A bird view on Software Evolution research

S. Ducasse
http://stephane.ducasse.free.fr

Software is Complex

Laws of software evolution

Continuing change

- A program that is used in a real-world environment must change, or become progressively less useful in that environment.

Increasing complexity

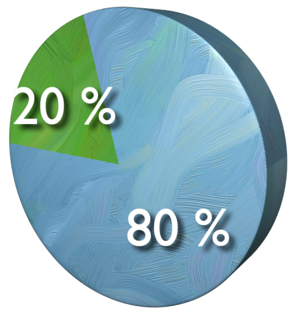
- As a program evolves, it becomes more complex, and extra resources are needed to preserve and simplify its structure.

Software is a living entity...

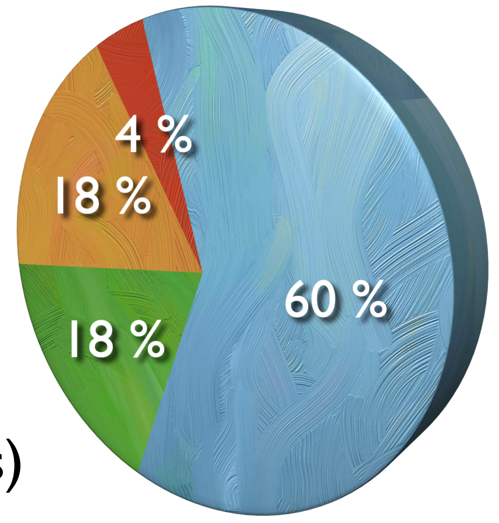
- Early decisions were certainly good at that time
- But the context changes
- Customers change
- Technology changes
- People change

We only maintain useful successful software

Maintenance is *continuous* Development



Between **70%** and **90%** of ***global*** effort is spent on "maintenance" !



4% Other

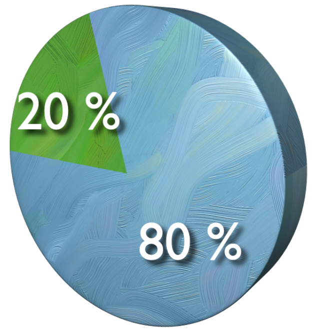
18% Adaptive
(new platforms or OS)

18% Corrective
(fixing reported errors)

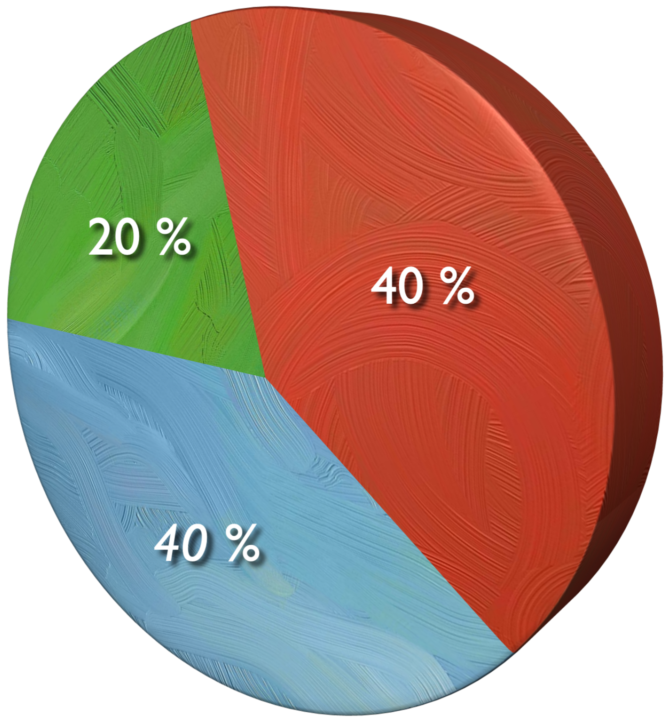
60% Perfective
(new functionality)

"Maintenance"

50% of development time is lost trying to understand code !



Between **50%** and **80%** of the ***overall cost is spent in the evolution***



We lose a lot of time with inappropriate and ineffective practices

Legacy systems exist in ***any*** language

Berger-Levrault by example

1 MLOCS

21 433 classes

95 164 méthodes

500 pages web

36 ans/homme de migration

Depuis GWT vers Angular

Bottom up team: interested in problems

code analysis, metamodeling, software metrics, program understanding, ***program visualization***, ***reverse engineering***, evolution analysis, refactorings, quality,

