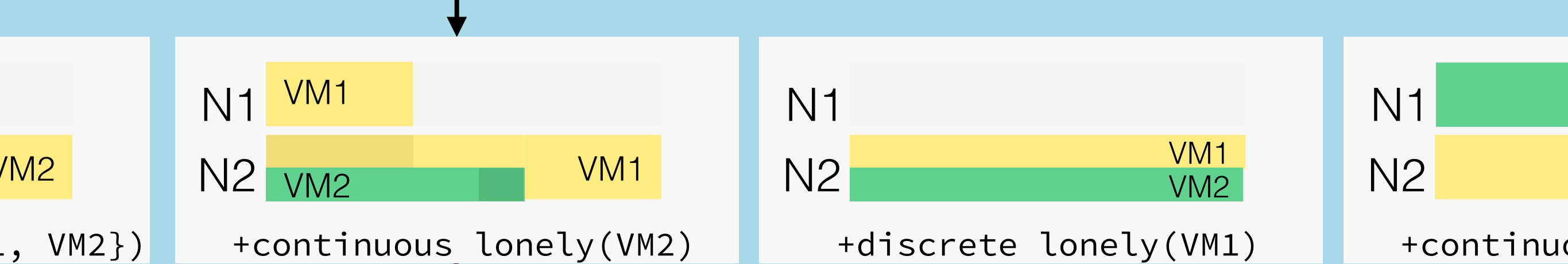
```
@CstrTest(groups = {"lonely", "affinity"})
public void testLonely(TestCampaign c) {
    c.fuzz().constraint("lonely")
        .vms(2).nodes(2).srcVMs(1, 9, 0);
    c.limits().tests(100).failures(1);
}
```



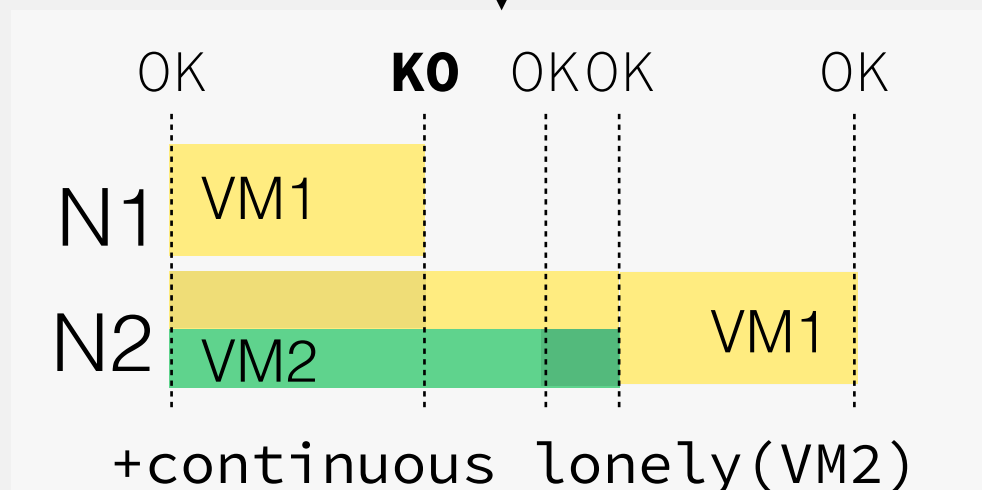
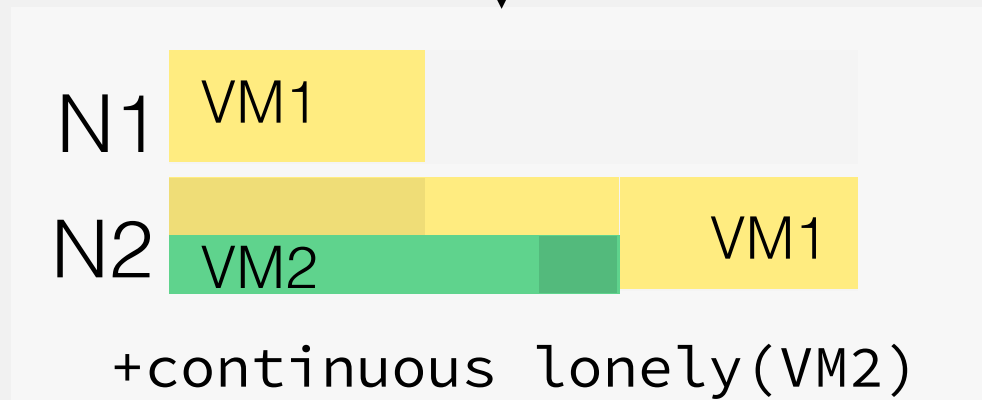
test case fuzzer

