



High-performance language virtual machines: an analysis and challenges

Stéphane Ducasse, Guillermo Polito, Pablo Tesone, Gaël Thomas, Loïc Lagadec

► To cite this version:

Stéphane Ducasse, Guillermo Polito, Pablo Tesone, Gaël Thomas, Loïc Lagadec. High-performance language virtual machines: an analysis and challenges. 2022. hal-03770053

HAL Id: hal-03770053

<https://inria.hal.science/hal-03770053>

Submitted on 6 Sep 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

High-Performance Language Virtual Machines: an analysis and challenges

S. Ducasse G. Polito [RMOD/Evref - Inria] P. Tesone [Pharo consortium]
G. Thomas [Telecom SudParis]
L. Lagadec [ENSTA]

March 2022



ENSTA
BRETAGNE



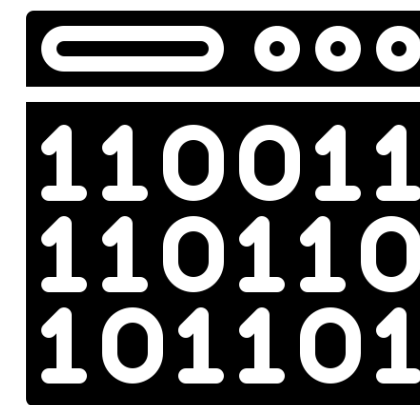
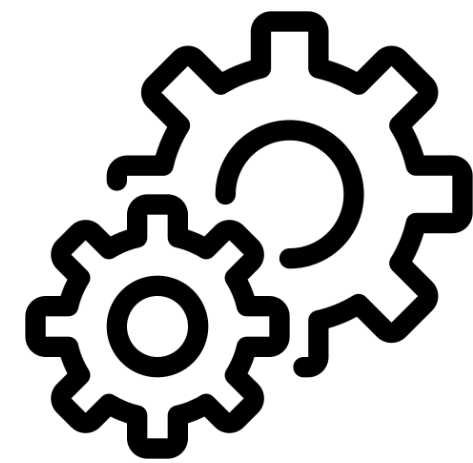
Language VMs are Ubiquitous

- They are **everywhere**: browsers, mobile phones, drones, robots...
 - Banks, servers, aircrafts
- Portability, self-optimisation and adaptation, high-level services (GC)
 - Java, Javascript, Pharo, PHP, Python, Ruby, C#...
 - Derivates: Typescript, Scala, web assembly

Language Virtual Machines

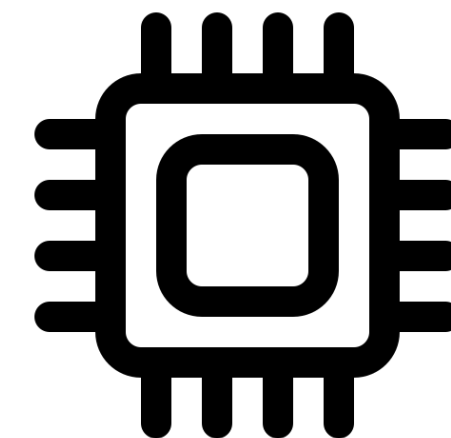
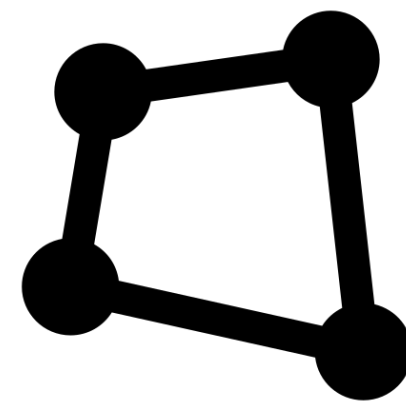
Modern Language Implementations

Managed **Execution**



Runtime Binary Translation

Managed **Memory**



Hardware/System Interaction

Key Players

- Javascript: Safari (Apple), V8 (Google), SpiderMonkey (Mozilla)
- Java: Truffle, GraalVM (Oracle)
- .NET, C#, VB: (Microsoft)