changes analysis, commit,

dependencies, merging support

rule and bug assessment

semi-automatic migration

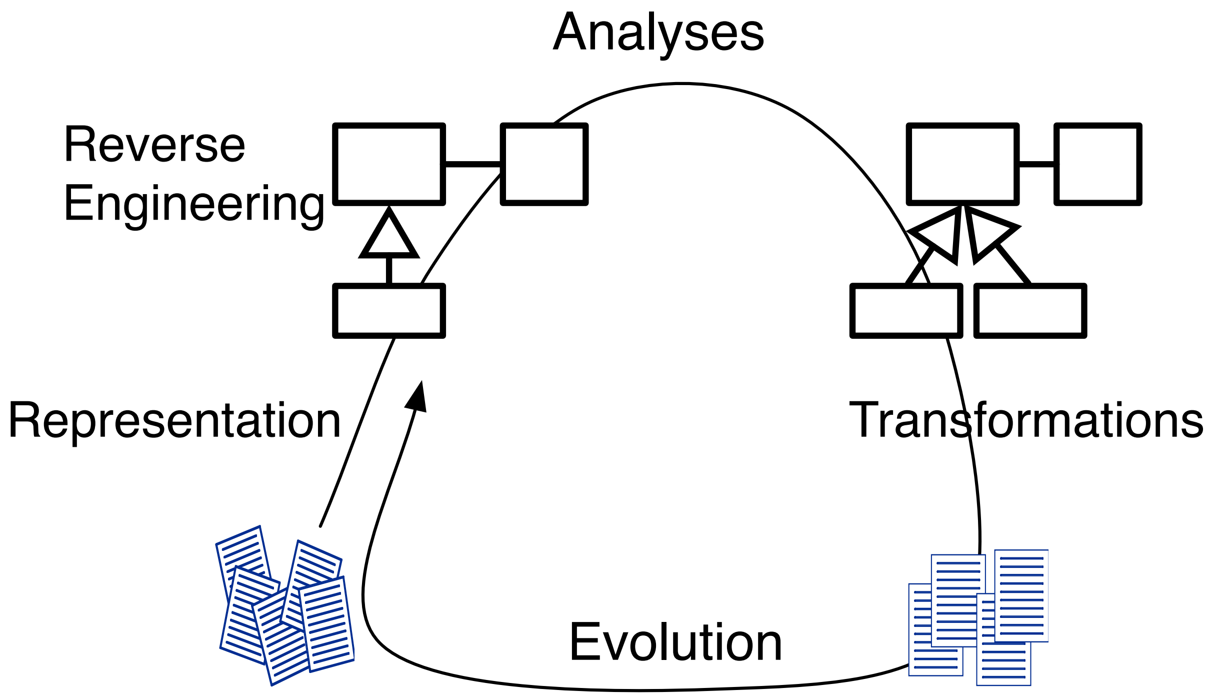
example-based transformations

test selection, rearchitecturing

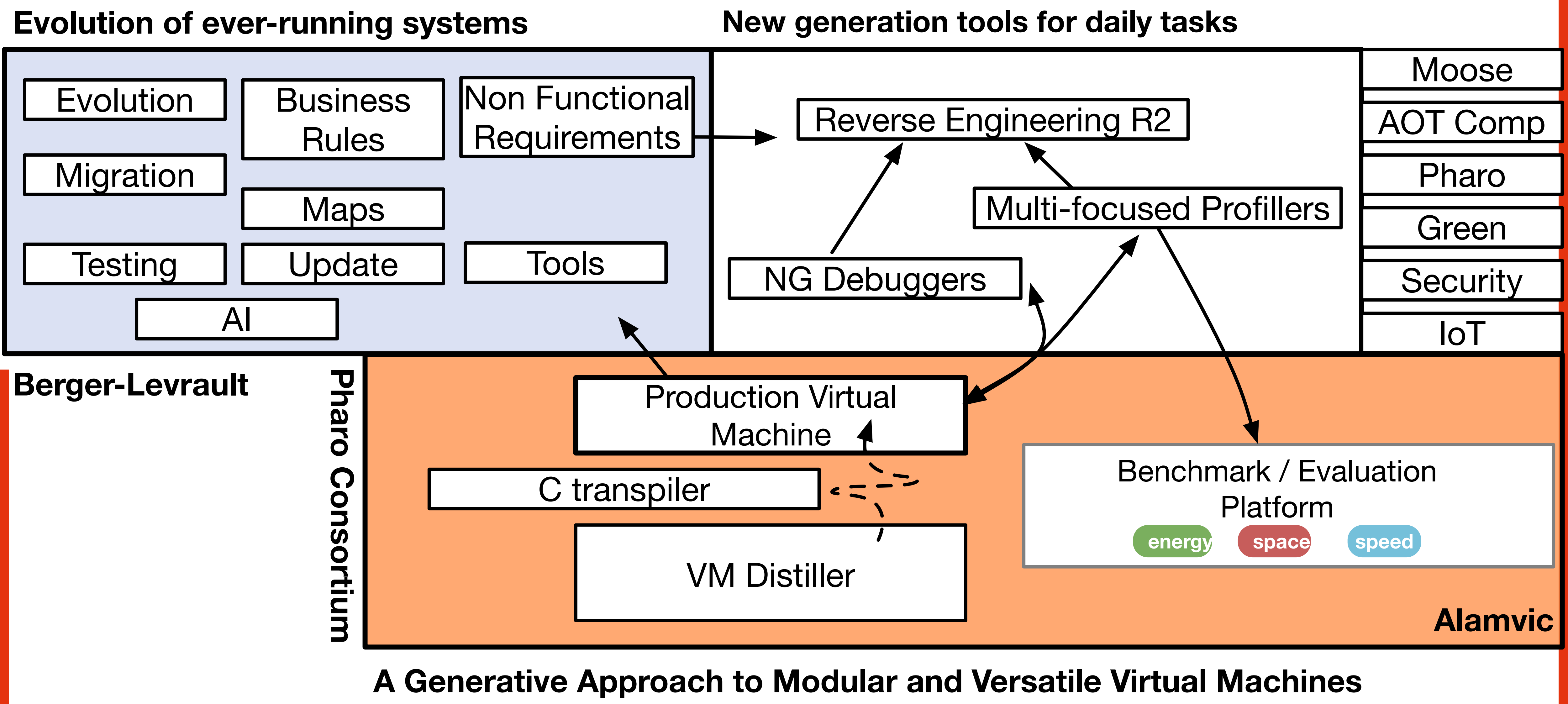
blockchains, ***ui-migration***

Collaborations

- IMT Douai, Soft (VUB), ENSTA (Bretagne)
- Berger-Levrault, Siemens, Thales, CIM, Arolla, Lifeware, WordLine/ATOS



RMOD: 3 axes in synergy



New generation tools

Debuggers (back in time, object-centric)