TestCase
valid plan + sample constraint





The testing phase exhibits the inconsistencies

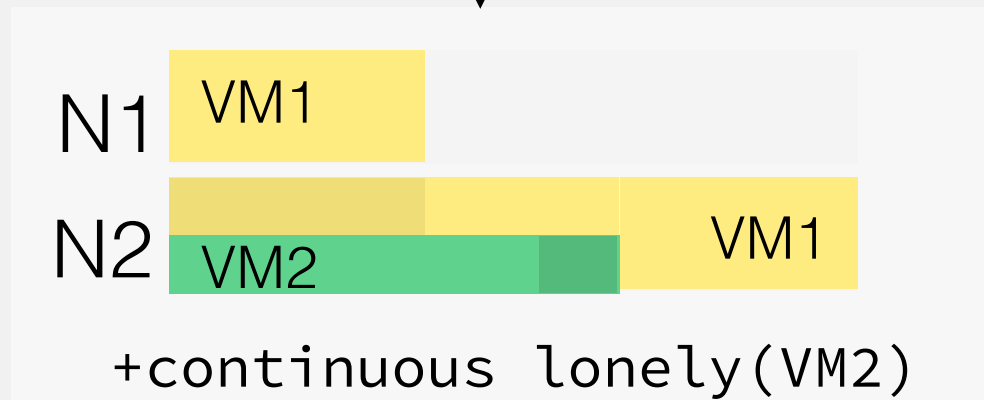


Testing with an oracle

a simulator executes the plan

the invariant is checked at every timestamp of interest

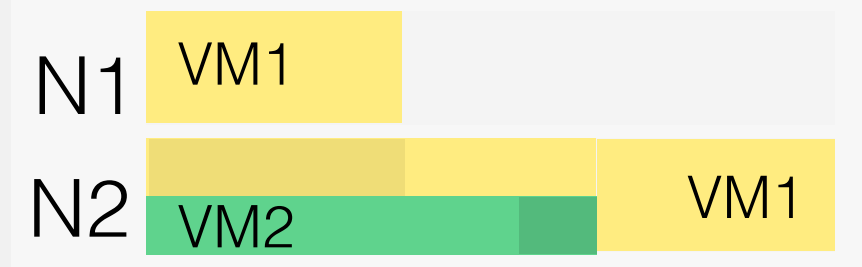
```
lonely(vs <: vms) ::=
  !(i : vs) vmState(i) =
    running --> (colocated(i) - {i}) <: vs
```



online(N[1..2])
running(VM1)
shutdown(VM2)
fence(VM1, N2)
schedule(VM1, 0, 11)
schedule(VM2, 20, 26)
continuous lonely(VM2)