ORACLE



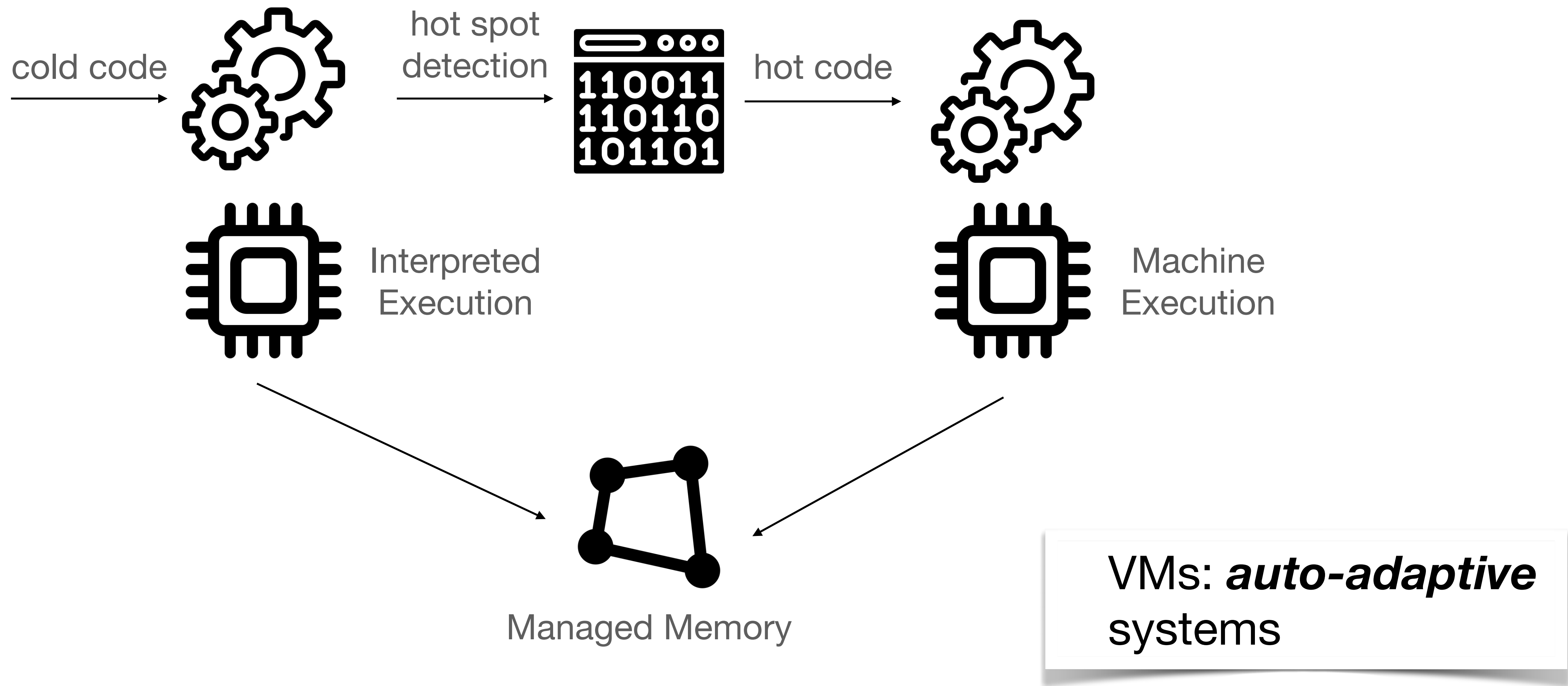
VMs as Competitive Advantage

Large companies developed their OWN

- Hack: Facebook's PHP 
- Ruby: Shopify 
- GemTalk Systems 
- Netflix 
- Many python, ruby are popping up

Virtual Machines

Typical Architecture Overview



Compiler Pipeline Example

source code - to - bytecode interpreter.

Example: arithmetics

`a + b`

push a
push b
send +

```
send_+(op1, op2){  
    if (isInteger(op1) && isInteger(op2)) {  
        r = op1 + op2;  
        if (!overflow){  
            return push(r);  
        }  
    }  
    send_message(+)  
}
```