Profilers

Reverse engineering tools

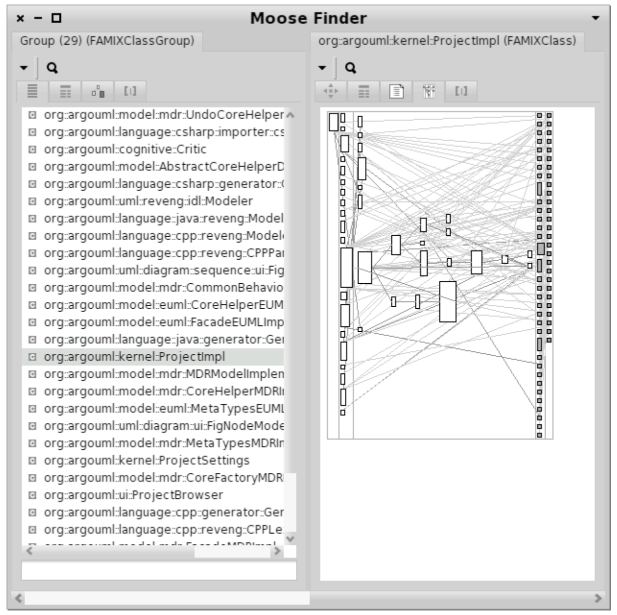
Rotten green test detectors

Regression testing, selection,…

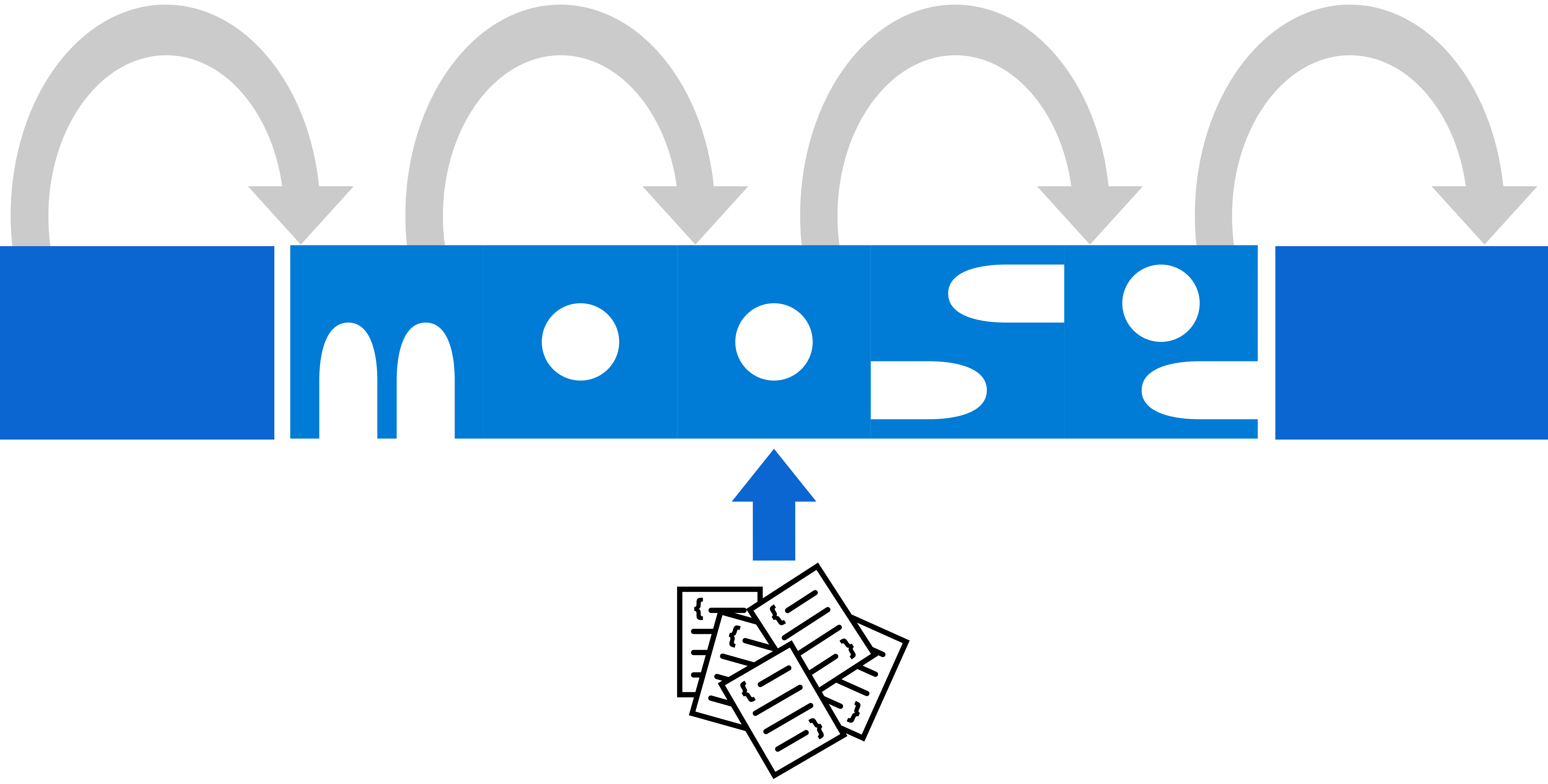
classes select: #isGod

McCabe = 21

LOC = 753,000


Moose Finder
Group (29) (FAMIXClassGroup)
org:argouml:kernel:ProjectImpl (FAMIXClass)

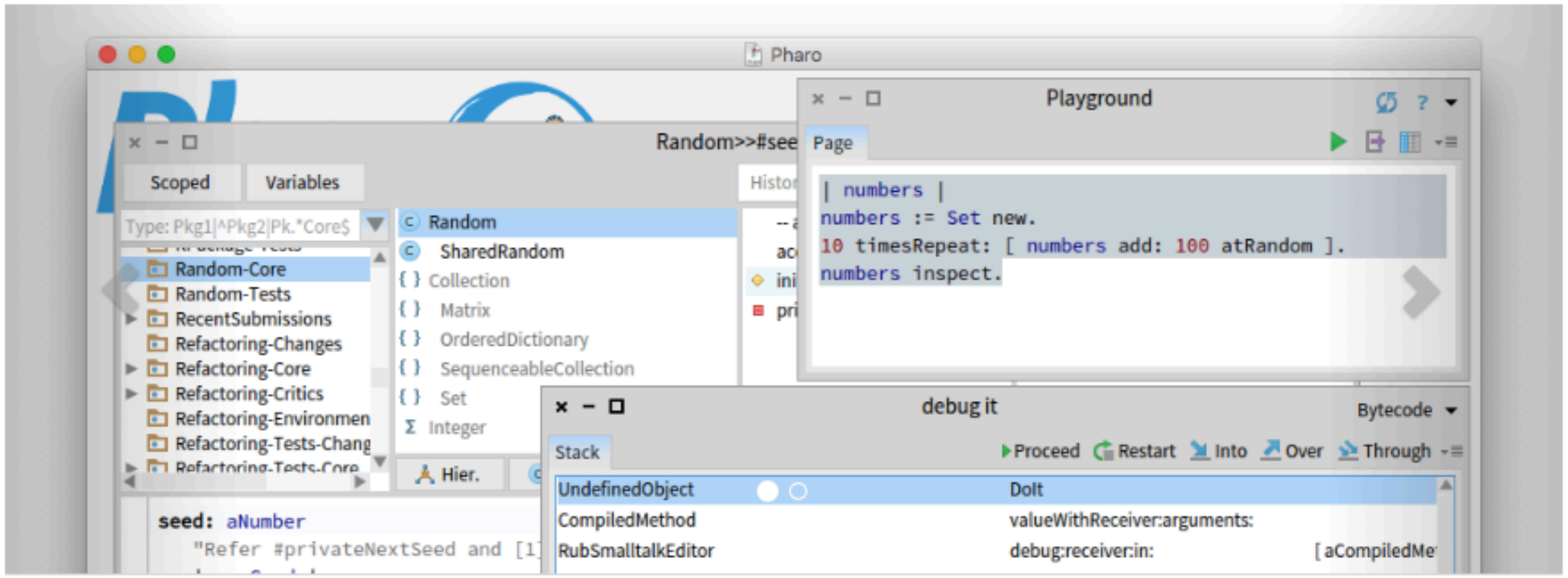
org:argouml:model:mdr:UndoCoreHelper
org:argouml:language:csharp:importer:cs
org:argouml:cognitive:Critic
org:argouml:model:AbstractCoreHelperD
org:argouml:language:csharp:generator:
org:argouml:uml:reveng:idl:Modeler
org:argouml:language:java:reveng:Model
org:argouml:language:cpp:reveng:Model
org:argouml:language:cpp:reveng:CPPPa
org:argouml:uml:diagram:sequence:ui:Fig
org:argouml:model:mdr:CommonBehavio
org:argouml:model:euml:CoreHelperEUM
org:argouml:model:euml:FacadeEUMLImp
org:argouml:language:java:generator:Ge
org:argouml:kernel:ProjectImpl
org:argouml:model:mdr:MDRModelImplen
org:argouml:model:mdr:CoreHelperMDRI
org:argouml:model:euml:MetaTypesEUML
org:argouml:uml:diagram:ui:FigNodeMode
org:argouml:model:mdr:MetaTypesMDRI
org:argouml:kernel:ProjectSettings
org:argouml:model:mdr:CoreFactoryMDR
org:argouml:ui:ProjectBrowser
org:argouml:language:cpp:generator:Ge
org:argouml:language:cpp:reveng:CPPLe



