Searching Architecture Models for Proactive Software Diversification

Benoit Baudry

joint work with J. Bourcier, F. Fouquet, S. Allier, M. Monperrus

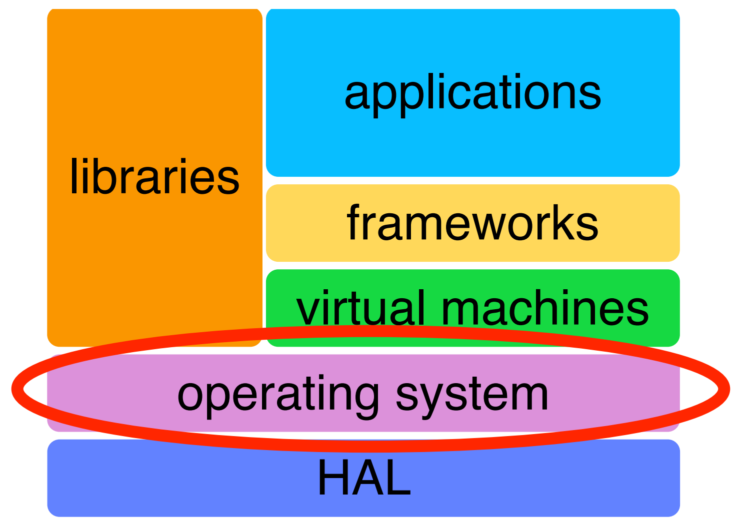
SEVENTH FRAMEWORK PROGRAMME

DIVERSIFY

Early software monocultures

Software monoculture

- Massive monoculture at the bottom of the software stack
  - operating system, web servers
- Emerged with the increase of the software market
  - personnal computers
  - Internet



Software monoculture – PC

Microsoft

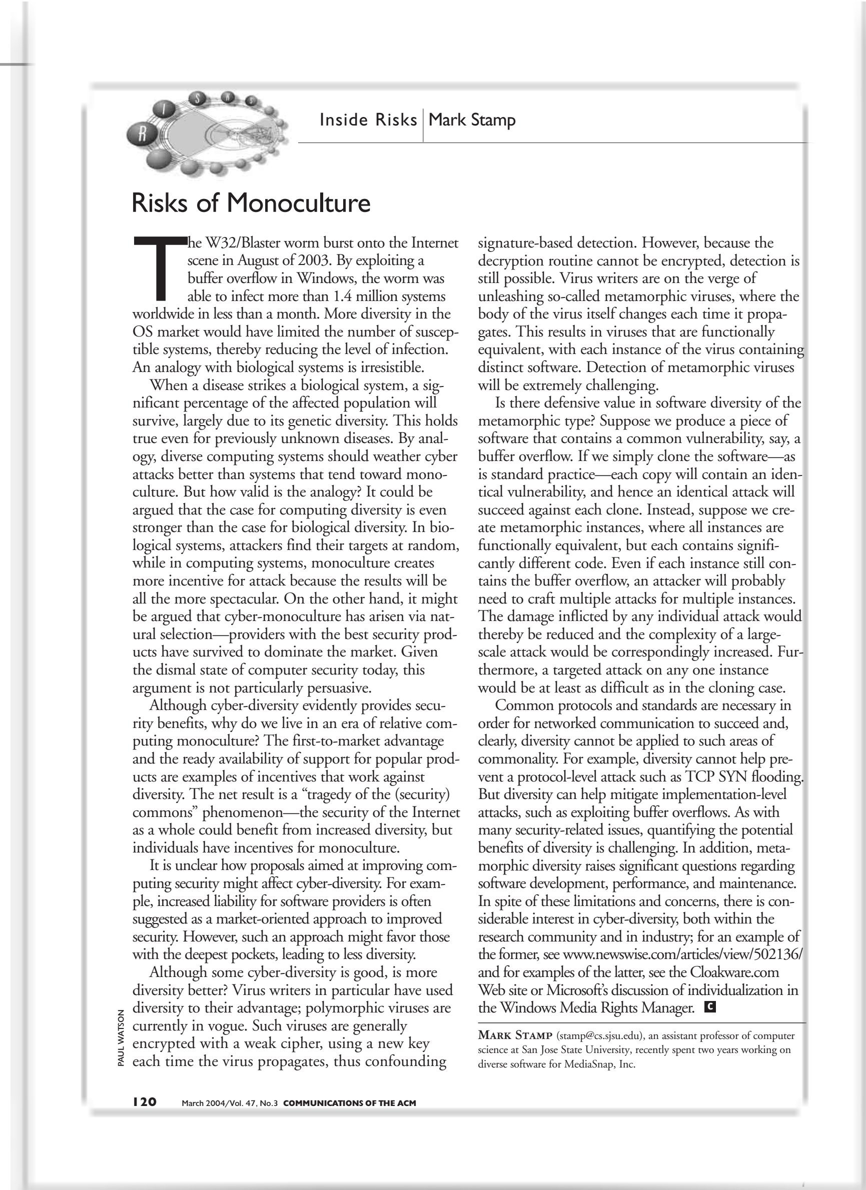
Software monoculture – web servers

Software monoculture – routers

CISCO ™

Risks very well known

- Single point of failure
- Cascading effects
  - error / virus propagation
- BOBE
  - blow one, blow everything
- Massive reuse of attack vectors


Inside Risks | Mark Stamp

Risks of Monoculture

The W32/Blaster worm burst onto the Internet scene in August of 2003. By exploiting a buffer overflow in Windows, the worm was able to infect more than 1.4 million systems worldwide in less than a month. More diversity in the OS market would have limited the number of susceptible systems, thereby reducing the level of infection. An analogy with biological systems is irresistible.

When a disease strikes a biological system, a significant percentage of the affected population will survive, largely due to its genetic diversity. This holds true even for previously unknown diseases. By analogy, diverse computing systems should weather cyber attacks better than systems that tend toward monoculture. But how valid is the analogy? It could be argued that the case for computing diversity is even stronger than the case for biological diversity. In biological systems, attackers find their targets at random, while in computing systems, monoculture creates more incentive for attack because the results will be all the more spectacular. On the other hand, it might be argued that cyber-monoculture has arisen via natural selection—providers with the best security products have survived to dominate the market. Given the dismal state of computer security today, this argument is not particularly persuasive.

Although cyber-diversity evidently provides security benefits, why do we live in an era of relative computing monoculture? The first-to-market advantage and the ready availability of support for popular products are examples of incentives that work against diversity. The net result is a "tragedy of the (security) commons" phenomenon—the security of the Internet as a whole could benefit from increased diversity, but individuals have incentives for monoculture.

It is unclear how proposals aimed at improving computing security might affect cyber-diversity. For example, increased liability for software providers is often suggested as a market-oriented approach to improved security. However, such an approach might favor those with the deepest pockets, leading to less diversity.

Although some cyber-diversity is good, is more diversity better? Virus writers in particular have used diversity to their advantage; polymorphic viruses are currently in vogue. Such viruses are generally encrypted with a weak cipher, using a new key each time the virus propagates, thus confounding signature-based detection. However, because the decryption routine cannot be encrypted, detection is still possible. Virus writers are on the verge of unleashing so-called metamorphic viruses, where the body of the virus itself changes each time it propagates. This results in viruses that are functionally equivalent, with each instance of the virus containing distinct software. Detection of metamorphic viruses will be extremely challenging.

Is there defensive value in software diversity of the metamorphic type? Suppose we produce a piece of software that contains a common vulnerability, say, a buffer overflow. If we simply clone the software—as is standard practice—each copy will contain an identical vulnerability, and hence an identical attack will succeed against each clone. Instead, suppose we create metamorphic instances, where all instances are functionally equivalent, but each contains significantly different code. Even if each instance still contains the buffer overflow, an attacker will probably need to craft multiple attacks for multiple instances. The damage inflicted by any individual attack would thereby be reduced and the complexity of a large-scale attack would be correspondingly increased. Furthermore, a targeted attack on any one instance would be at least as difficult as in the cloning case.

Common protocols and standards are necessary in order for networked communication to succeed and, clearly, diversity cannot be applied to such areas of commonality. For example, diversity cannot help prevent a protocol-level attack such as TCP SYN flooding. But diversity can help mitigate implementation-level attacks, such as exploiting buffer overflows. As with many security-related issues, quantifying the potential benefits of diversity is challenging. In addition, metamorphic diversity raises significant questions regarding software development, performance, and maintenance. In spite of these limitations and concerns, there is considerable interest in cyber-diversity, both within the research community and in industry; for an example of the former, see www.newswise.com/articles/view/502136/ and for examples of the latter, see the Cloakware.com Web site or Microsoft's discussion of individualization in the Windows Media Rights Manager. C

Mark Stamp (stamp@cs.sjsu.edu), an assistant professor of computer science at San Jose State University, recently spent two years working on diverse software for MediaSnap, Inc.

