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



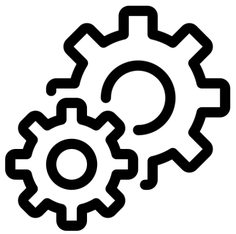
Language VMs are Ubiquitous

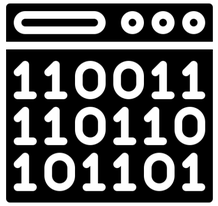
- They are **everywhere**: browsers, mobile phones, drones, robots…
  - Banks, servers, aircrafts
- Portability, self-optimisation and adaptation, high-level services (GC)
  - Java, Javascript, Pharo, PHP, Python, Ruby, C#…
  - Derivates: Typescript, Scala, web assembly

Language Virtual Machines

Modern Language Implementations

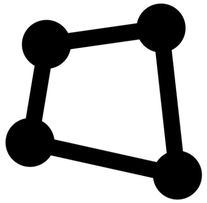
Managed **Execution**

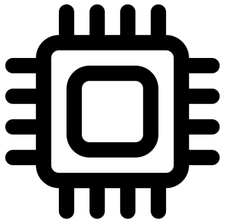




Runtime Binary Translation

Managed **Memory**





Hardware/System Interaction

Key Players

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

VMs as Competitive Advantage

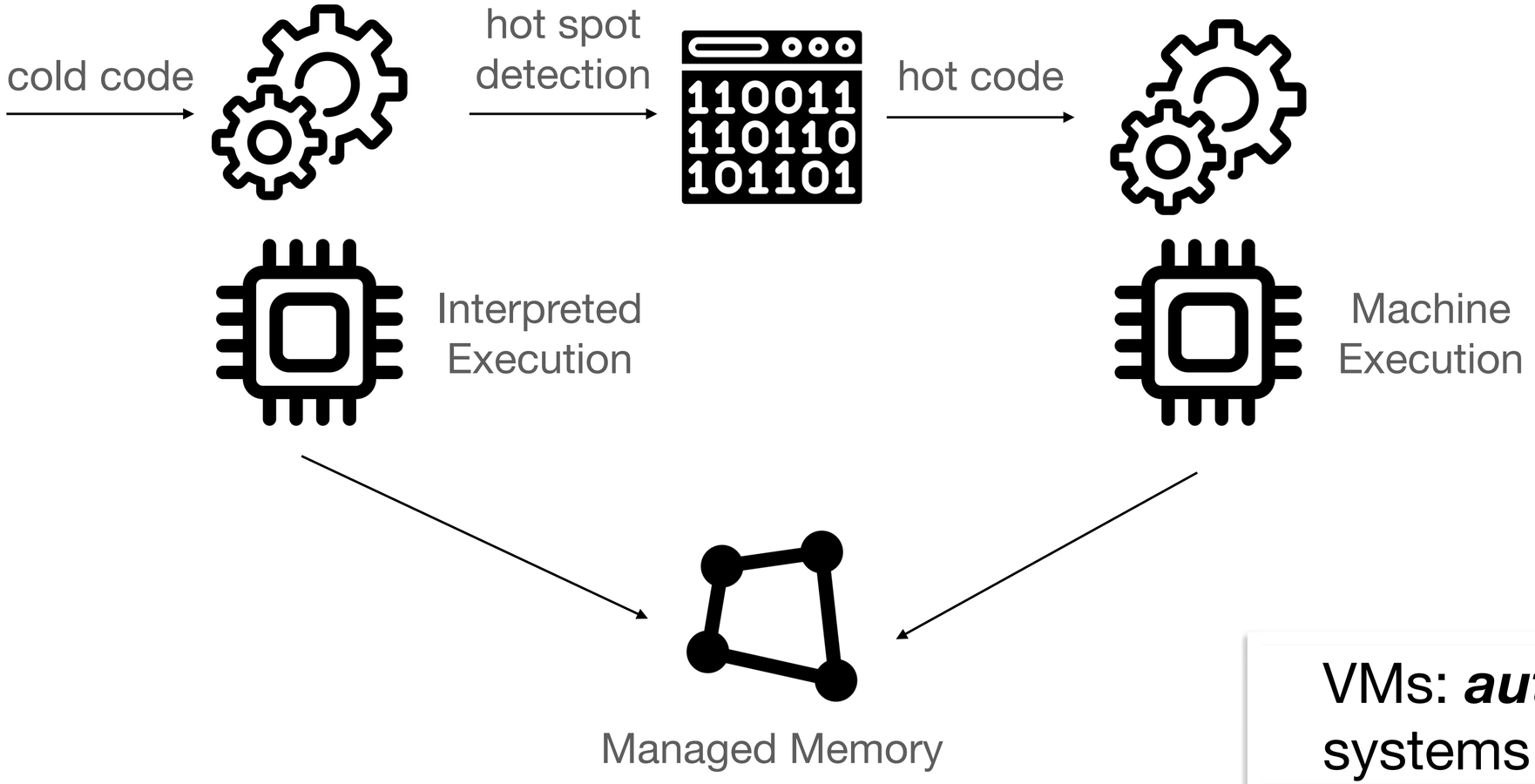
Large companies developed their OWN