Pharo

The immersive programming experience

Pharo is a pure object-oriented programming language *and* a powerful environment, focused on simplicity and immediate feedback (think IDE and OS rolled into one).



```
| numbers |
numbers := Set new.
10 timesRepeat: [ numbers add: 100 atRandom ].
numbers inspect.
```

```
seed: aNumber
	"Refer #privateNextSeed and [1]
```

```
UndefinedObject          DoIt
CompiledMethod           valueWithReceiver:arguments:
RubSmalltalkEditor       debug:receiver:in:          [ aCompiledMe
```

Discover

Learn more about Pharo's key features and elegant design.

Download

Download latest version (8.0)!
Read more about here

Learn

Access the Pharo Mooc!
3000 people registered and follow the Pharo Mooc. You can find it here.
Watch the teaser!

Subscribe to the Pharo Newsletter

email address | Subscribe

Follow us on Twitter: @pharoproject

Pharo 90
~740 packages
- 9 000 classes
- 120 000 methods

250 forks sur Github
up to 100 contributors
30 regulars
- 8 sub projets
  - graphics
  - vcs
  - tools

Consortium
~ 28 companies
~ 25 academic

Virtual Machines

Modern Language Implementations

Managed **Execution**

Runtime Binary Translation

Managed **Memory**

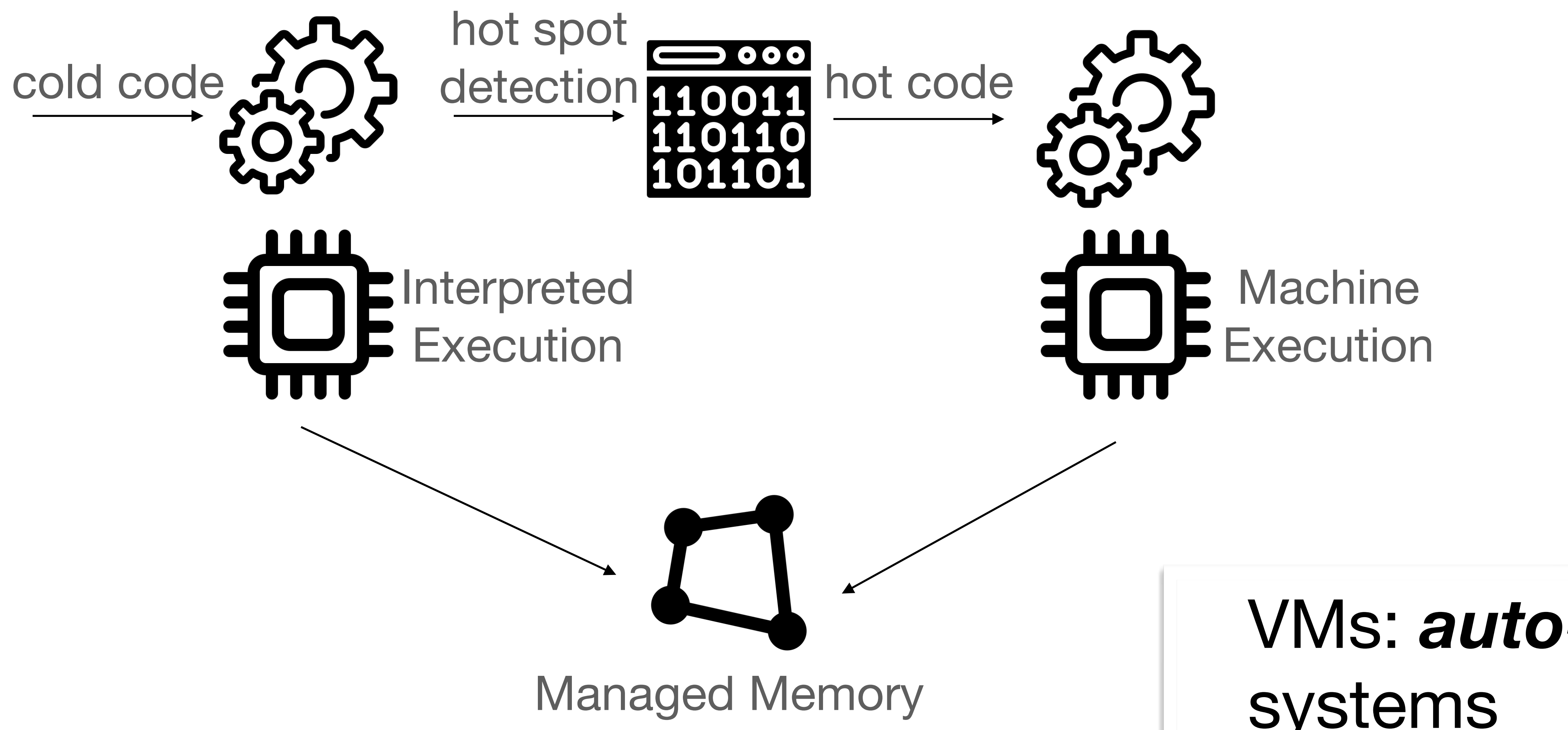
Hardware/System Interaction

ORACLE

Microsoft

Virtual Machines

Typical Architecture Overview



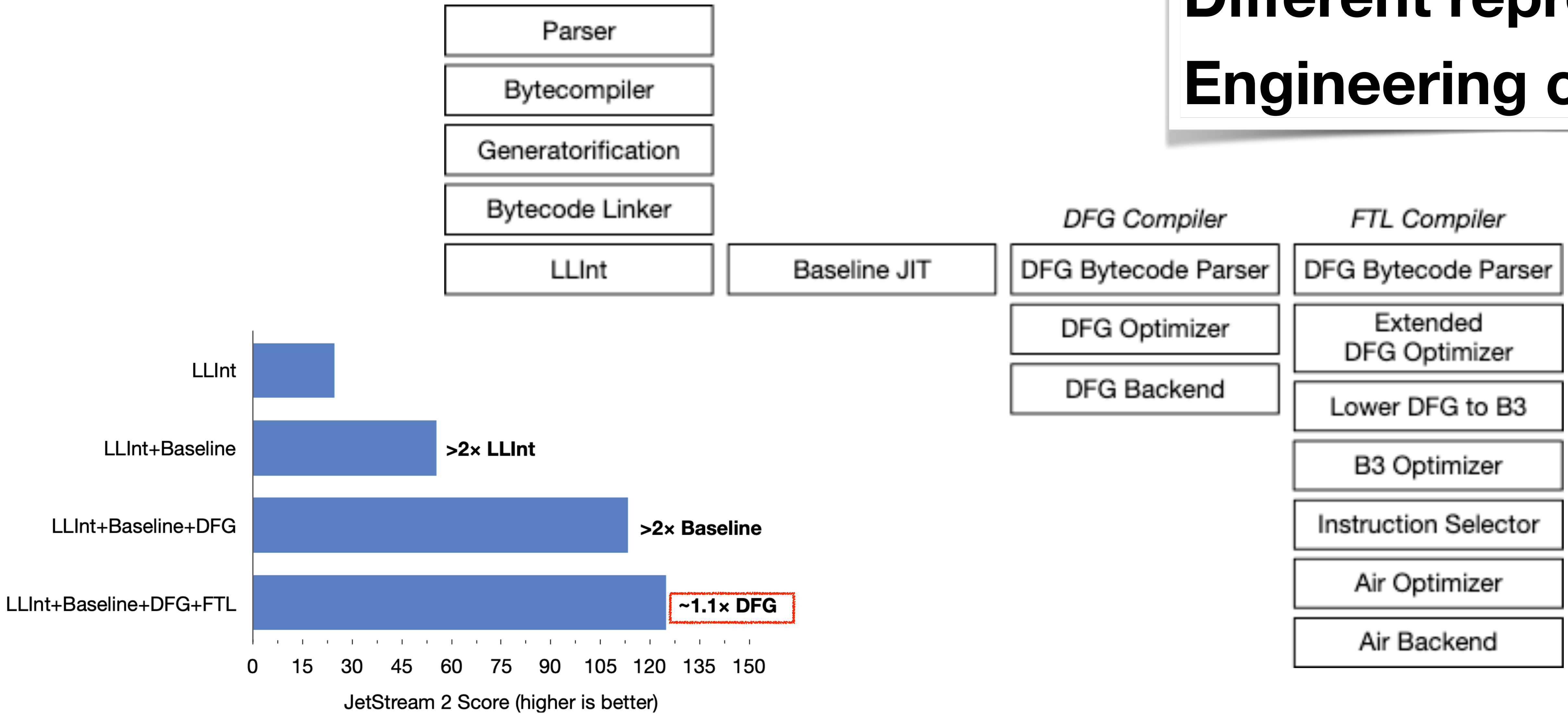
VMs: ***auto-adapt***
systems

Complexity and Cost of VMs

Multiple levels

Different represe

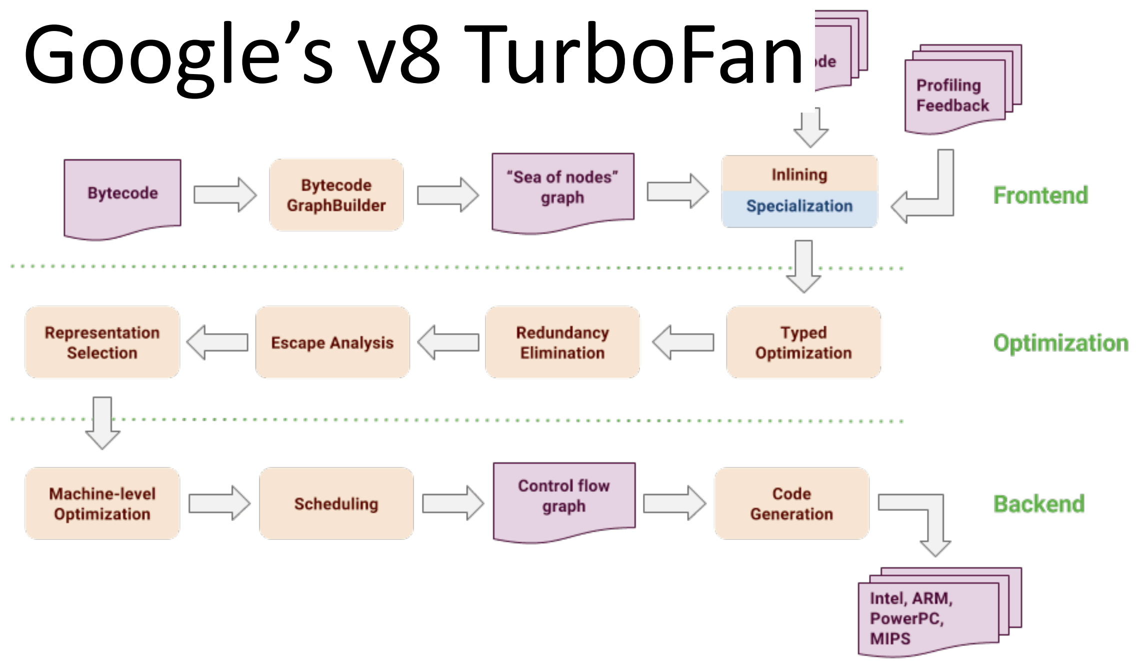
Engineering cos

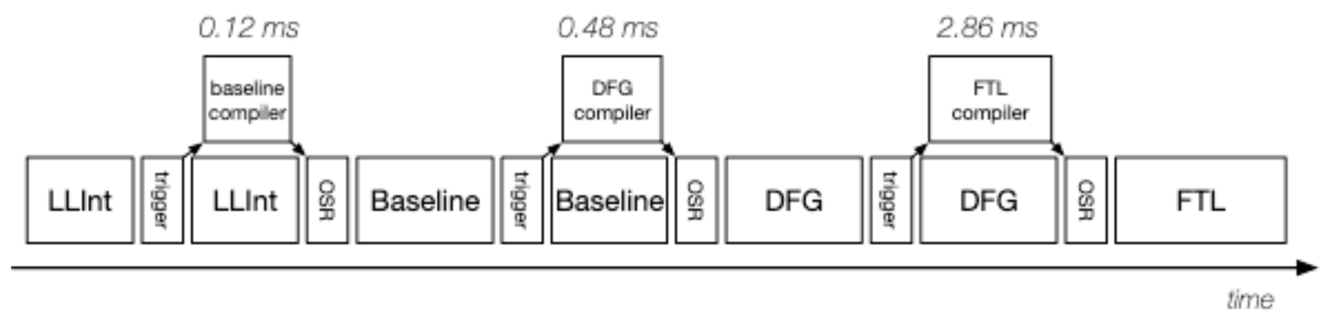


https://webkit.org/blog/10308/speculation-in-javascriptcore

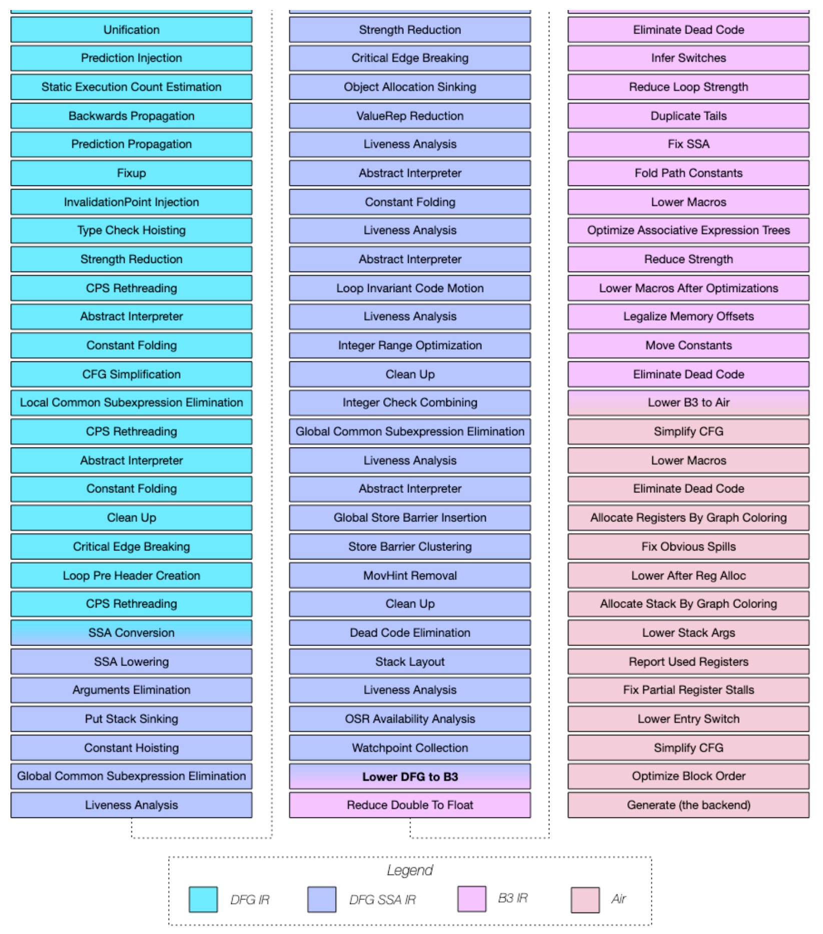
Complexity and Cost of VMs (II)

Google's v8 TurboFan





Apple's Safari JavascriptCore[2021]



https://webkit.org/blog/10308/speculation-in-javascriptcore/
https://ponyfoo.com/articles/an-introduction-to-speculative-optimization-in-v8

Managed Execution
Remarkable Challenges

- What are ***optimal*** organisations of multi-tier engines?
    - Combining interpreters with ***many levels*** of optimising compilers
- What is a ***better/minimal runtime*** support for developer ***tooling***?