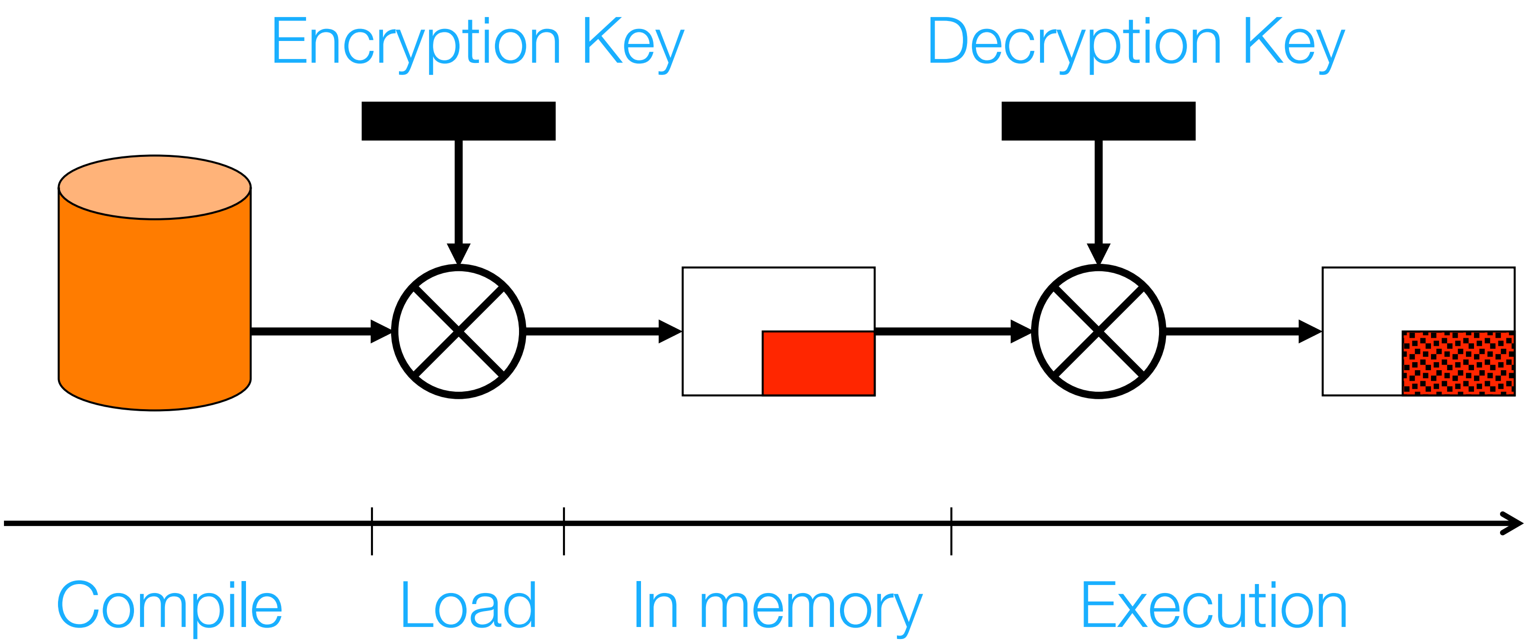
PAUL WATSON

Systems software diversification

Software diversity

- In operating systems
- Seminal papers in the 1990's
  - Fred Cohen 1993 « Operating system protection through program evolution »
  - Stephanie Forrest 1997 « Building Diverse Computer Systems »
- For security purposes
  - mitigate code injection, buffer overflows

Instruction set randomization



Randomized instruction set emulation. EG Barrantes, DH Ackley, S Forrest, D Stefanović. ACM TISSEC, 8 (1), 3-40

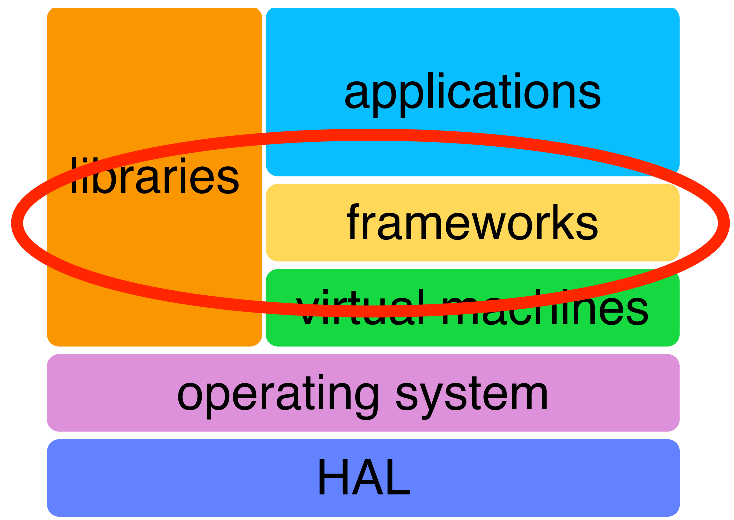
Software diversity

- Address space layout randomization
  - randomize binary addresses at load time
  - a program's address space is different on each machine
  - Deployed in all mainstream operating systems
  - Effective against buffer overflows

New software monocultures

Software monoculture today

- Continues growing in upper levels of the software stack
  - libraries, frameworks, IDEs, CMS, search engine, browser, etc.



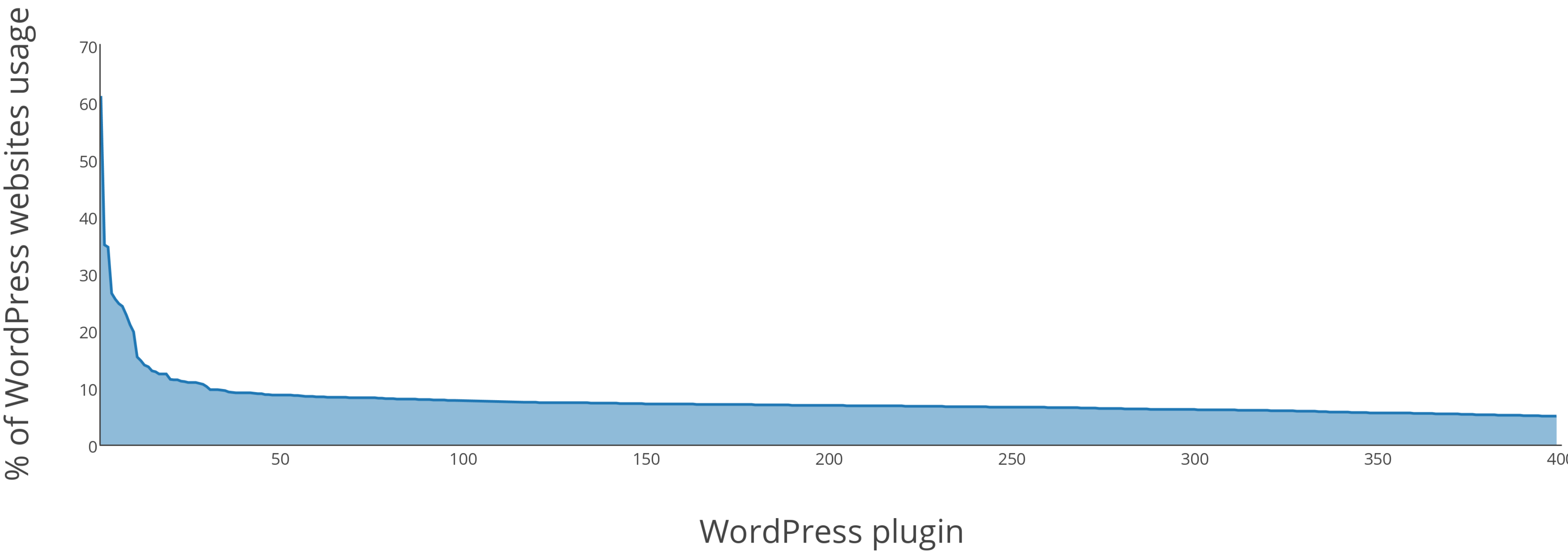
- Pushed by GOOD reasons
  - software engineering practices: modularity and reuse
  - compatibility and interoperability
  - maintenance and evolution costs reduction
  - economical motivations

The case of Wordpress

- CMS monoculture
  - March 2014: more than 20% of 500000 top site use Wordpress
- Plugins monoculture
  - 64% use the Akismet plugin
  - 23% use Jetpack, known to have an SQL injection vulnerability