- Hack: Facebook's PHP 
- Ruby: Shopify 
- GemTalk Systems 
- Netflix NETFLIX

- Many python, ruby are popping up

Virtual Machines

Typical Architecture Overview



VMs: ***auto-adaptive*** systems

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(+)
}
```

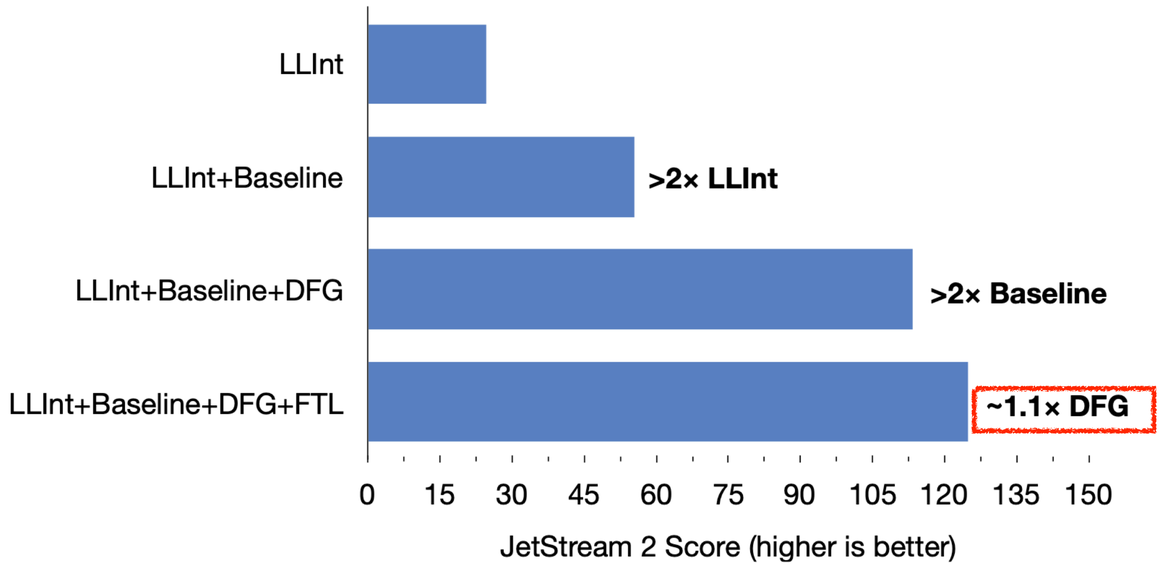
source code bytecode interpreter code

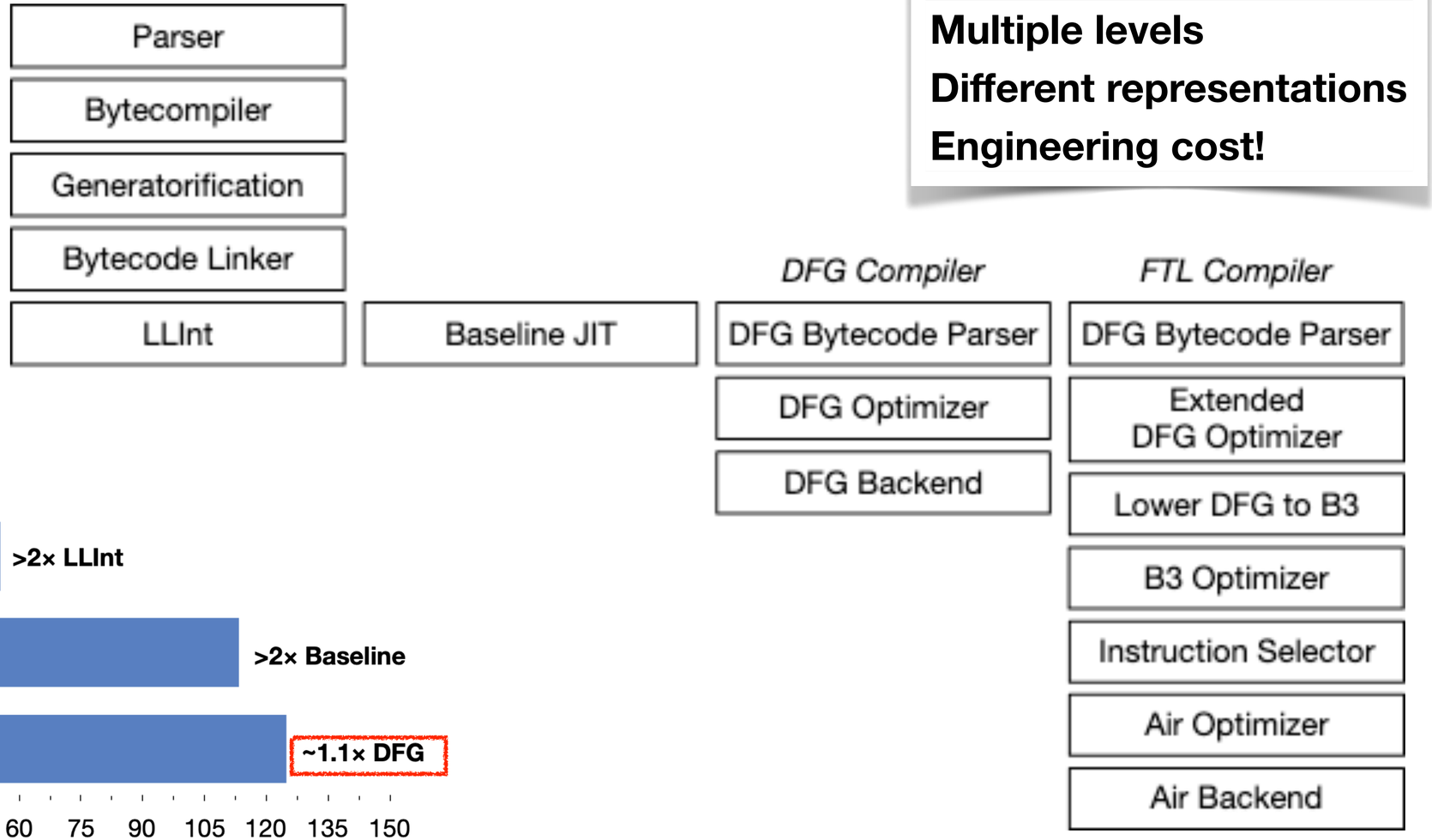
Interpreter and Compiler Semantics