btrplace < 1.9

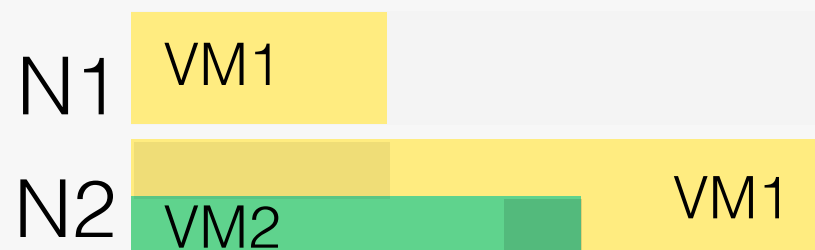


btrplace >= 1.9

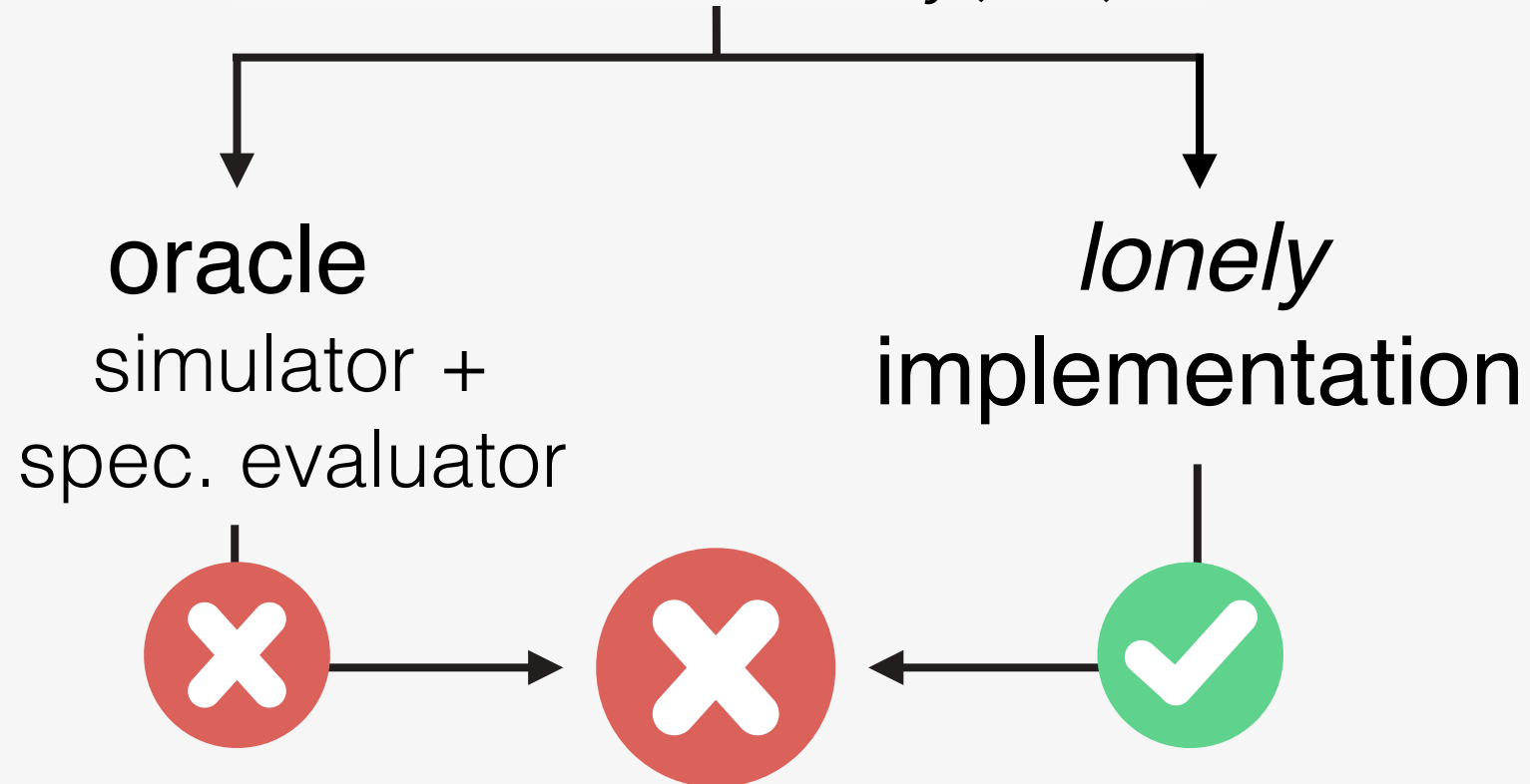


Testing BtrPlace

The test case is turned to a heavily constrained instance to solve.



+continuous lonely(VM2)



| | | implementation | | |
|--------|----|-----------------|----------------|-------|
| | | OK | KO | CRASH |
| oracle | OK | | over filtering | crash |
| | KO | under filtering | | crash |

Evaluation

usable
for developers

useful
to find reasoning
defects



Specification capabilities

Formal documentation
Outside the business code



All the constraints (27)
state transition, action schedule,
resource sharing, affinities, counting