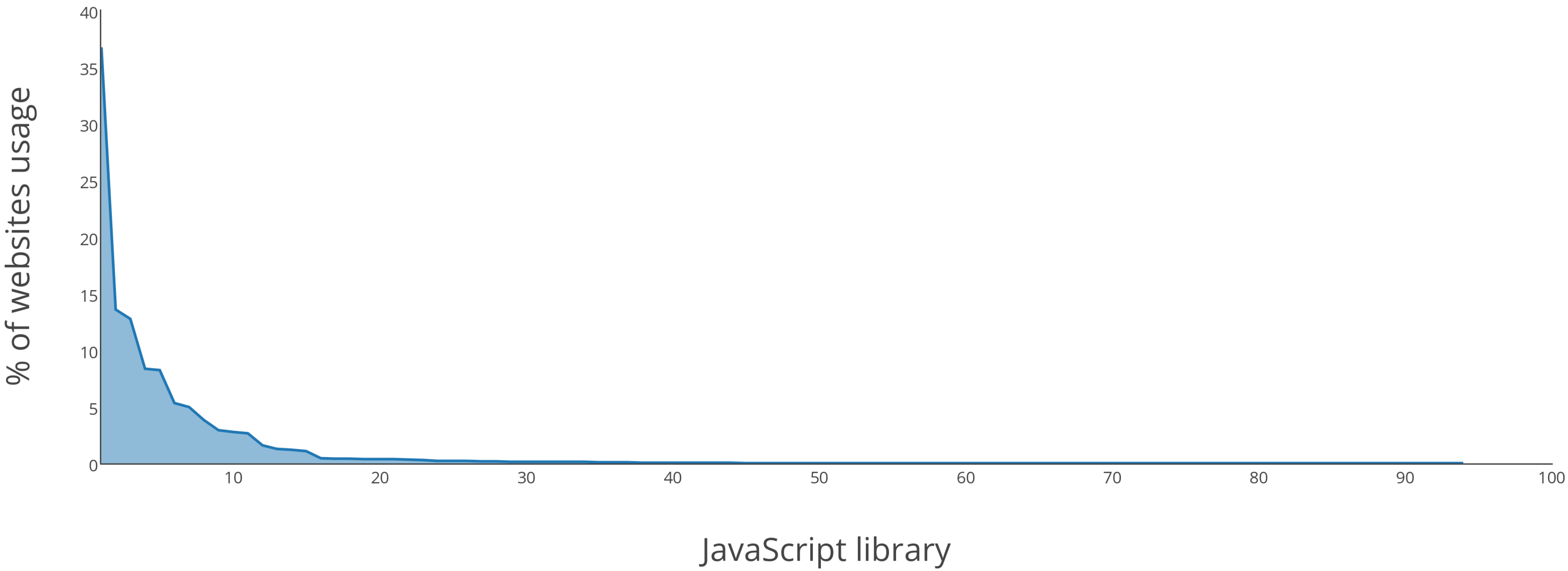
“Multi-tier diversification in Internet-based software applications”. Simon Allier, Olivier Barais, Benoit Baudry, Johann Bourcier, Erwan Daubert, Franck Fleurey, Martin Monperrus, Hui Song, Maxime Tricoire. To appear in IEEE Software, Jan 2015

The case of Wordpress



110000 web sites
mean of 5 plugins per site

JS libraries



110000 web sites

Cryptographic protocols

OpenSSL™

Cryptography and SSL/TLS Toolkit

Cryptographic protocols

source: https://t37.net/4-lessons-every-startup-should-learn-from-the-heartbleed-catastrophe.html

Cryptographic protocols

2. Monoculture kills

66% of the servers connected to the Internet use OpenSSL to provide https. This, like every kind of monoculture, is a real problem for many reasons.

Monoculture is a danger for security. It means security researchers will all focus on the same target instead of working on many. This was true in the late 80's and 90's when MS DOS and Windows were sharing 95% of the desktop market share: viruses coders were only targeting the platforms as they were easy to exploit, and most people were using them. It's still true today with Wordpress powering 12% of the sites worldwide, and using third party poorly coded if not willingly infected themes and plugins.

Monoculture is a danger for innovation. Innovation comes from diversity and the will to explore new, undiscovered paths. On the contrary, monoculture brings the "everyone does it this way, so we're doing it too" syndrome. I remember, 10 years ago when we were developing Web sites and applications for Internet Explorer 6 only. It was a real pain as alternate browsers were supporting new feature faster. For the record, Internet Explorer 6 was discontinued only 2 years ago.

Social networks

Is the social networking monoculture ready to crumble?

Summary: *The emergence of new social networking services such as Pinterest and a growing base of disgruntled 3rd party developers for the leading services shows that changes in the social networking industry are far from over. It's also causing a rethinking of the business models and partner ecosystems of what's become the old guard, Facebook and Twitter.*

By Dion Hinchcliffe for Enterprise Web 2.0 | August 28, 2012 -- 22:29 GMT (15:29 PDT)

Follow @dhinchcliffe

Get the ZDNet Insights newsletter now

source: http://www.zdnet.com/is-the-social-networking-monoculture-ready-to-crumble-7000003329/

Knowledge

Software development

Garann Means asks: are geek stereotypes and the prevailing developer monoculture putting the web industry at risk?

source: http://www.creativebloq.com/netmag/bacon-bad-you-dangers-dev-monoculture-21410684

Alternatives are emerging

Web servers

Microsoft Windows Server System

NGINX http server

Cloud platforms

amazon
webservices™

openstack™

Java virtual machines

Apps


Huge reservoir of functionally similar software solutions

Yet, software systems remain highly homogeneous

Take-away

- Software monocultures exist
  - at a very large scale
  - in application level code
- Software diversity exists
  - machine-code level
- Alternative software solutions emerge
  - must be exploited
- Next challenge: diversify applications in a proactive/automatic way