```
Interpreter >> bytecodePrimAdd
    | rcvr arg result |
    rcvr := self internalStackValue: 1.
    arg := self internalStackValue: 0.
    (objectMemory areIntegers: rcvr and: arg) ifTrue: [
        result := (objectMemory integerValueOf: rcvr) + (
            objectMemory integerValueOf: arg).
        "Check for overflow"
        (objectMemory isIntegerValue: result) ifTrue: [
            self
                internalPop: 2
                thenPush: (objectMemory integerObjectOf: result).
            ^ self fetchNextBytecode "success"]].
    "Slow path, message send"
    self normalSend
```

```
    ... # previous bytecode IR
            checkSmallInteger t0
            jumpzero notsmi
            checkSmallInteger t1
            jumpzero notsmi
            t2 := t0 + t1
            jumpIfNotOverflow continue
notsmi: #slow case first send
            t2 := send #+ t0 t1
continue:
    ... # following bytecode IR
```

A concrete example: Javascript core



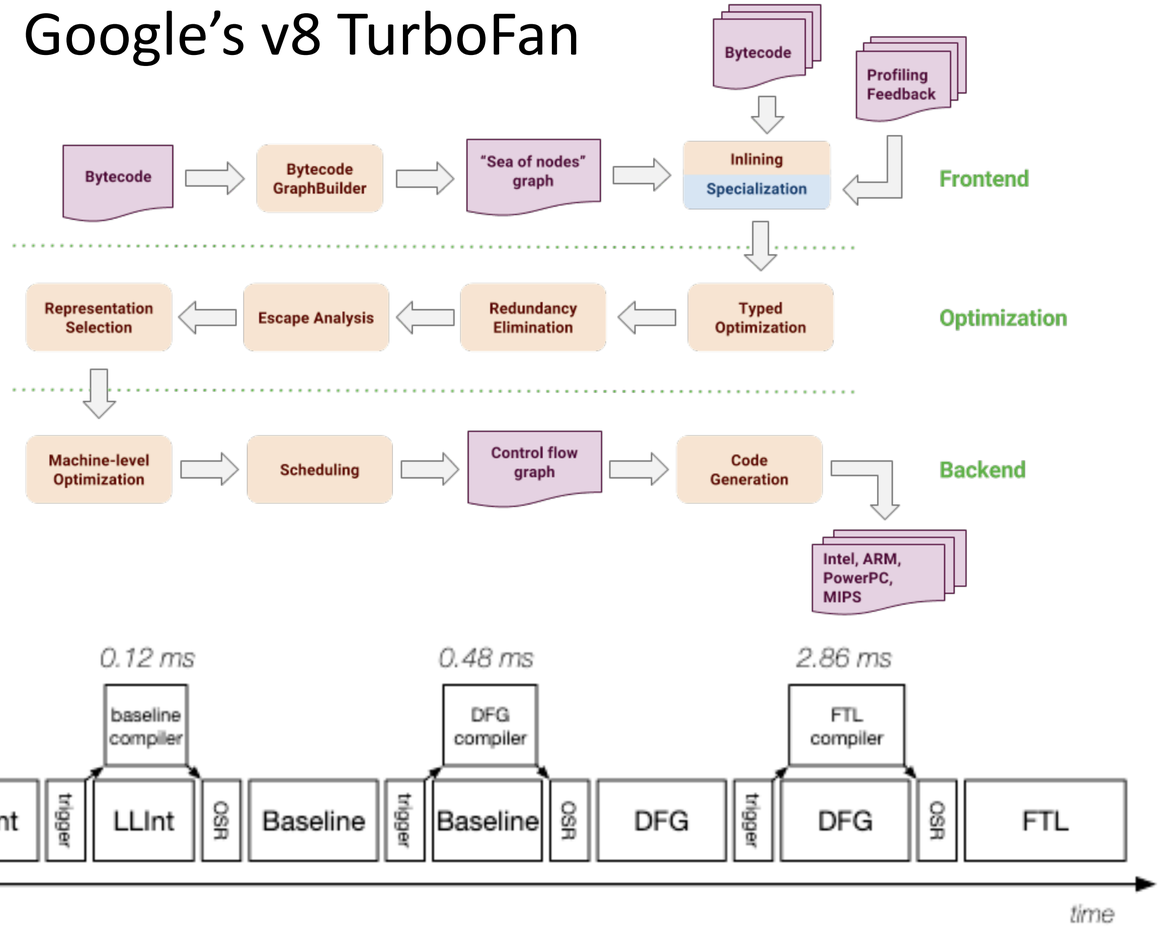


Multiple levels
Different representations
Engineering cost!