    - Better debugging support
    - Runtime (speed, energy…) profiling
    - Benchmark automatic generation

Runtime Binary Translation
Remarkable Challenges

VMs are ***auto-adaptive*** systems

- How can runtime-compilers ***better speculate*** on application behaviour?
    - Speculate **on** more than types
    - Speculate **for** more than speed
- How can we improve the efficiency of ***cold code***?
    - Better interpreter optimisations
    - Low overhead binary translators

Managed Memory

Remarkable Challenges

- How can ***managed memory adapt*** to memory consumption patterns?
  - Scalability to ***multi-TB*** heaps
  - Automatically memory re-organisation
  - Reduce pauses
  - Support for modern hardware (e.g., non-volatile memories)

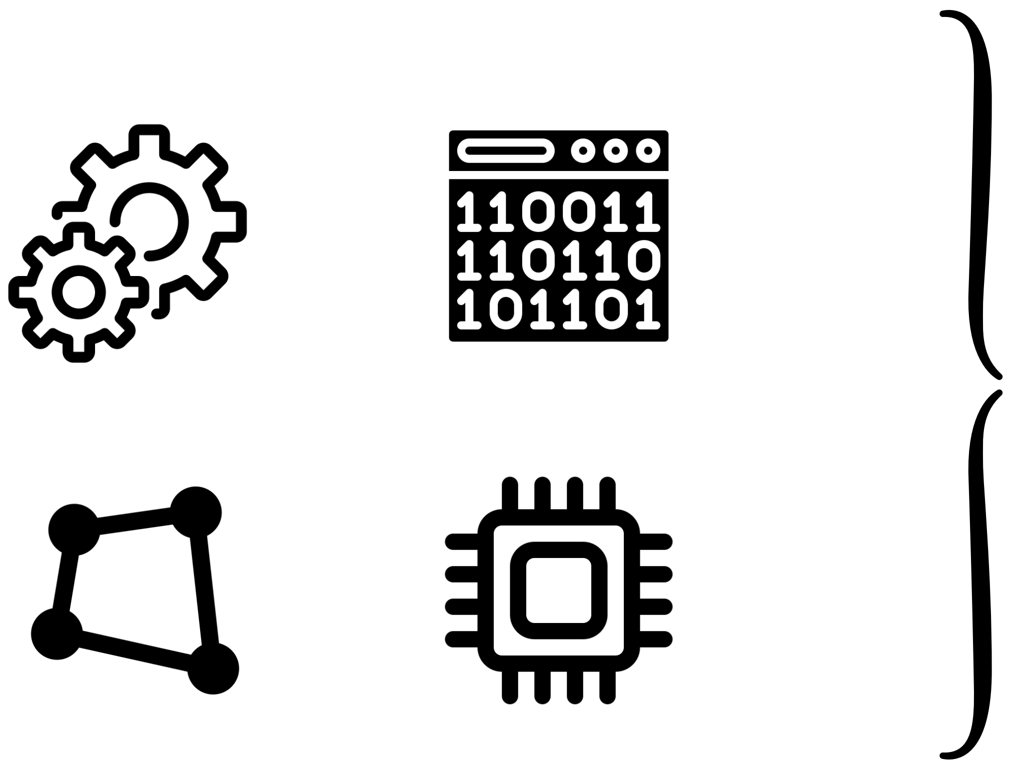
Hardware/System Interaction

Remarkable Challenges

- How can modern VMs exploit ***hardware-software co-design***?
- Automatic deport computation to dedicated hardware
  - GPU
  - FPGA
  - Extensible ISAs (e.g., RISC-V)

Cross-Cutting Challenges

(And Contradictory Challenges!)