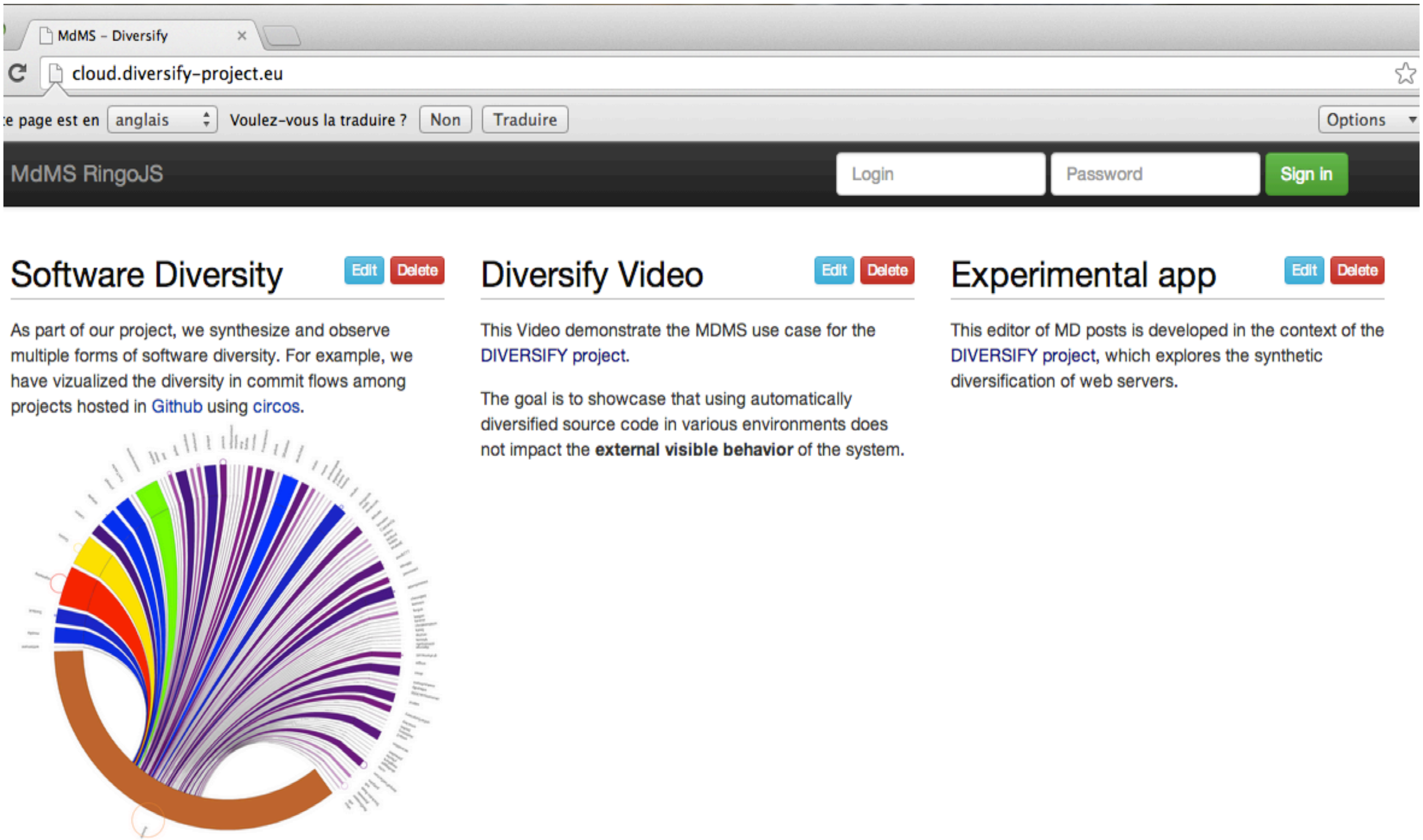
Our claim

MDE and SBSE

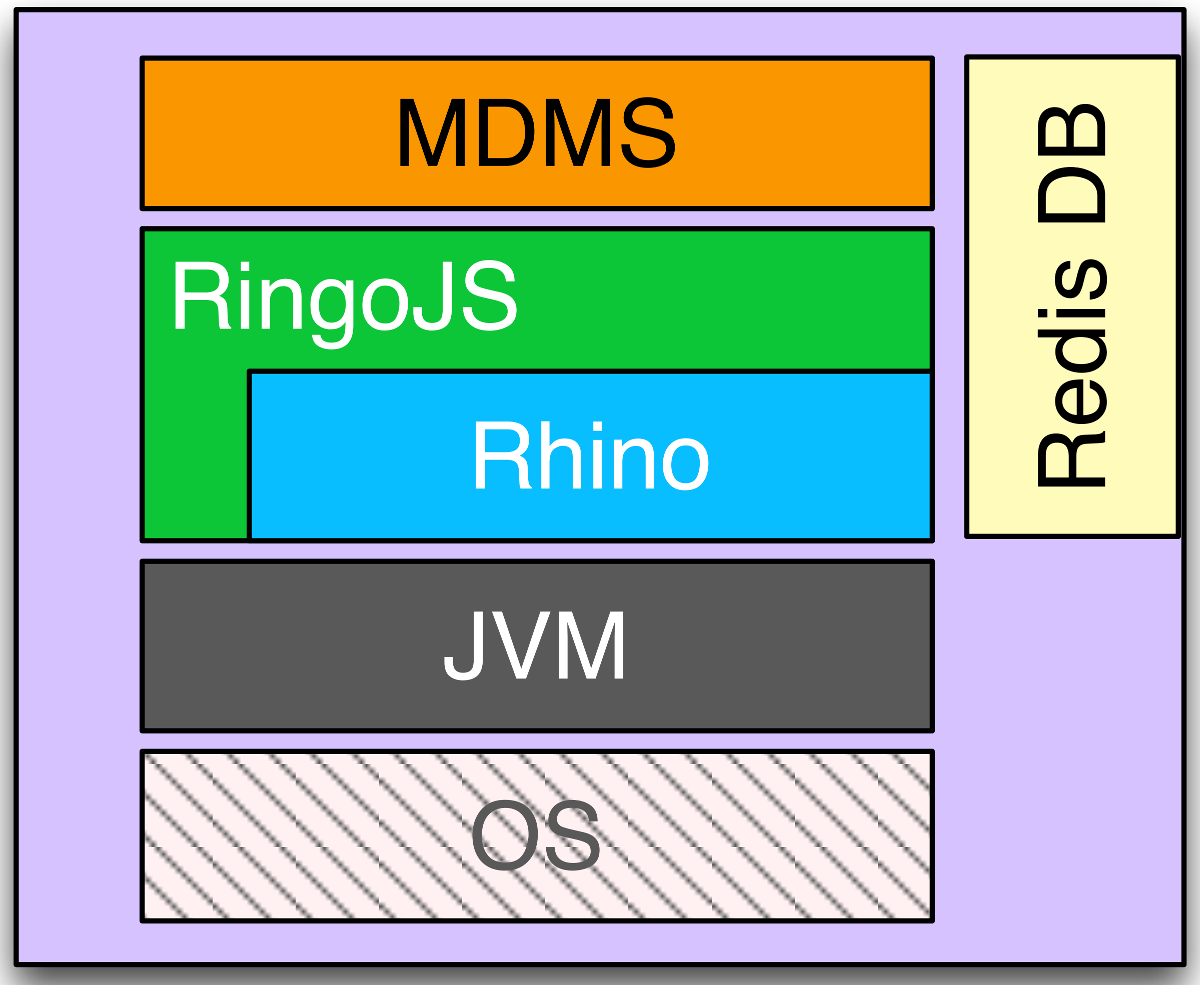
can spur

aplication software diversity radiation

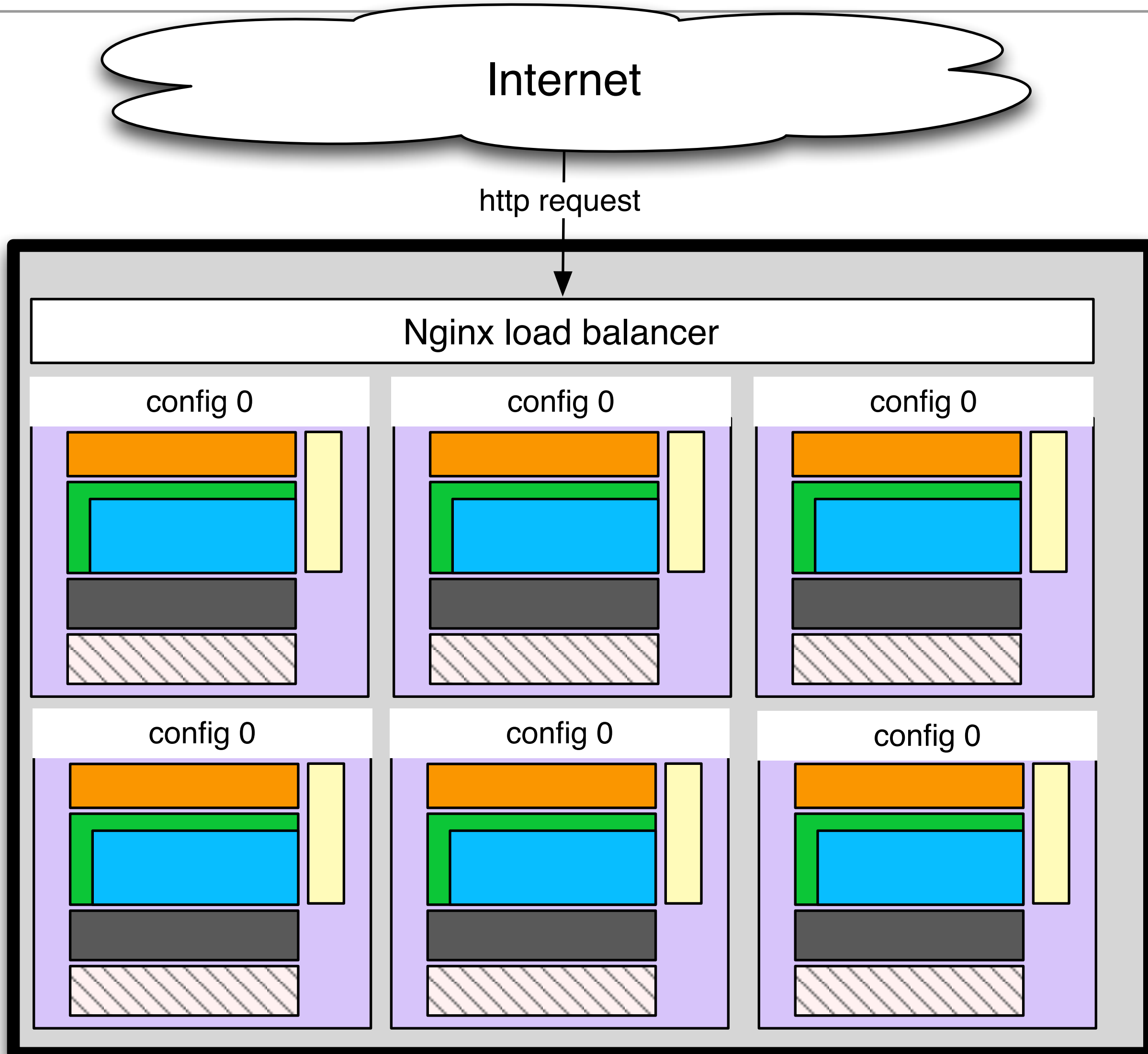
Web app example



Server side software stack

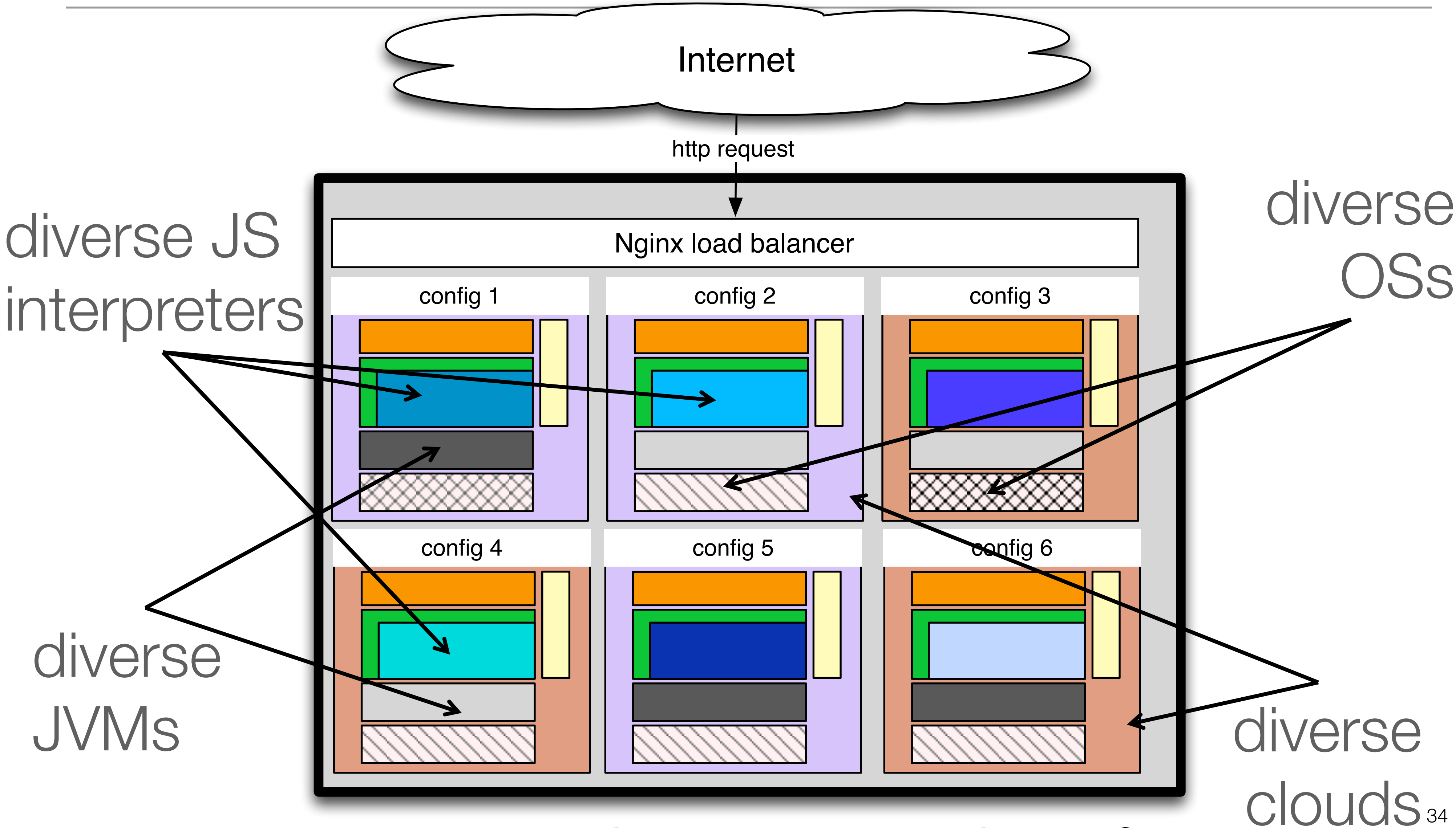


Server side deployment



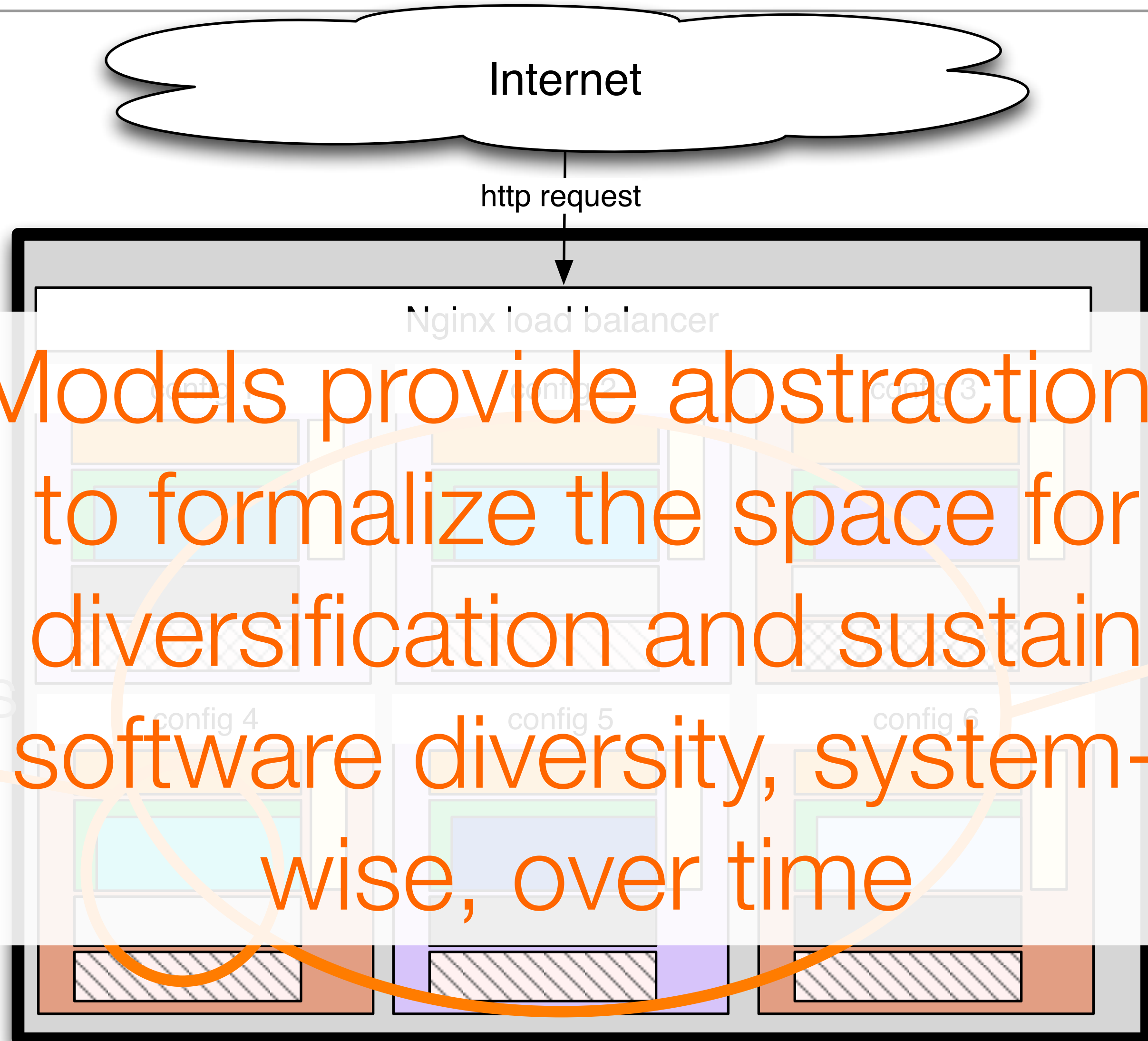
Monoculture deployment of MDMS

Server side deployment



Multi-diversified deployment of MDMS

Where models can support diversification



Models provide abstractions to formalize the space for diversification and sustain software diversity, system-wise, over time

trade-off between diversity and other criteria

formal dependencies

Multi-diversified deployment of MDMS