source code

bytecode

interpreter code

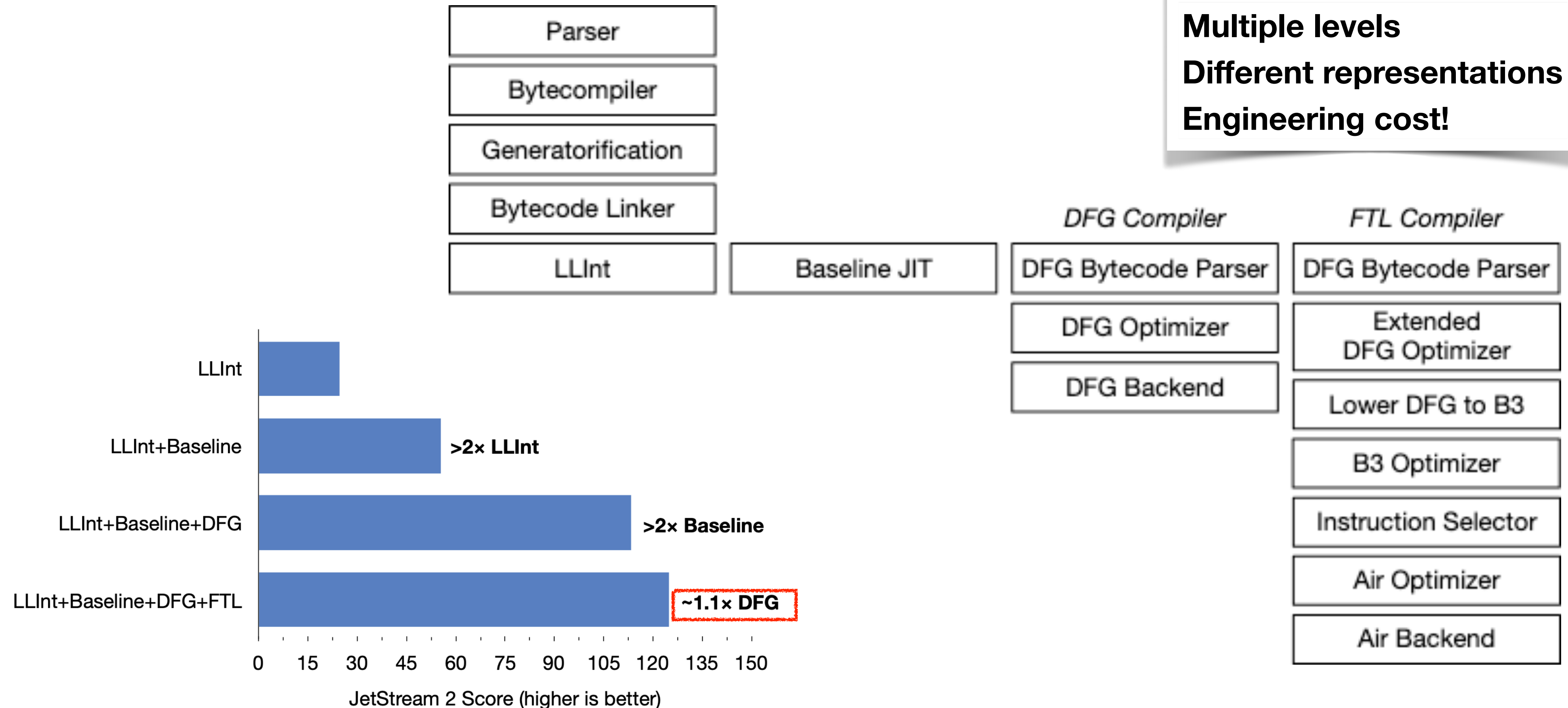
Interpreter and Compiler Semantics

```
1  Interpreter >> bytecodePrimAdd
2  | rcvr arg result |
3  rcvr := self internalStackValue: 1.
4  arg := self internalStackValue: 0.
5  (objectMemory areIntegers: rcvr and: arg) ifTrue: [
6    result := (objectMemory integerValueOf: rcvr) + (
7      objectMemory integerValueOf: arg).
8    "Check for overflow"
9    (objectMemory isIntegerValue: result) ifTrue: [
10     self
11       internalPop: 2
12       thenPush: (objectMemory integerObjectOf: result).
13     ^ self fetchNextBytecode "success"]].
14 "Slow path, message send"
self normalSend
```

```
1  ... # previous bytecode IR
2    checkSmallInteger t0
3    jumpzero notsmi
4    checkSmallInteger t1
5    jumpzero notsmi
6    t2 := t0 + t1
7    jumpIfNotOverflow continue
8  notsmi: #slow case first send
9    t2 := send #+ t0 t1
10 continue:
11  ... # following bytecode IR
```