All the constraints



Theoretical suitability

short invariants

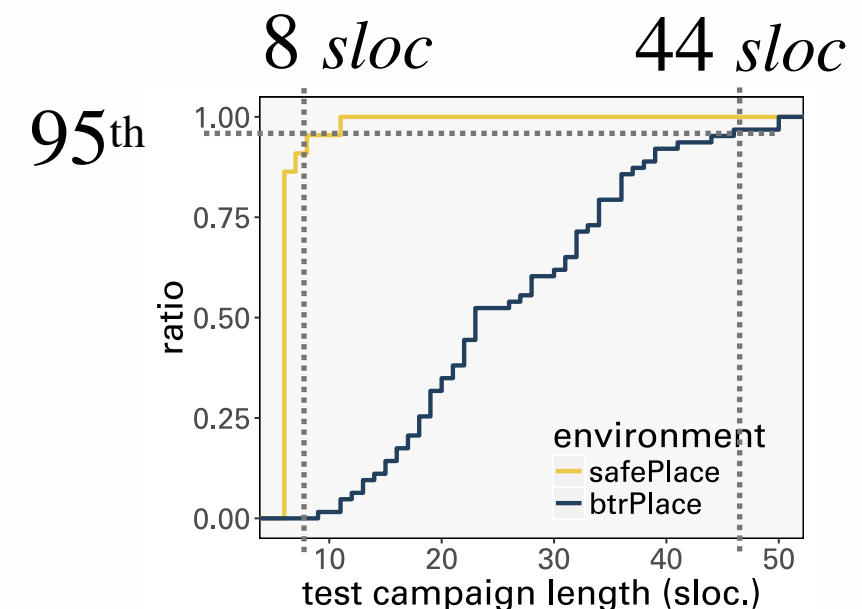
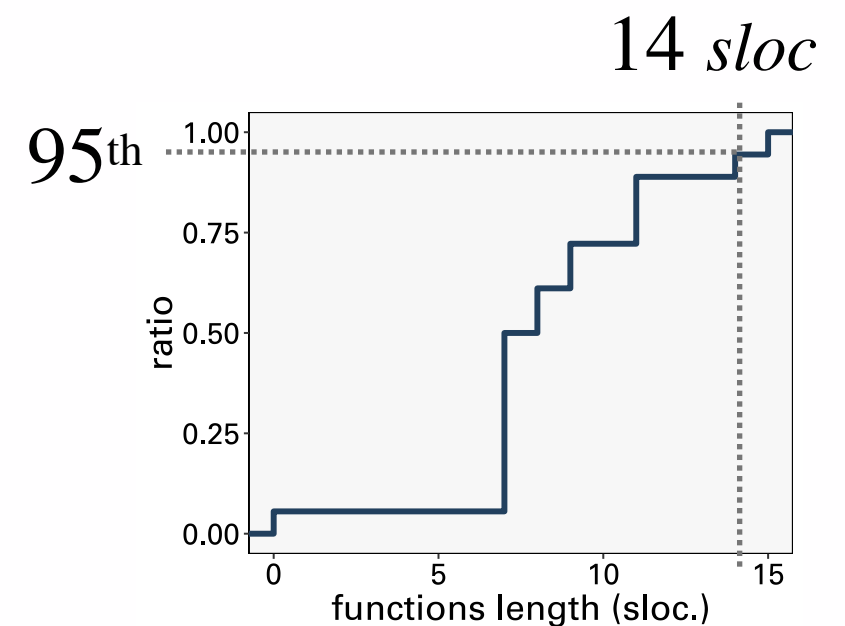
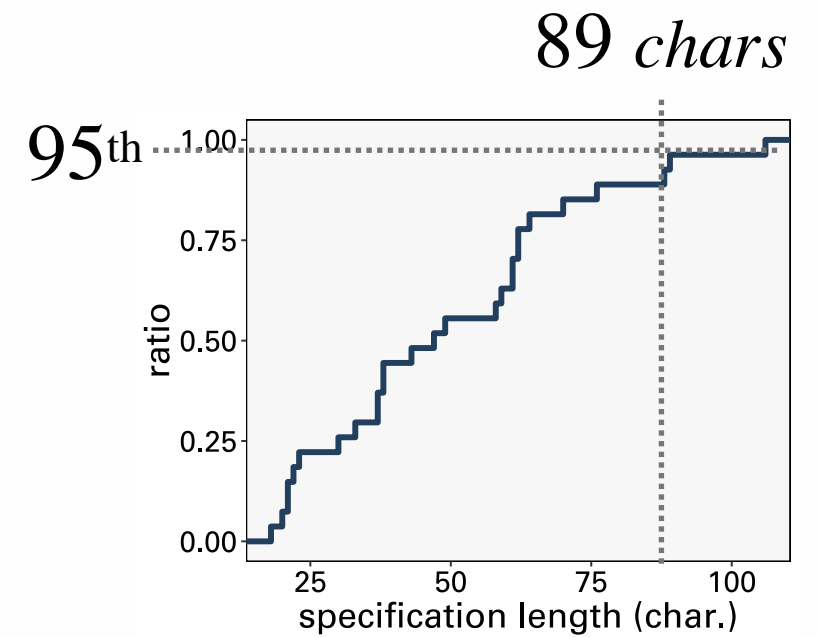
First order logic is effective
Easy to read