- Energy Consumption
- Execution Speed
- Security
- Correctness
- Modularity

Cross-Cutting Challenges

Selected Challenges

- **Security threats** of multi-tier execution engines
- Speculative runtime compilation for **frugal systems**
- **Profile-guided** detection of application parallelisation opportunities
- **Securing** VMs through **dedicated** hardware
- Minimising **energy impact** of garbage collection algorithms

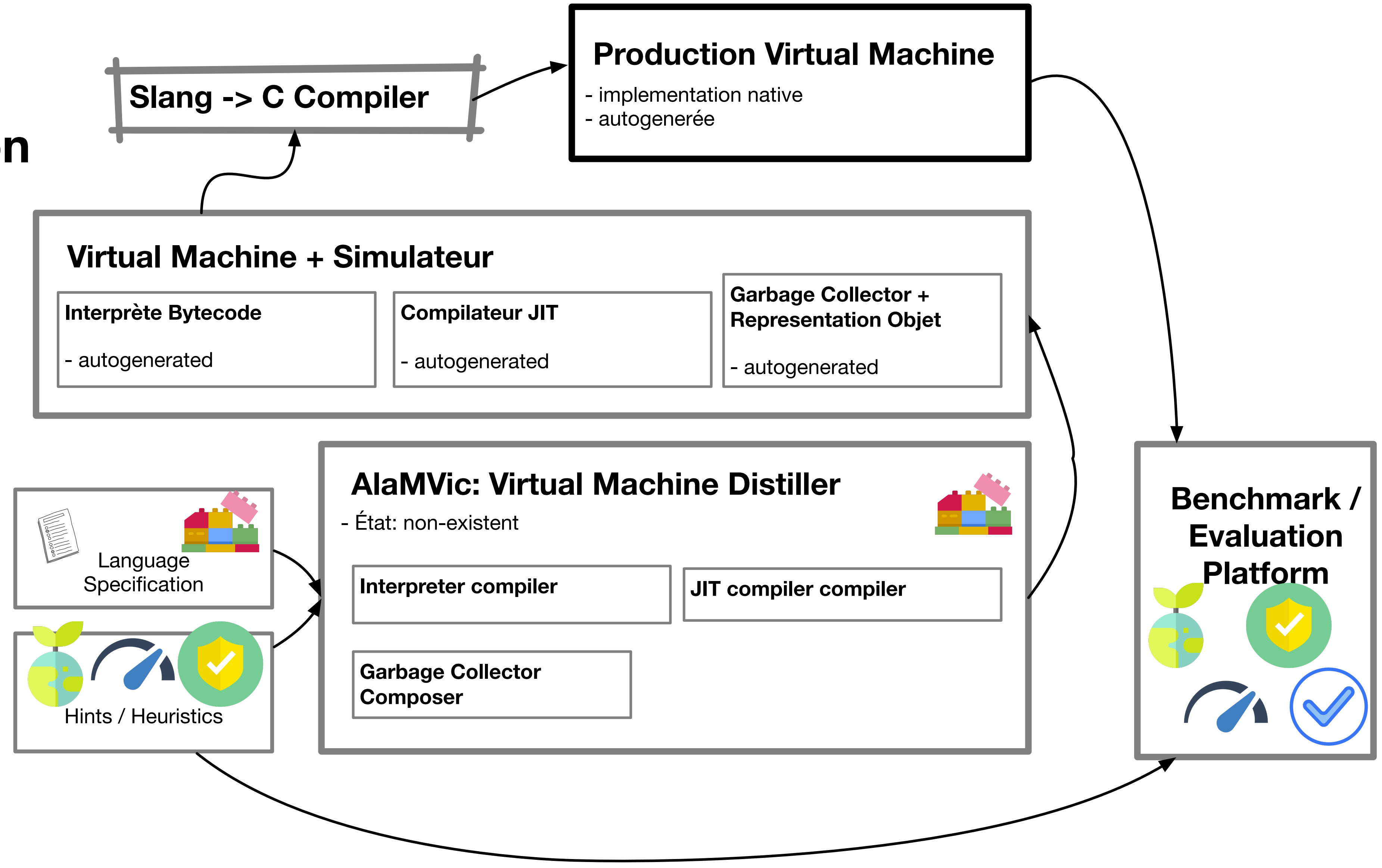
Selected Software Engineering Challenges

- **Automatic** detection of **performance** regressions
- **Automatic validation** of multi-tier execution engines
- Minimising the **construction cost** of efficient JIT compilers

RMod

AlaMVic: a generative approach

- Compiler generation
- Exchangeable components
- Optimization heuristics
- Open exploratory platform



Early RMOD achievements

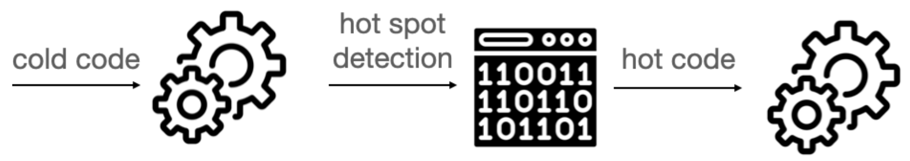
Dev side of things

- JIT for Apple M1, Windows, Raspberry ARM 64bits in production
- Helping ENSTA Bretagne to develop a Risc-V JIT
- Streamlining transpilation/compilation chain
- Taking advantage of VM tests [MPLR paper]
- Some productivity enhancer tools (Unicorn simulator, assembly browser, interactive CFG navigation,…)

Early RMOD achievements

Research side

- RQ: ***static*** code fall through reorganisation is it worth ? (alternative to Pettis-Hansen BB reordering)