A concrete example: Javascript core

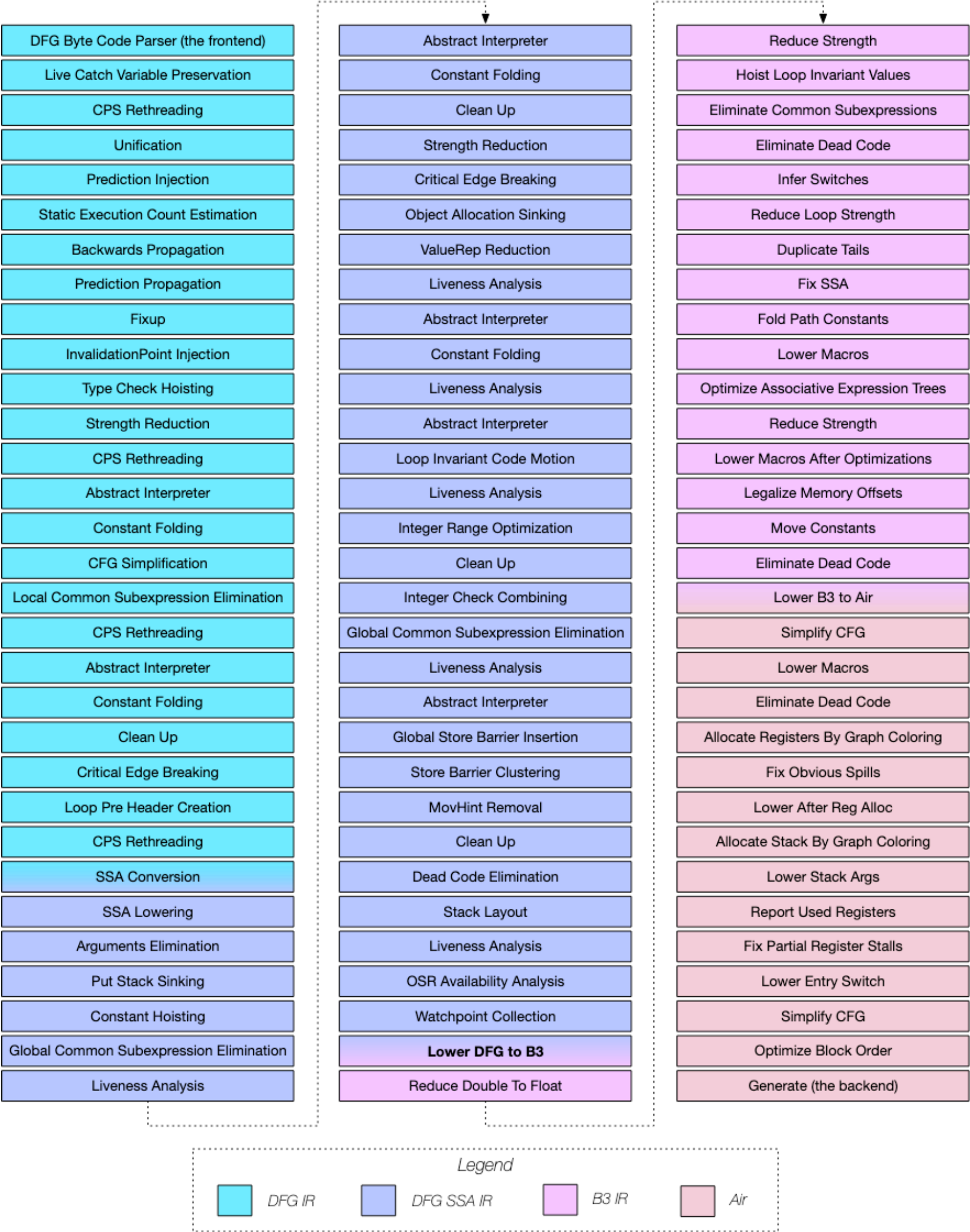
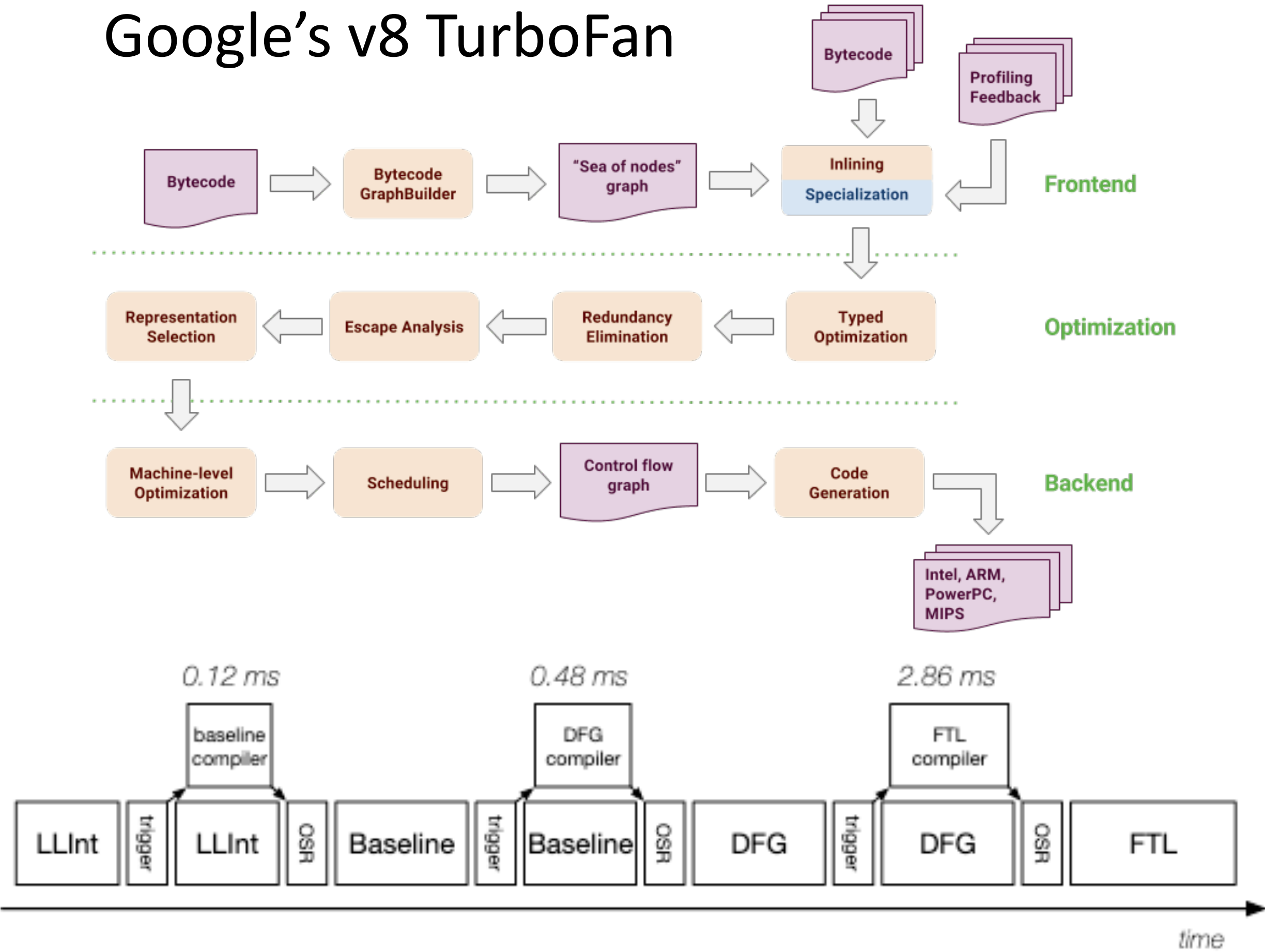
Multiple levels
Different representations
Engineering cost!