Searching for diverse architectures

- A reservoir of software diversity
  - natural diversity of OS or JVM
  - automatic diversification of the JS interpreter
- Automatic reasoning on the architecture
  - find valid, diverse deployment architectures
- Actual deployment of a diverse architecture
  - deploy the solution on a distributed setting

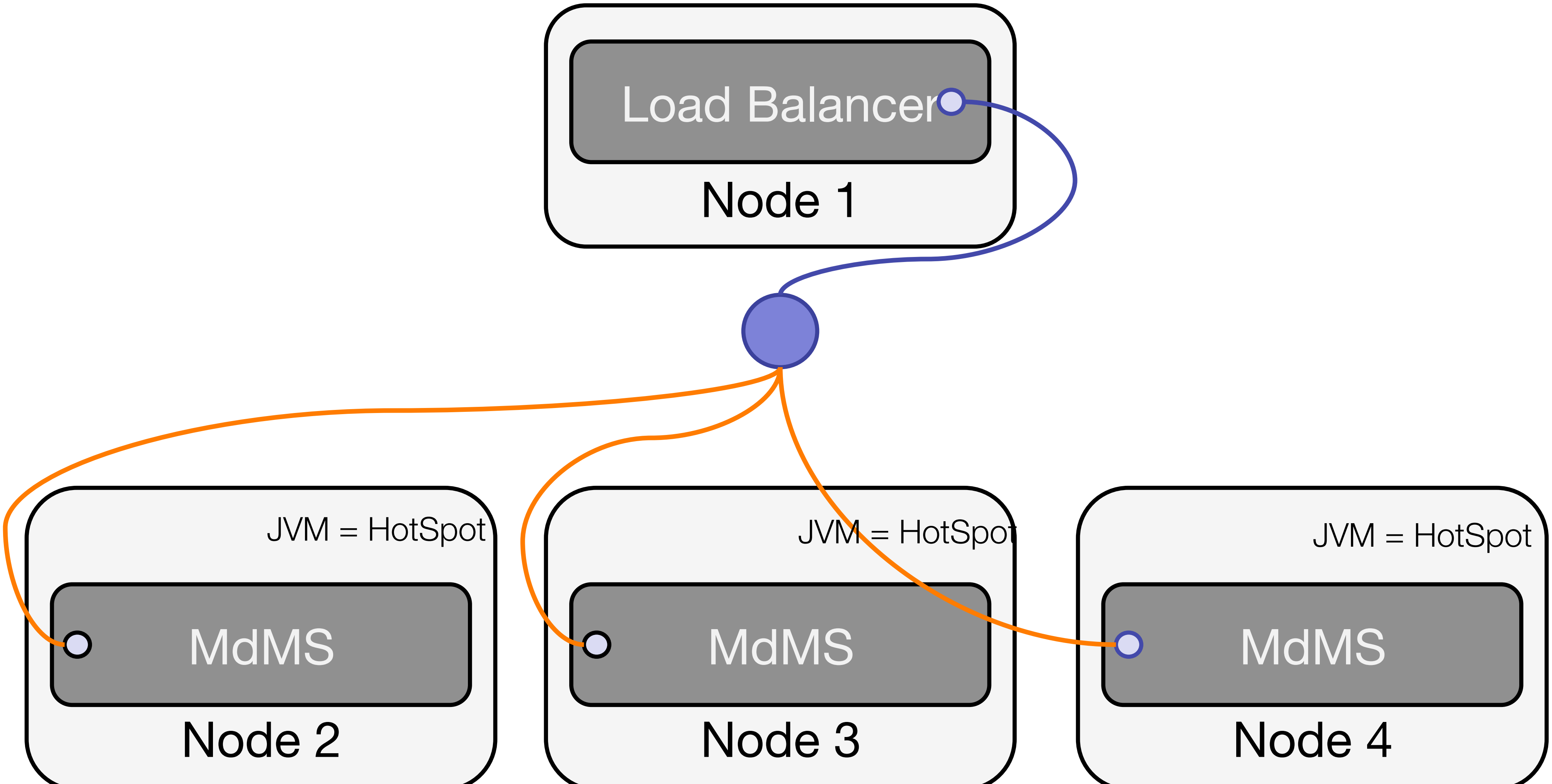
Synthesizing a diversity reservoir

- Sosies
  - a sosie program is a variant of a program that passes the same test suite
- Synthesized thousands of sosies
  - deleting or adding / replacing statements by others from the same program
- Synthesized 843 RingoJS sosies
  - that can be executed from the MDMS client