short functions

Reduce risks of bugs

small test campaigns

Inputs provided by the fuzzer
Expectation provided by the specification



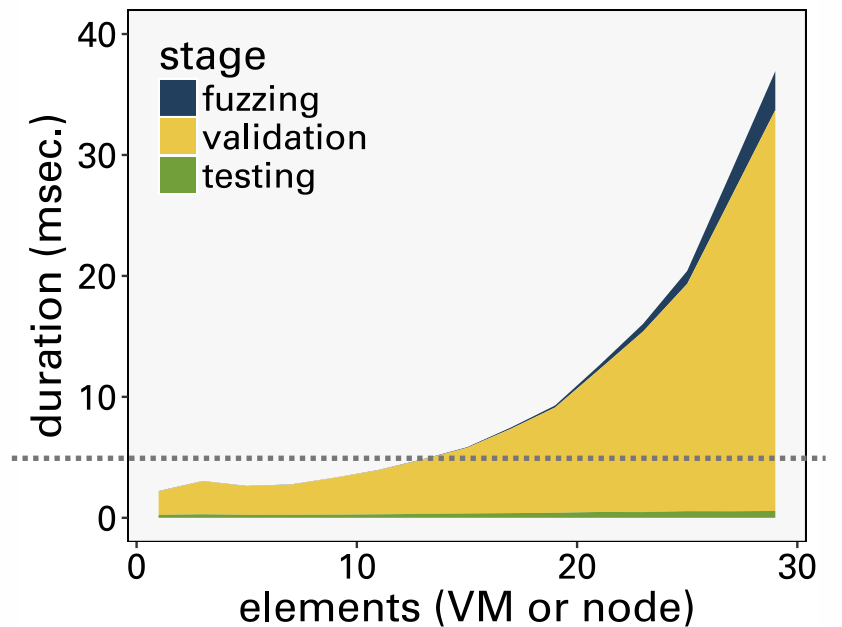
Fast enough for live testing

fuzzing: test case generation

validation: checking test case consistency wrt. core constraints

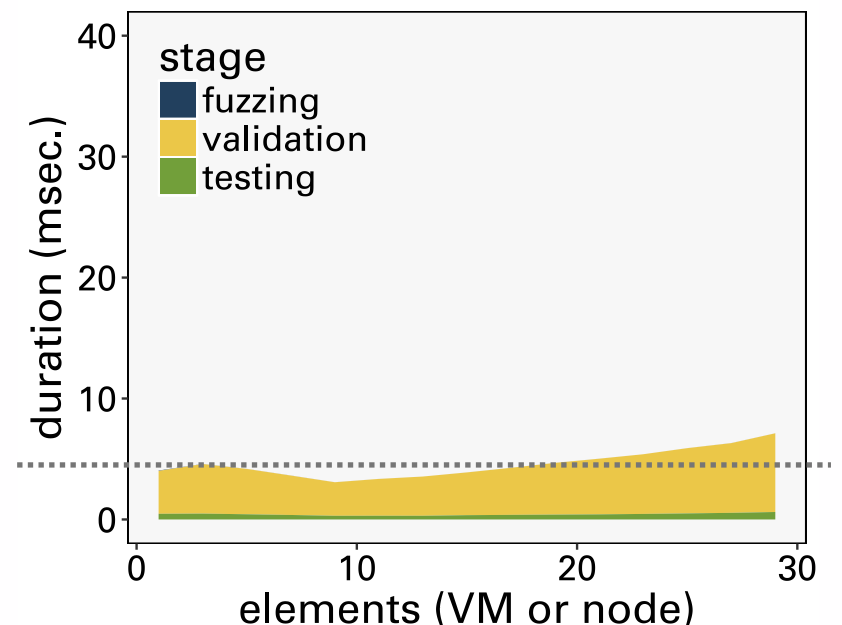
testing: checking the constraint under test