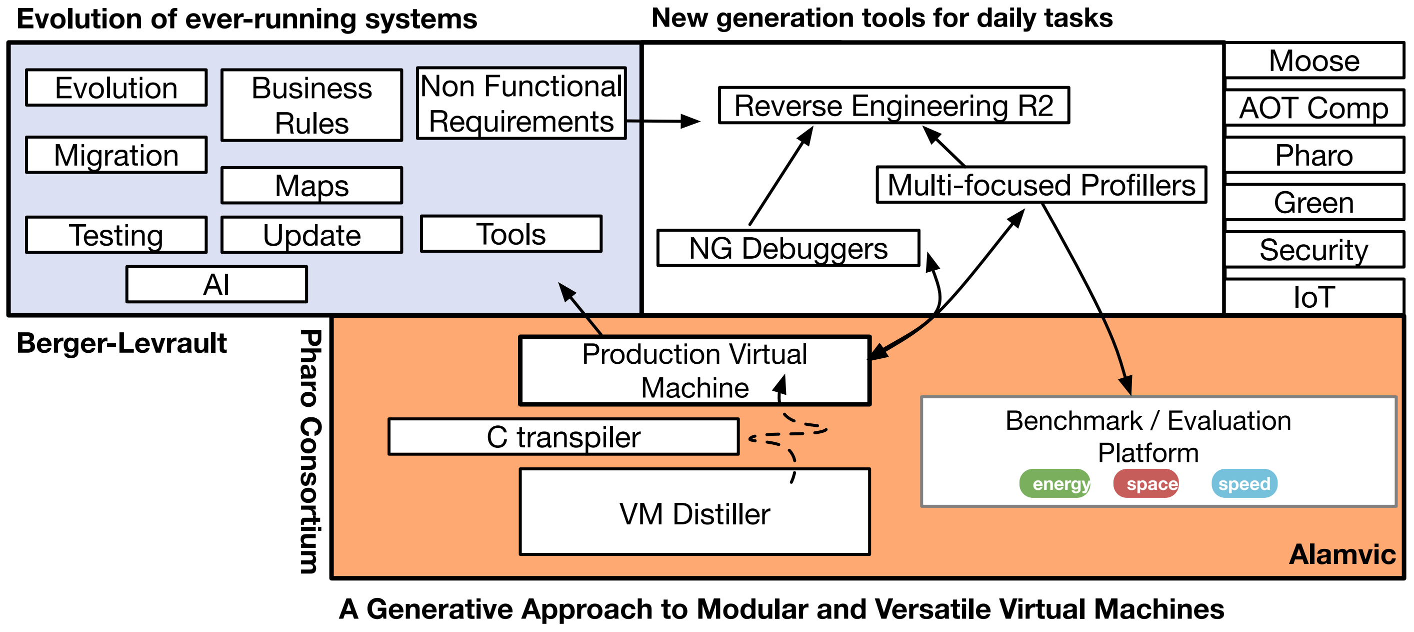
- Reducing the load of manual code (~100 bytecodes, ~300 primitives)
  - RQ1: Are interpreted and compiled code equivalent? Concolic + differential testing
  - RQ2: Can we remove manual compiled code? Abstract interpreter for compiled code generation (underway)

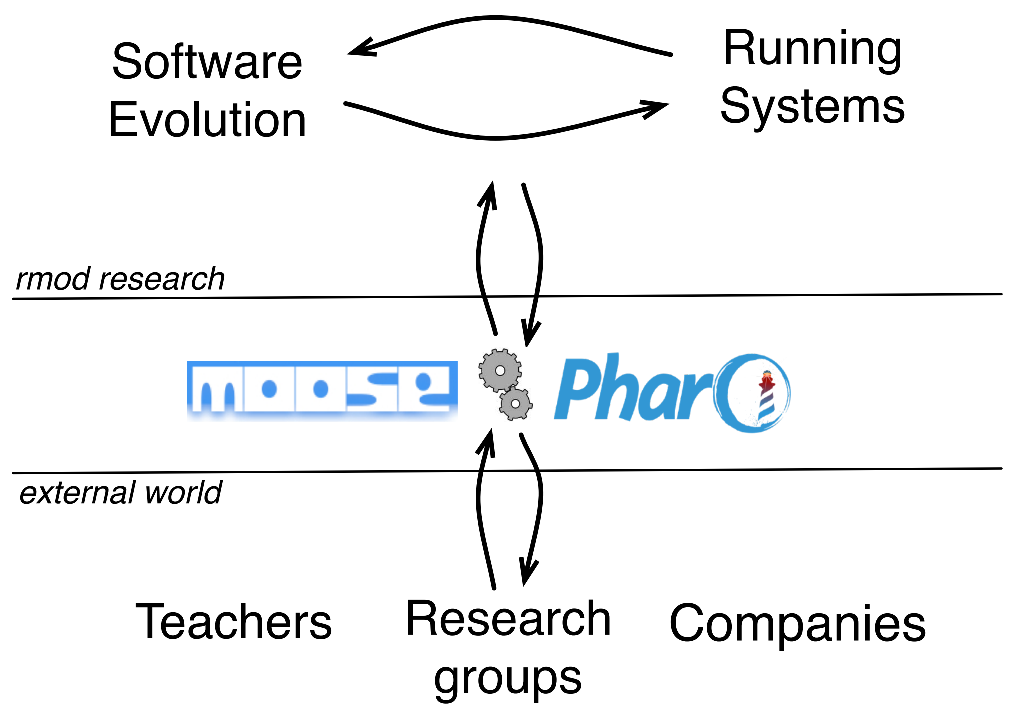
Internships

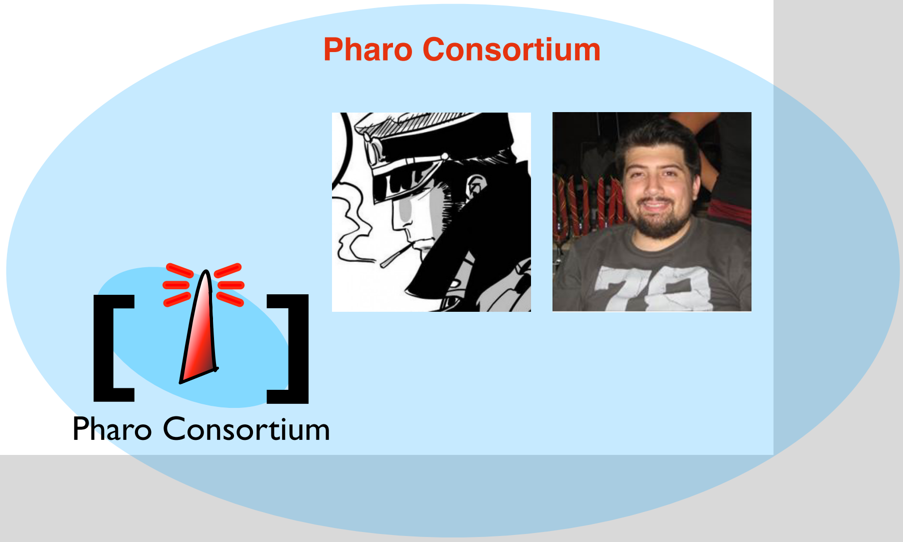
- https://alamvic.github.io/positions.html
- https://rmod.gitlabpages.inria.fr/website//jobs.html#jobs

RMod

Berger Levrault BL







Inria