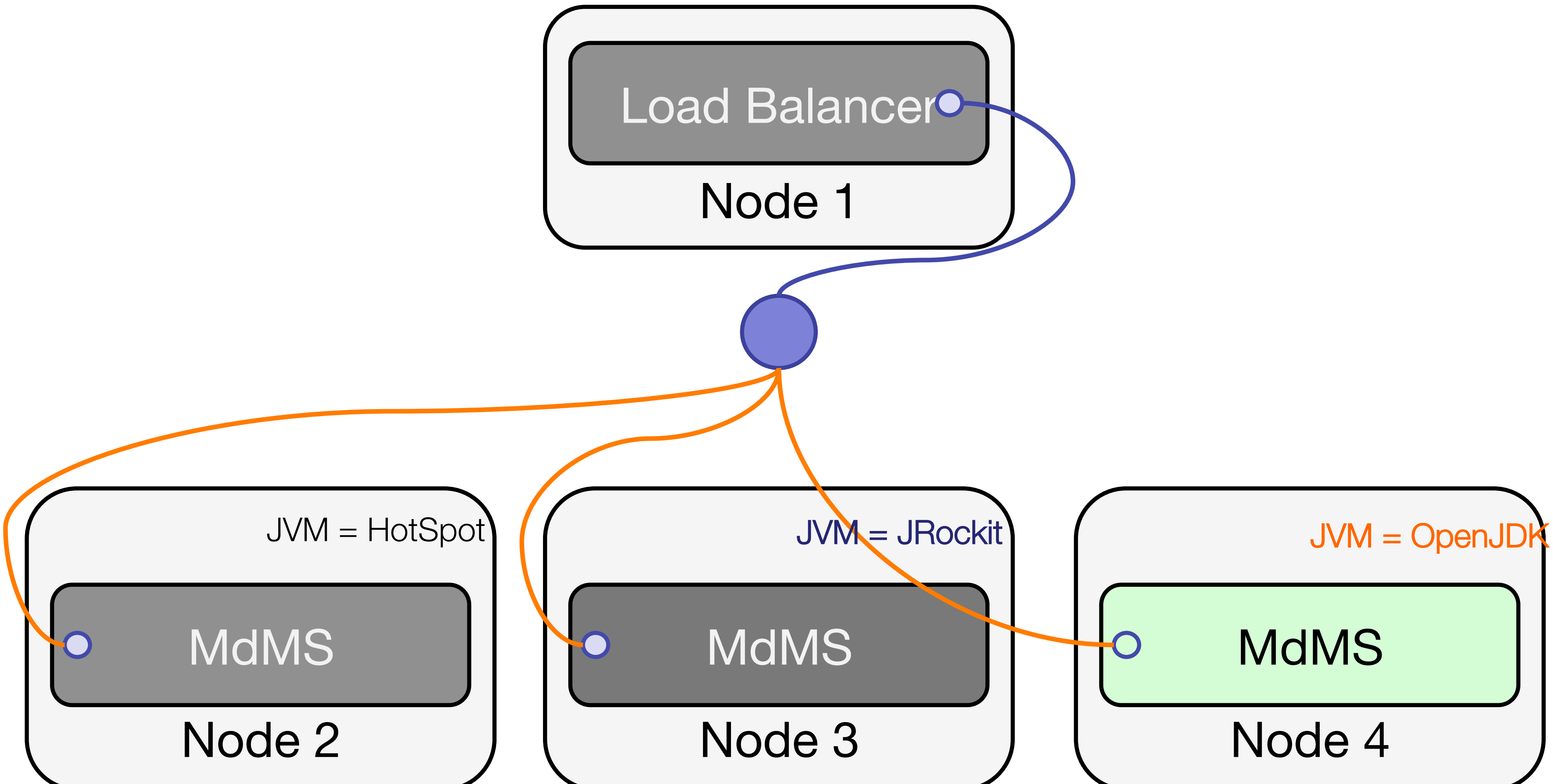
"Tailored Source Code Transformations to Synthesize Computationally Diverse Program Variants". Benoit Baudry Simon Allier, Martin Monperrus. ISSTA 2014

Architecture modeling

- Component-based software engineering

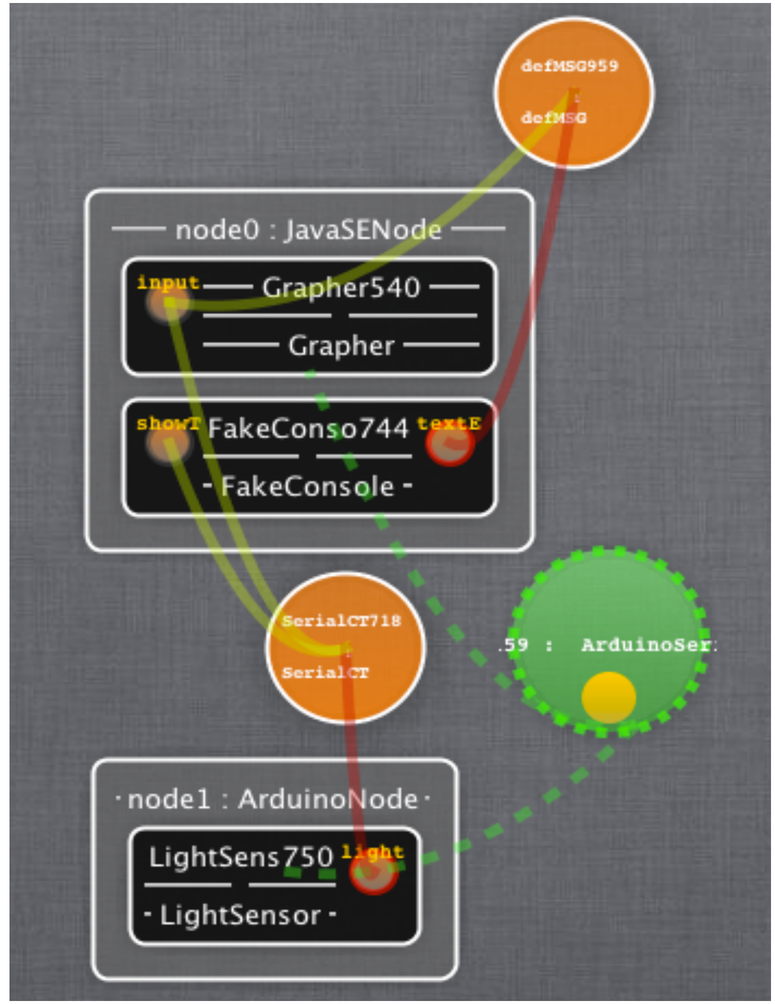


Architecture modeling



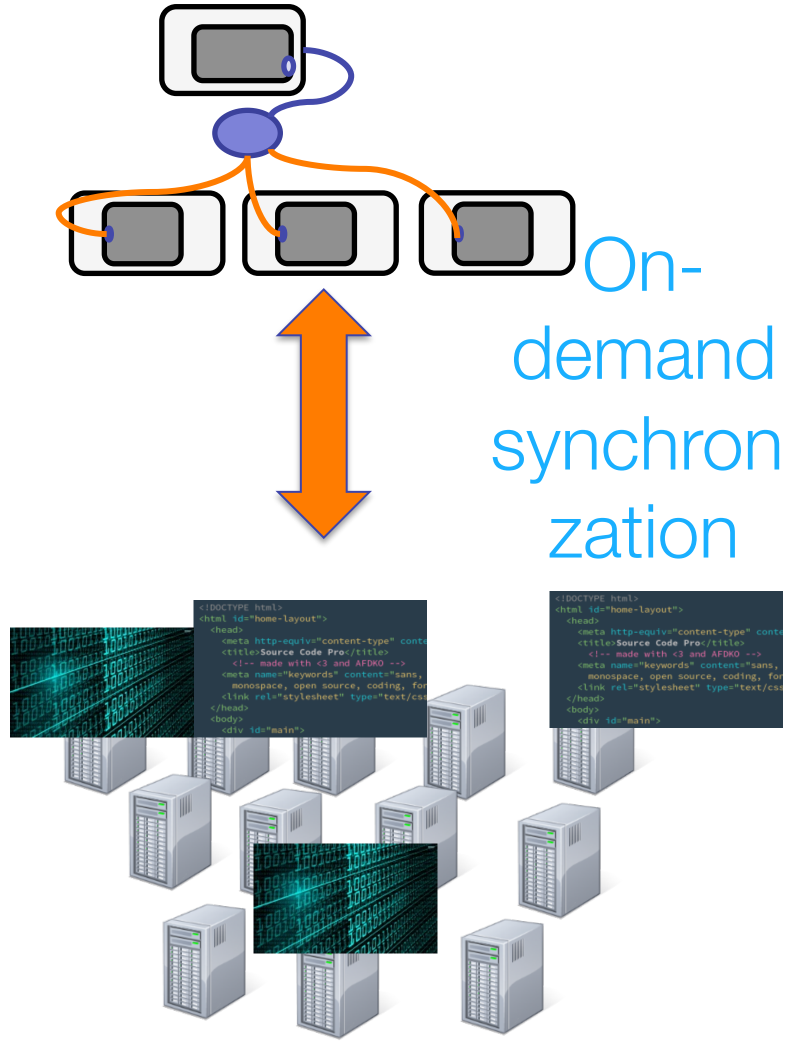
Architecture modeling

- Component
  - Code unit
  - I/O ports
- Channel
  - Communication between components
- Node
  - Execution platform for components
- Group
  - Group nodes together to have a consistent model