Quid Complexity and Cost of VMs?

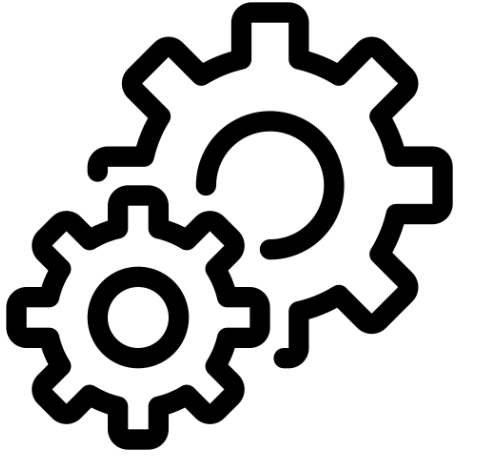
Apple's Safari JavascriptCore[2021]

Google's v8 TurboFan



Managed Execution

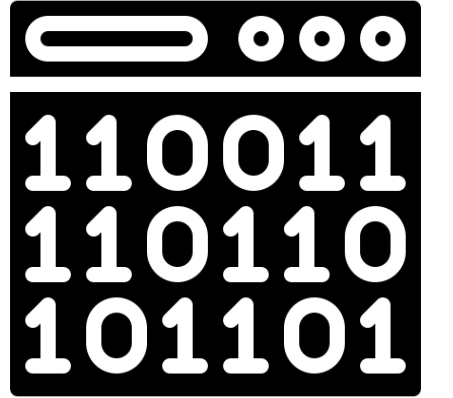
Remarkable Challenges



- Challenge 1: What are ***optimal*** organisations of multi-tier engines?
 - Combining interpreters with ***many levels*** of optimising compilers
- Challenge 2: 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