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

Quid Complexity and Cost of VMs?

Google's v8 TurboFan

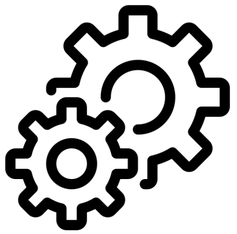


Apple's Safari JavascriptCore[2021]

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

Managed Execution

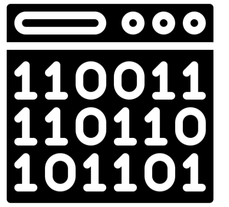
Remarkable Challenges



- 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

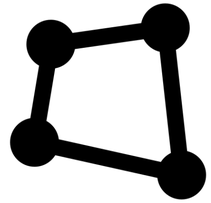


VMs are ***auto-adaptive*** systems

- 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

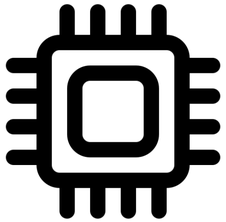


Challenge 5: How can ***managed memory adapt*** to memory consumption patterns?

- 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 Interations

Hardware/System Interaction

Remarkable Challenges

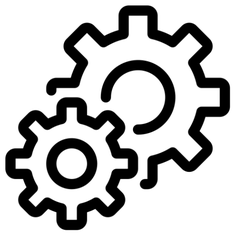


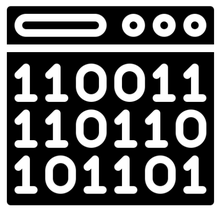
Challenge 6: How can modern VMs exploit ***hardware-software co-design***?

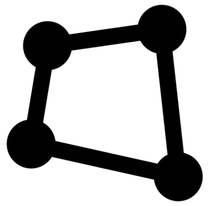
- Automatic deport computation to dedicated hardware
  - GPU
  - FPGA
  - Extensible ISAs (e.g., RISC-V)

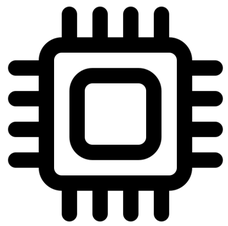