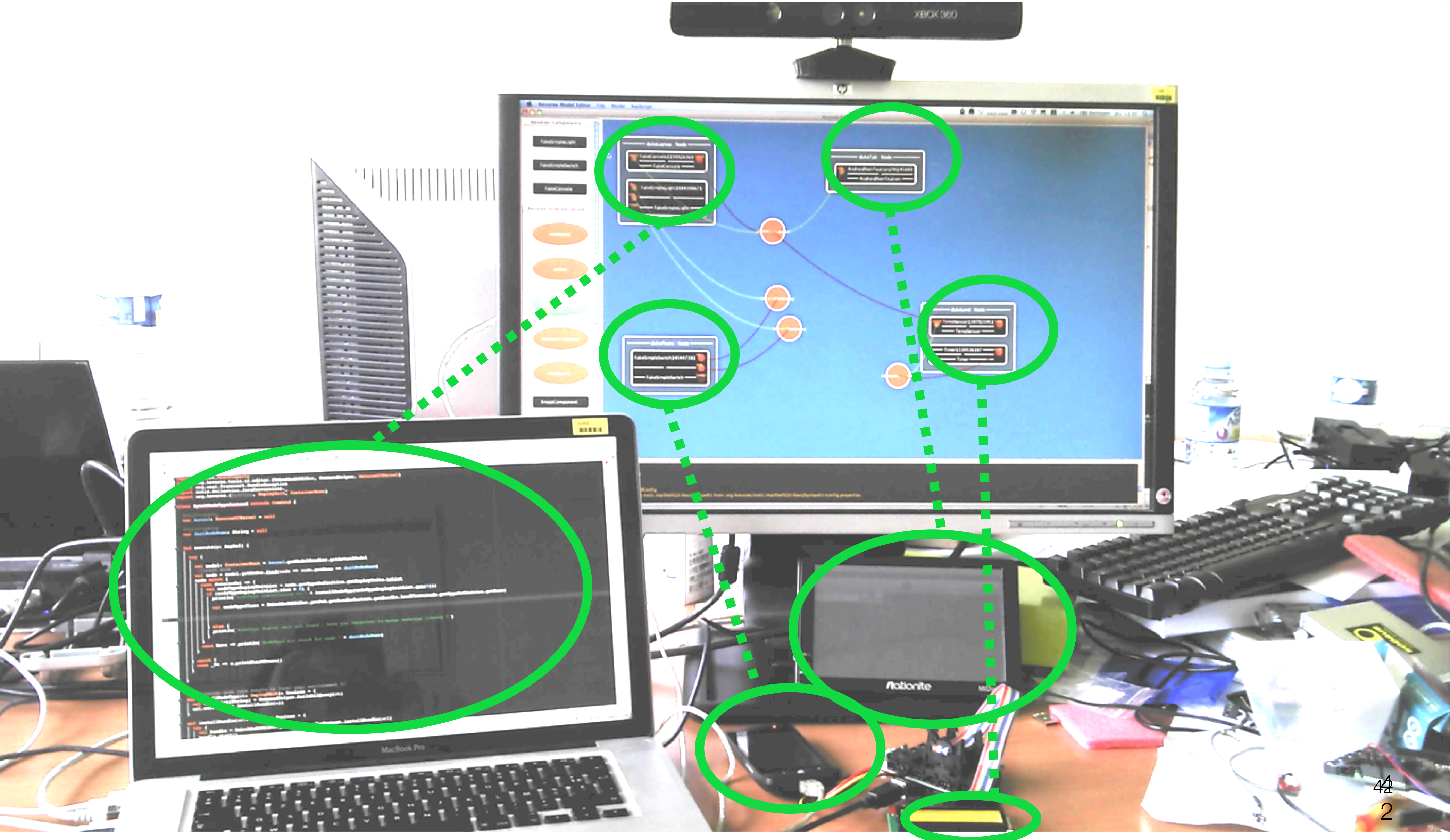


Kevoree for distributed deployment

- An open-source framework
- A structural model that represents the distributed running system and that can be synchronized in both directions on-demand



Kevoree for distributed deployment



Synthesizing software architecture

- Given a reservoir of diverse software components
  - natural diversity of VMs, JVMs, machines
  - automatic diversity: sosie RingoJS
- What is the the good trade-off between
  - capacity
  - cost
  - diversity: need to estimate ‘good’ diversity

Polymer Framework

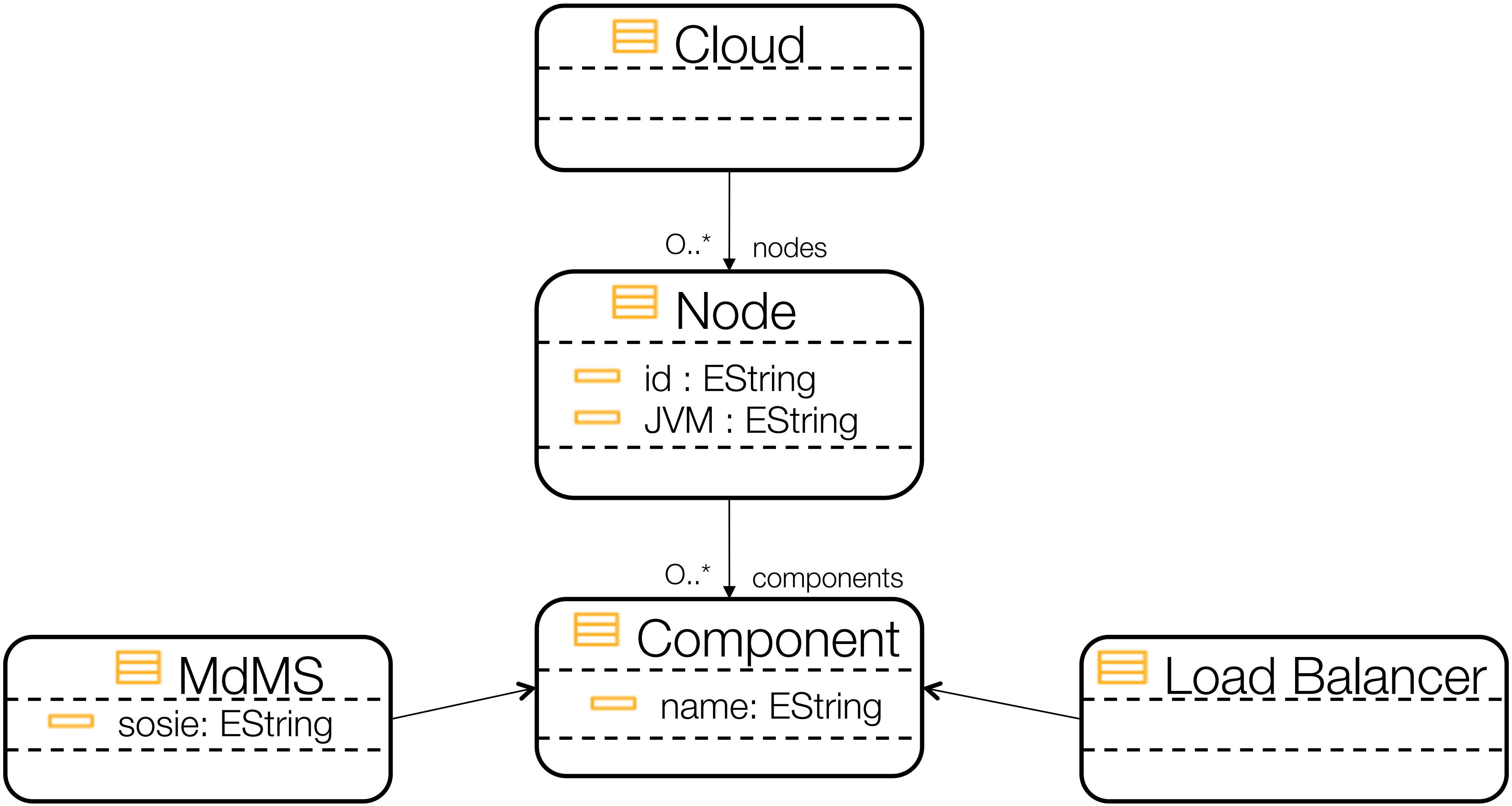
- Polymer
  - Open-source framework to enable runtime usage of SBSE techniques
- Works to make SSBSE usable @Runtime
  - Define dedicated domains, actions, fitness
  - Find heuristics to converge faster to acceptable tradeoffs