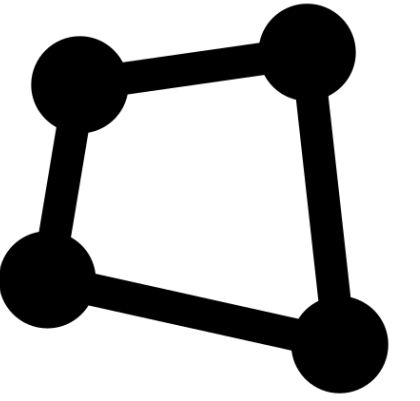


VMs are *auto-adaptive* systems

- Challenge 3: How can runtime-compilers *better speculate* on application behaviour?
 - Speculate **on** more than types
 - Speculate **for** more than speed
- Challenge 4: How can we improve the efficiency of *cold code*?
 - Better interpreter optimisations
 - Low overhead binary translators

Managed Memory

Remarkable Challenges

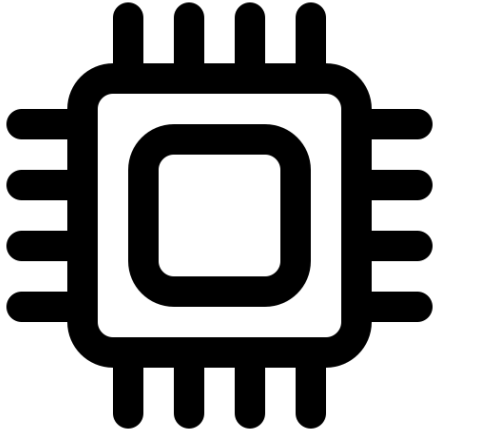


Challenge 5: How can *managed memory adapt* to memory consumption patterns?

- Scalability to *multi-TB* heaps
- Automatically memory re-organisation
- Reduce pauses
- Support for modern hardware (e.g., disaggregated memory, non-volatile memories)
- encrypted memory (arm trustzone/intel sgx), compressed memory
- OS and System VM Interactions

Hardware/System Interaction

Remarkable Challenges

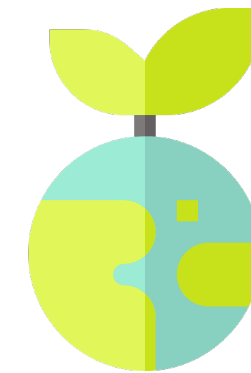
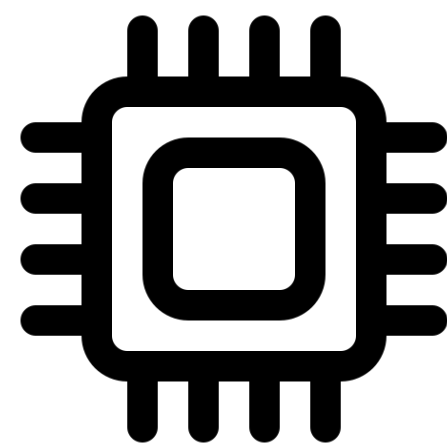
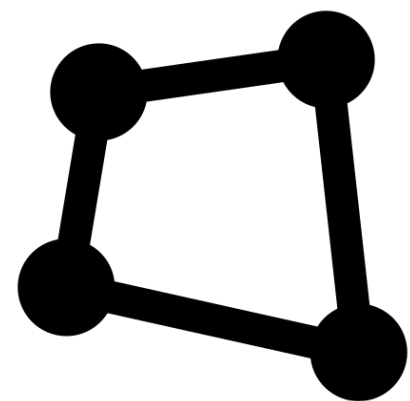
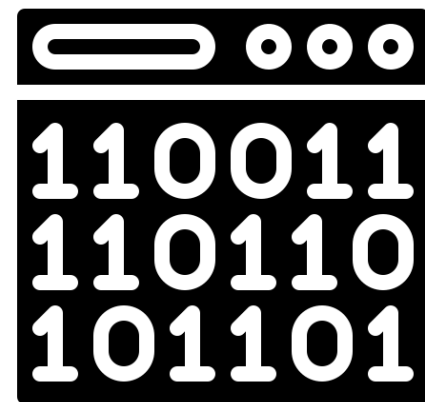
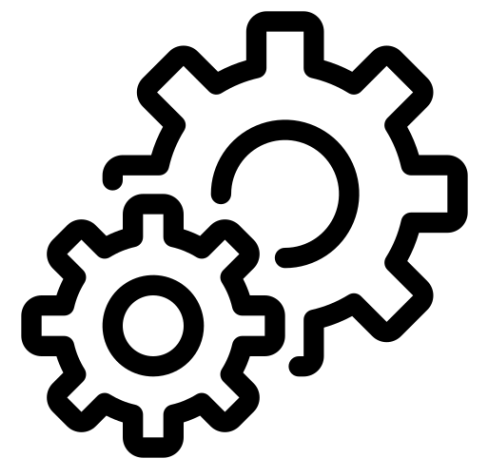


Challenge 6: How can modern VMs exploit ***hardware-software co-design?***

- Automatic deport computation to dedicated hardware
 - GPU
 - FPGA
 - Extensible ISAs (e.g., RISC-V)

Cross-Cutting Challenges

(And Contradictory Challenges!)