200
tests/sec



Fuzzer tuning to speed up the validation phase

200
tests/sec



Testing BtrPlace

22 constraints

1,000 non-unique tests per campaign

| Cause | Constraints | Tests |
|---------------------------------------|-------------|-------|
| Initial violation in continuous mode | 7 | 704 |
| Unexpected arguments | 4 | 642 |
| Discrete filtering in continuous mode | 3 | 45 |
| Unsupported action synchronisation | 4 | 20 |
| Bad action semantic comprehension | 1 | 16 |
| Unconsidered initial element state | 1 | 4 |

Exhibit known and unknown bugs

Lead to under-filtering (57%), over-filtering, crashes

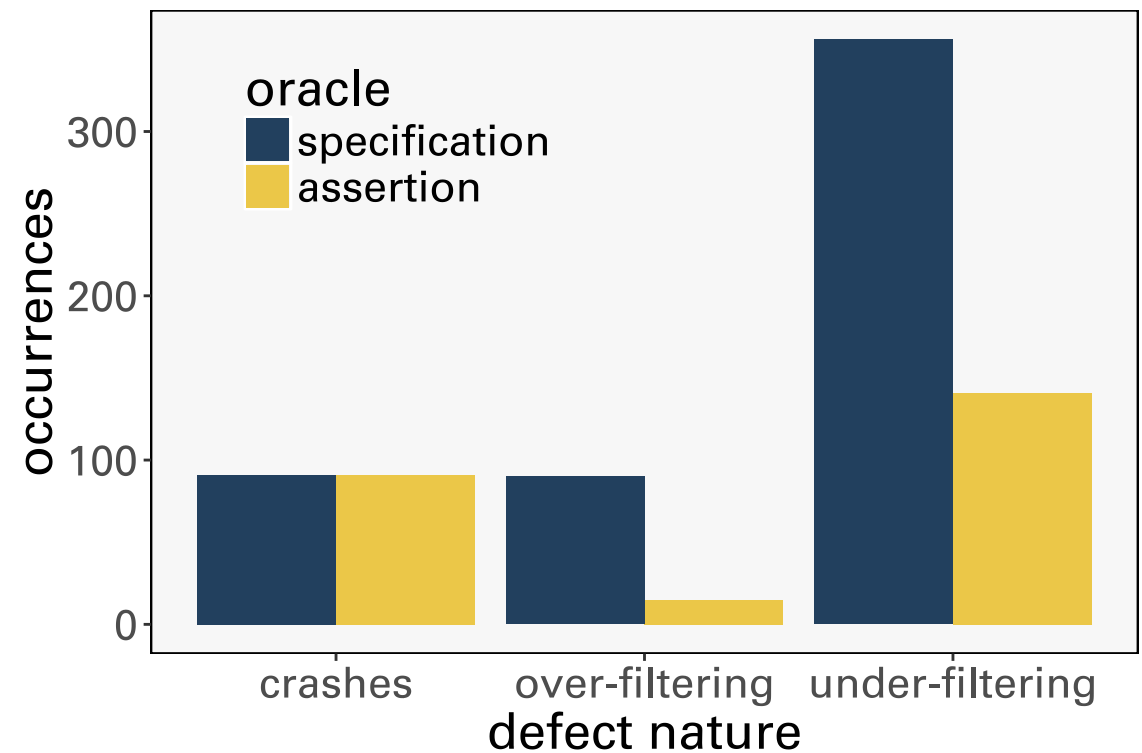
Specification vs. btrplace assertions

Assertion system

Written by the developer

Event based

Verbose



Programmatic approach is error prone

Developers forgot about action interleaving

Reasoning bugs cannot be exhibited
through regular testing methods



A concise DSL to specify the constraint invariants

Fuzz testing to detect inconsistencies

Non disruptive

Exhibit representative reasoning issues

Read the paper for more details and evaluation results

<http://www.btrplace.org>