Polymer

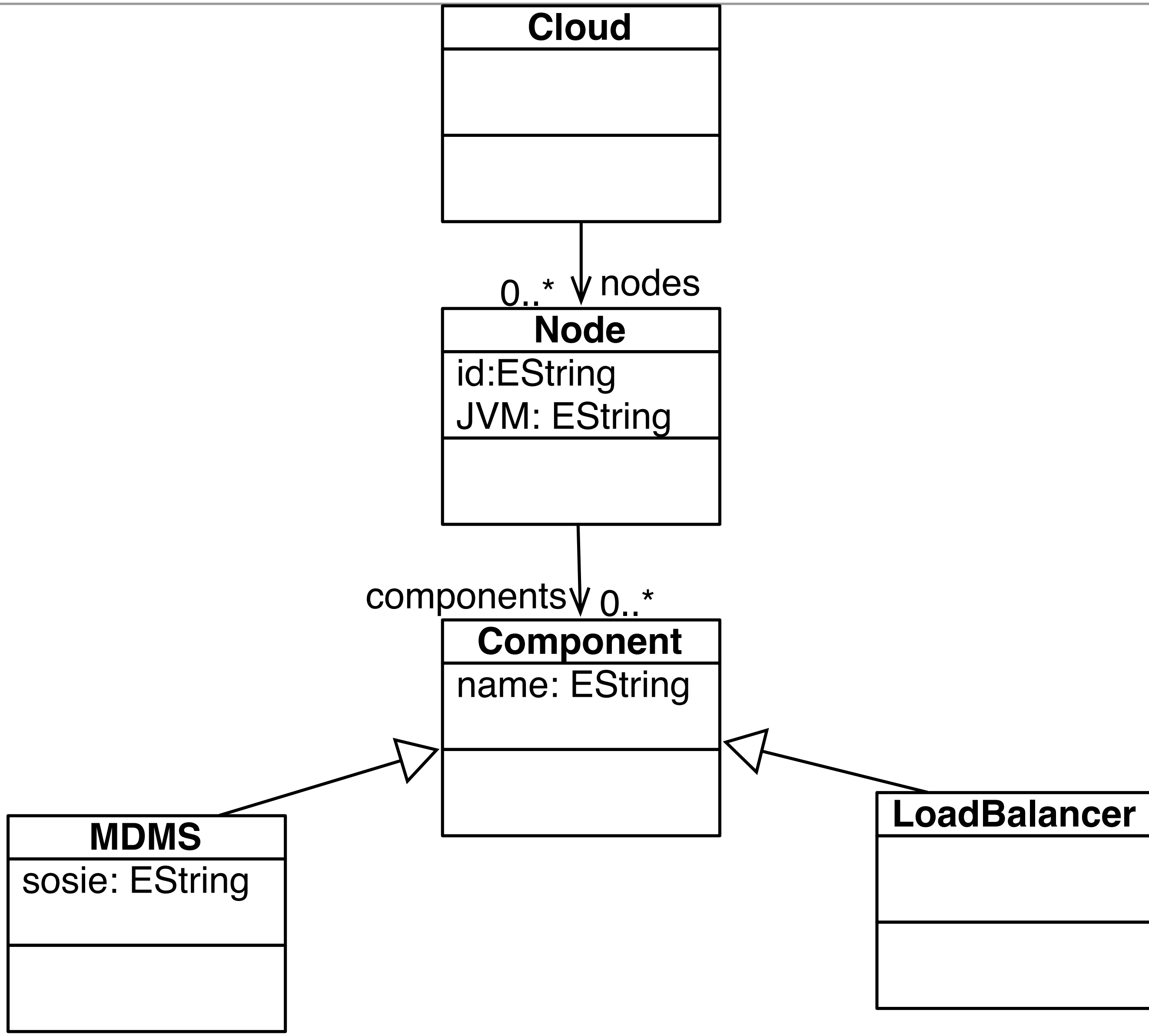
- Leverage the KMF framework to reason on domain models
- Mutation, Fitness, and crossover are defined as model transformation
- Multi-objectives
- Extensible framework
  - Define your own model, your own operators, your own fitnesses, define your own search algorithm

Implemented algorithms : Genetic (MOEAd, NSGAII), Greedy (progression each steps), Local Full Search

Concrete domain example



Domain model



Polymer usage

```
GeneticEngine<Cloud> engine = new GeneticEngine<Cloud>();
```

The model to use

```
engine.setAlgorithm(GeneticAlgorithm.EpsilonCrowdingNSGII);
```

The Search algorithm to use

```
engine.addOperator(new AddNodeMutator());
engine.addOperator(new RemoveNodeMutator());
engine.addOperator(new AddComponentMutator());
…
```

The mutation operators to use

```
engine.addFitnessFuntion(new CloudCostFitness());
engine.addFitnessFuntion(new CloudCapacityFitness());
engine.addFitnessFuntion(new CloudDiversityFitness());
…
```

The fitnesses to use

```
engine.setMaxGeneration(300);
```

Fix search parameters

```
engine.run();
```

Run

Defining Mutation

Usage of model elements

```
...
cloud.getNodes().add(new Nodes().setName("node_5555"));
...
```

Defining the cost Fitness

```
function evaluate(cloud : CloudModel) : Double {
...
        return sum(cloud.getNodes.price);
...
}
```

Usage of model elements

Defining the capacity Fitness

```
function evaluate(cloud : CloudModel) : Double {
...
        return sum(cloud.getNodes.capacity);
...
}
```

Usage of model elements

Defining the Diversity Fitness

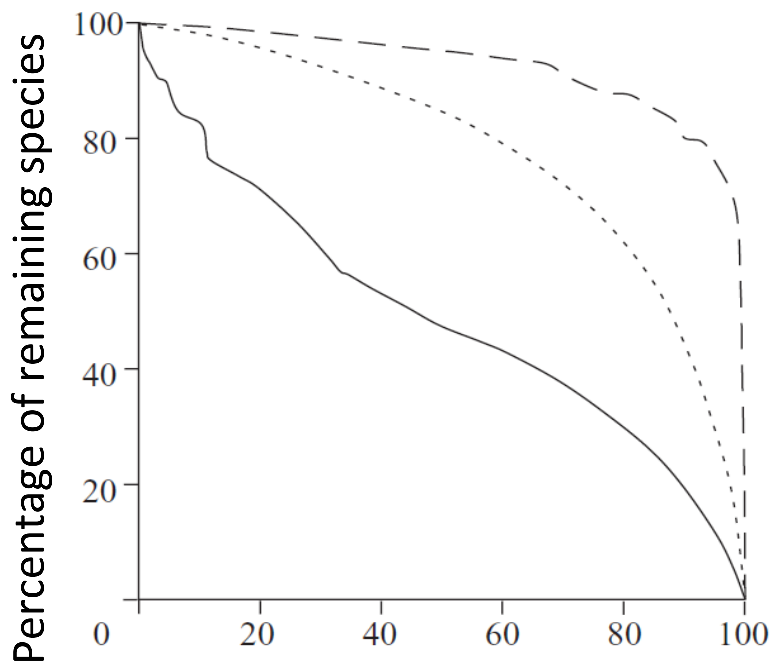
```
function evaluate(cloud : CloudModel) : Double {
...
        return extinctionSequence(cloud);
...
}
```

Usage of model elements

This function computes a value corresponding to the extinction sequence of the cloud given in parameters

Diversity fitness: robustness

Robustness: how fast extinctions lead to collapse of other species (secondary extinctions)



Percentage of plant species deleted:
extinction sequence

Memmott et al. 2004

Extinction sequence algorithm

1. While the application still provides a service
    1. We select a specific component A
        - Such as a specific sosies of MdMS
    2. We kill all the instances of A
    3. We evaluate the capacity of the system to serve user request and incrementally draw a curve
2. We measure the area behind the curve to determine the robustness of the system

Robustness Measurement

- Quantifies the resistance of the graph to random perturbations
- Reports the change in apps alive as the platforms are individually killed
- Robust networks allow the maximum amount of apps to remain alive in the face of systemic platform death

Conclusion

- Software monocultures grow at all levels in software stacks
  - for good engineering and business reasons
- MDE and SBSE can be key enablers to balance this natural phenomenon
  - abstractions that characterize the diverse components
  - search-based techniques that sustain diversity

References

- « Multi-tier diversification in Internet-based software applications ». Simon Allier, Olivier Barais, Benoit Baudry, Johann Bourcier, Erwan Daubert, Franck Fleurey, Martin Monperrus, Hui Song, Maxime Tricoire. To appear in IEEE Software, Jan 2015
- « Tailored source code transformations to synthesize computationally diverse program variants ». Benoit Baudry, Simon Allier, and Martin Monperrus. ISSTA 2014.
- « Optimizing Multi-Objective Evolutionary Algorithms to enable Quality-Aware Software Provisioning ». Donia El Kateb, Francois Fouquet, Johann Bourcier, Yves Le Traon. QSIC 2014
- http://kevoree.org/polymer/
- http://kevoree.org/
- https://github.com/INRIA/spoon
- http://diversify-project.eu/