Energy Consumption



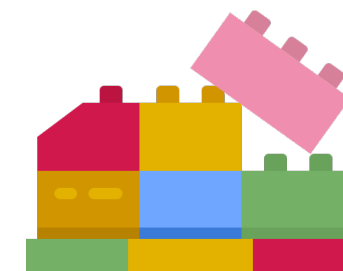
Execution Speed



Security





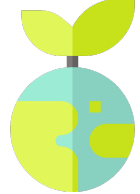

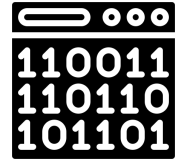

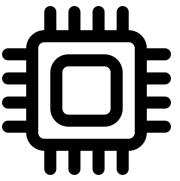

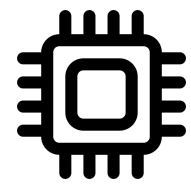
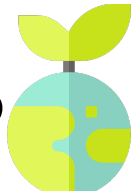
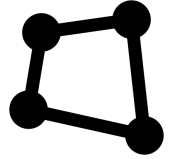
Correctness





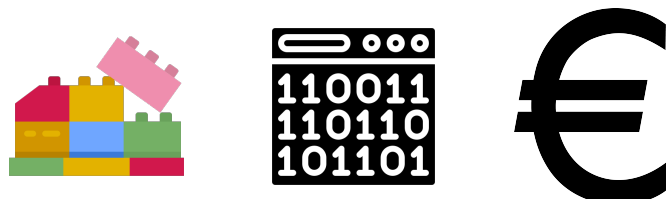
Modularity

Cross-Cutting Challenges

Selected Challenges

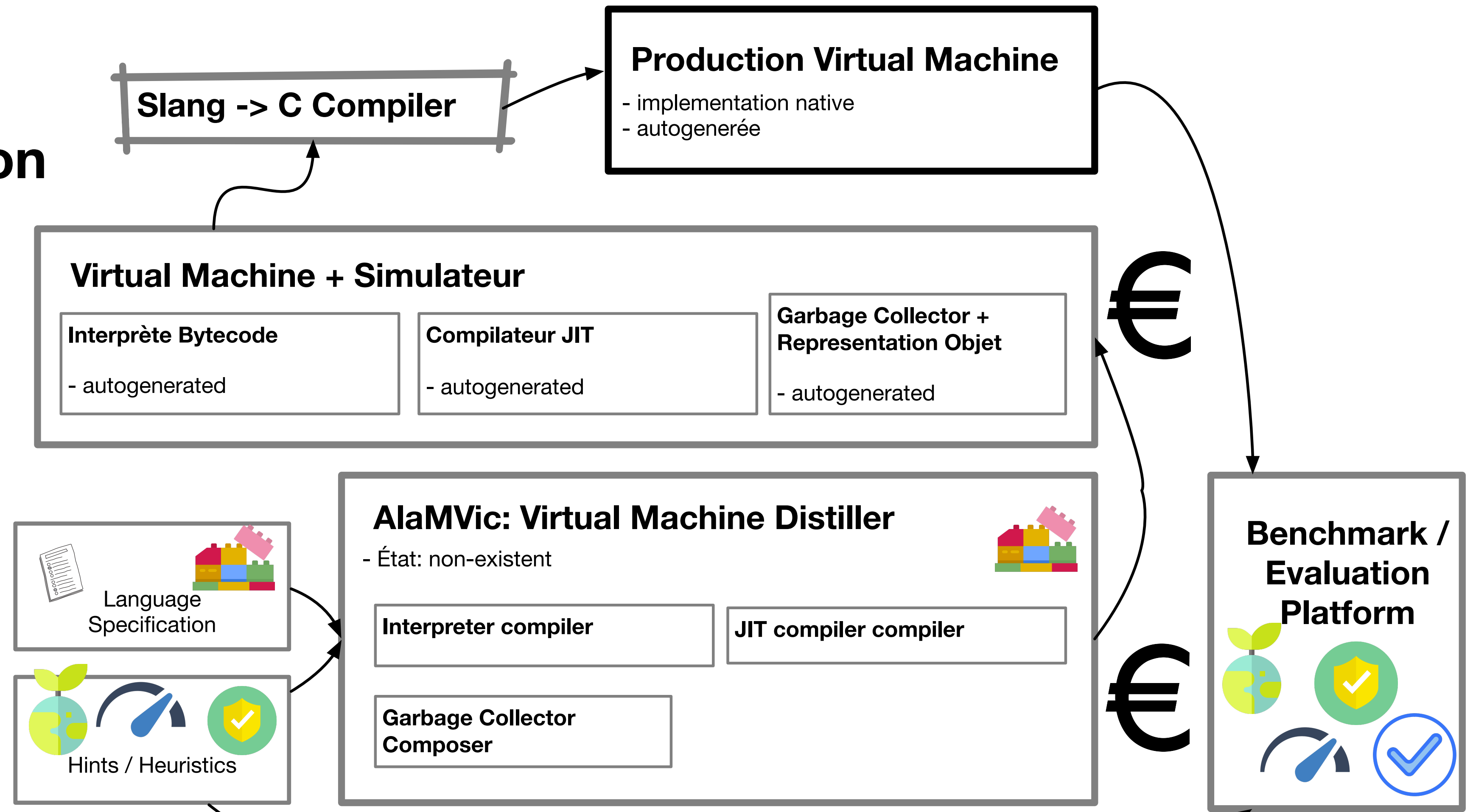
- **Security threats** of multi-tier execution engines  
- Speculative runtime compilation for **frugal systems**   
- **Profile-guided** detection of application parallelisation opportunities  
- **Securing** VMs through **dedicated** hardware  
- Minimising **energy impact** of garbage collection algorithms  