Cross-Cutting Challenges

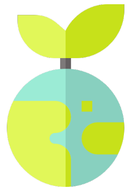
(And Contradictory Challenges!)









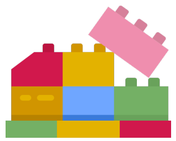


Energy Consumption

Execution Speed

Security

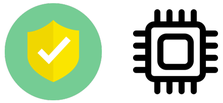
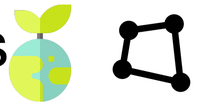
Correctness



Modularity

Cross-Cutting Challenges

Selected Challenges

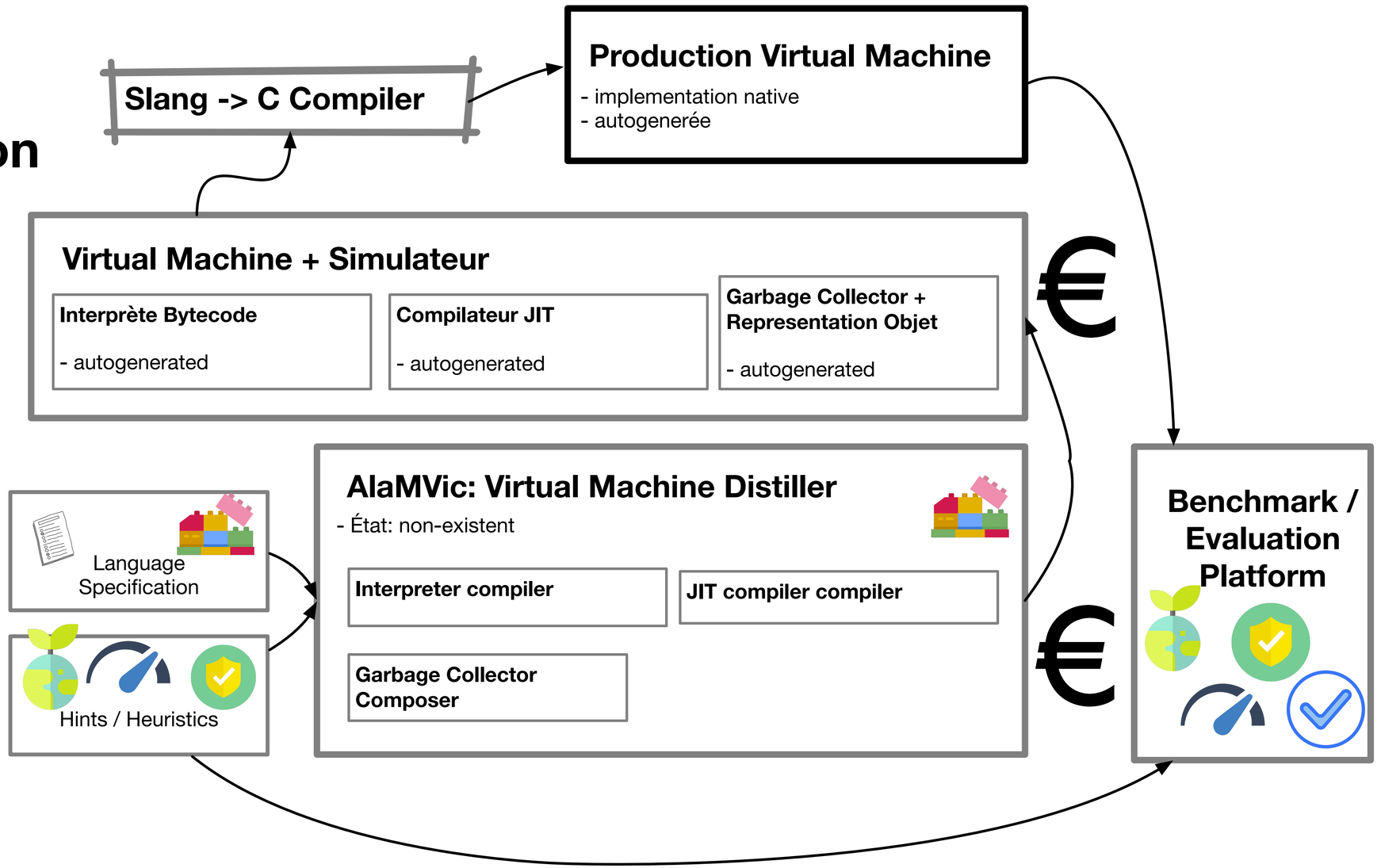
- **Security threats** of multi-tier execution engines 
- Speculative runtime compilation for **frugal systems** 
- **Profile-guided** detection of application parallelisation opportunities 
- **Securing** VMs through **dedicated** hardware 
- Minimising **energy impact** of garbage collection algorithms 

Selected Software Engineering Challenges €

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

AlaMVic: a generative approach

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



Early RMOD achievements

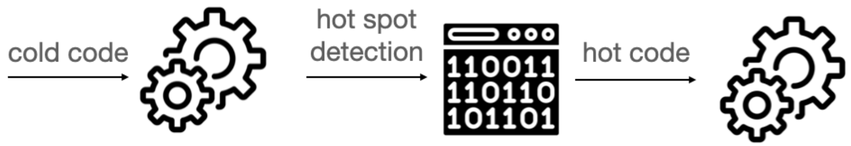
Dev side of things

- 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!



ENSTA BRETAGNE
TELECOM SudParis