Selected Software Engineering Challenges €

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

AlaMVic: a generative approach

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



Early RMOD achievements

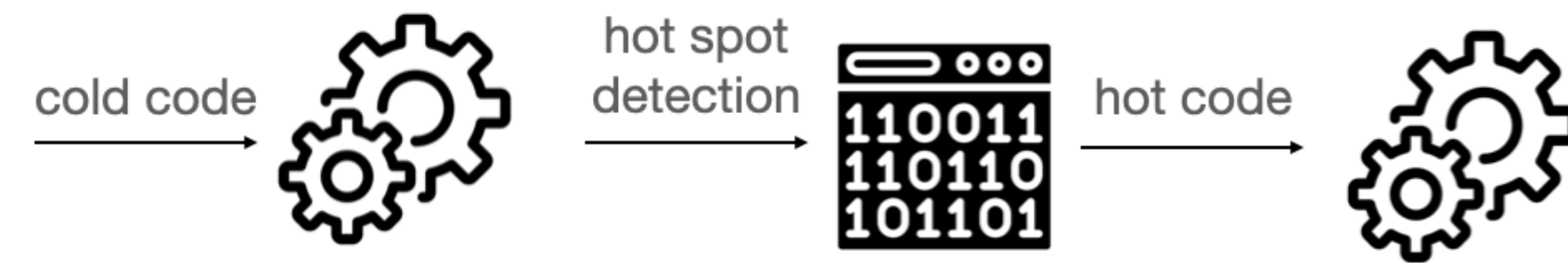
Dev side of things

- 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, MoreVM paper]
- Some productivity enhancer tools (Unicorn simulator, assembly browser, interactive CFG navigation,...)

Early RMOD achievements

Research side

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



- Reducing manual code (~100 bytecodes, ~300 primitives)
 - RQ: Are interpreted and compiled code equivalent? Concolic + differential testing
 - RQ: Can we generate JIT compilers? Abstract interpreter for compiled code generation (underway)

Benagil

Research side

- J-NVM: Efficient integration of a persistent memory in a Java Virtual Machine
- PrivaDSL: Use of Intel SGX in a Java virtual machine
- Study of a Java virtual machine for disaggregated memory
- A shell language and runtime for serverless applications

Early ENSTA achievements

Dev side of things

- RISC-V JIT for a production level VM - Pharo consortium
- RISC-V board

Research side

- Study language VM level attacks
- Starting to propose protections against language VM-level attacks

Language VMs are strategical assets

- Controlling the execution engine, controls the world
- French research should not miss the opportunity
- Independence from the will of big companies is crucial for research
- Rare french teams on the topic should be supported!

