

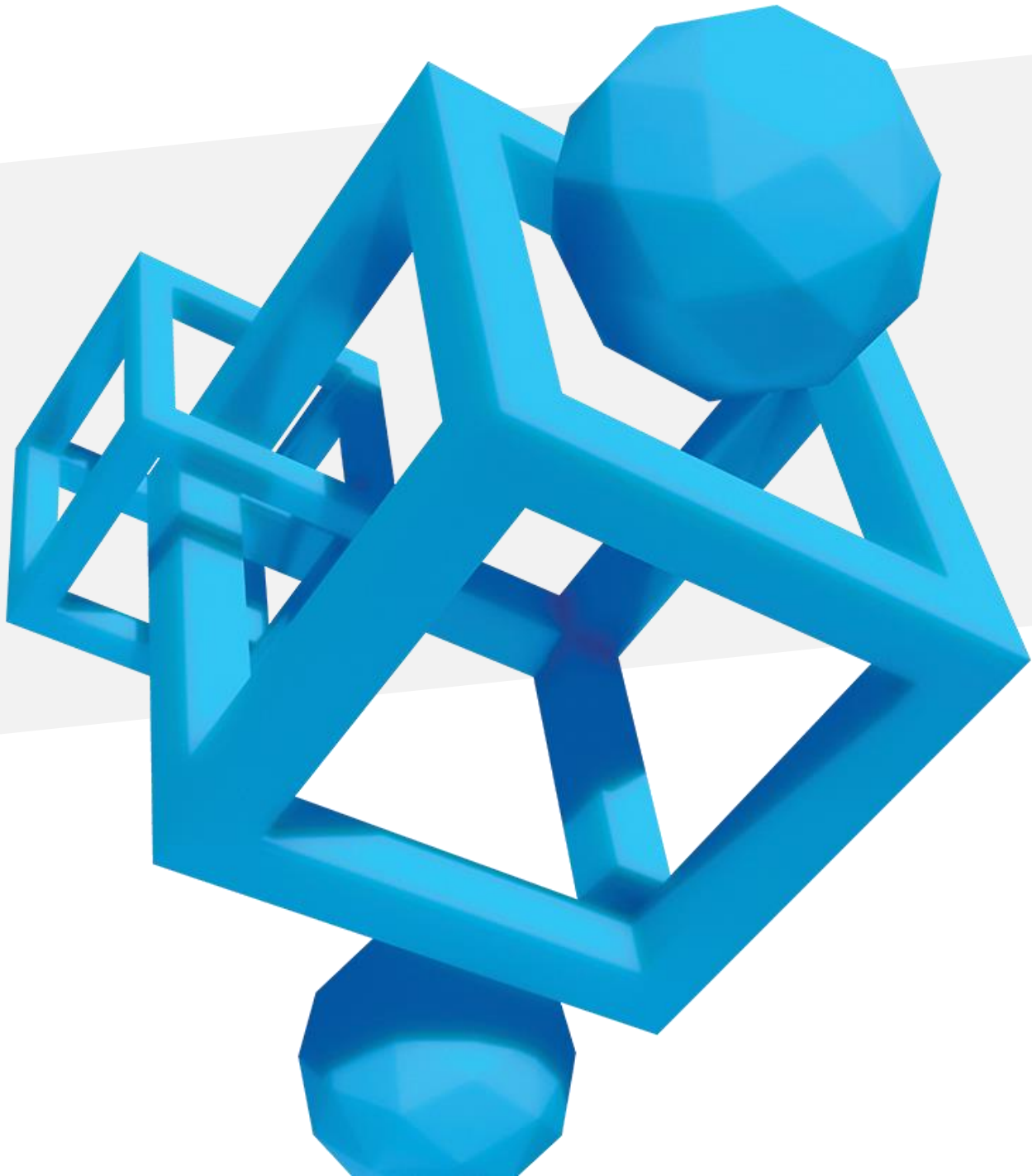


VM & JIT Profiling in VAST

Henrik Johansen

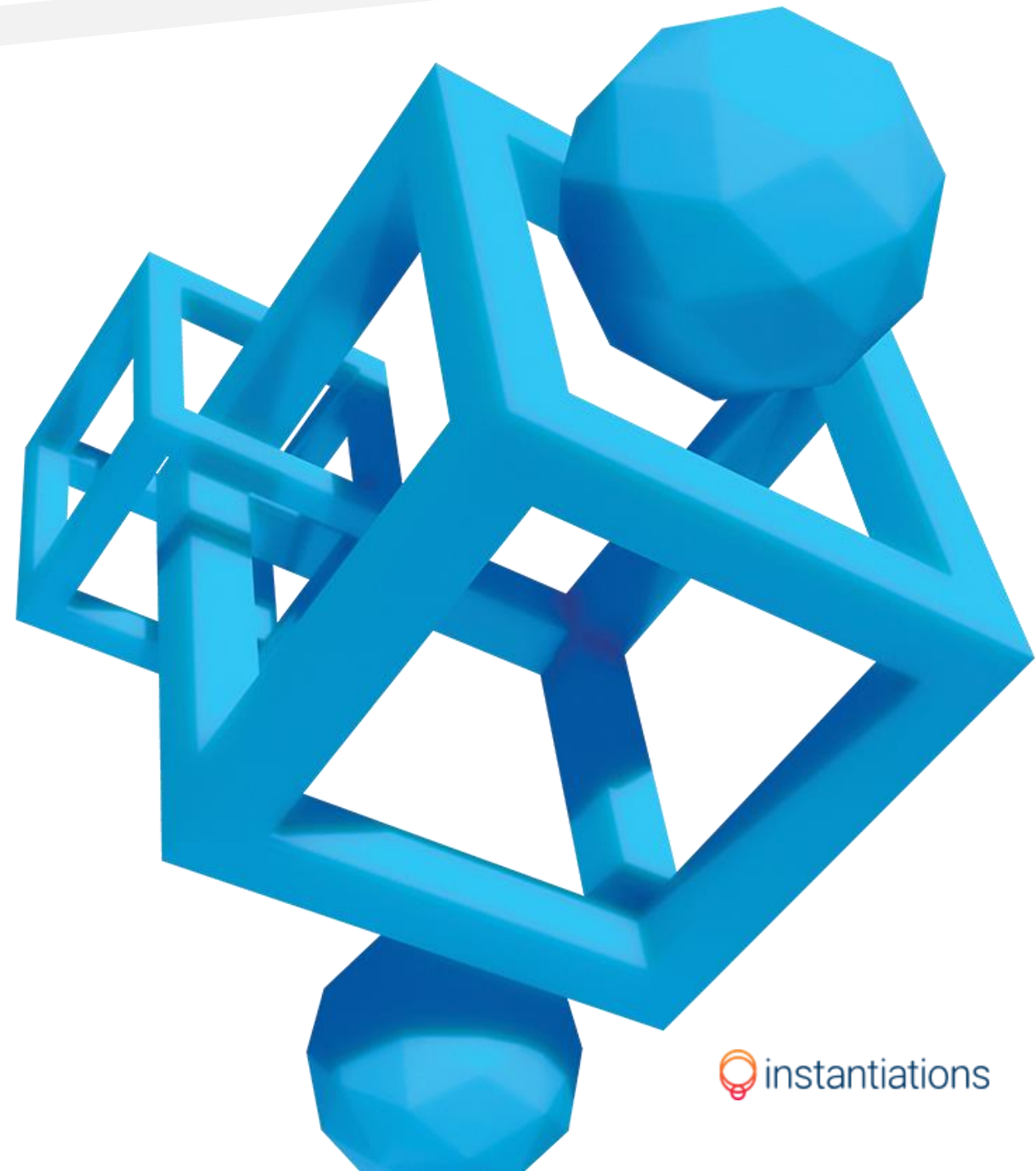
Senior Software Developer

✉ hjohansen@instantiations.com



Agenda

- VAST JIT overview
- VM profiling
 - Challenges
 - Solutions
- Demo



VAST JIT Overview

VAST JIT compiler overview

- ESUG 2019 – “JIT Compilation for VA Smalltalk”
- Based on LLVM
- Does not use the LLVM JIT library
 - Generates highly optimized code, but...
 - Large library footprint
 - Compile time too long
- Instead, bytecodes written using LLVM IRBuilder
 - At build time, emits machine code for bytecode templates
 - At runtime, templates are copied and patched with concretized values

Others do it too! (...now)

- <https://tonybaloney.github.io/posts/python-gets-a-jit.html>

What is a copy-and-patch JIT?

Never heard of a copy-and-patch JIT? Don't worry, nor had I and nor have most people. It's an idea only proposed recently in 2021 and designed as a fast algorithm for dynamic language runtimes.

VM Profiling

Image-side performance samplers

- Work great!
- ... in most cases

14.1.0arm64-b577-test: Method Execution

File Edit Methods Time Options Tree

☐ Total Time ☒ Local Time

(65.0%) UnicodeWriteStream>>#nextPutAll: (91.0%,65.0%) [8 scavenges]

(14.7%) Object>>#~ = (14.7%,14.7%) [1 scavenger][1 ggc]

<< >> 1 of 1 occurrences

~ = anObject

"Answer a Boolean which is false when the receiver and anObject are equivalent, and true otherwise."

^(self = anObject) not

12 Samples, 142 Scavenges, 2 Global GC's

14.1.0arm64-b577-test: Workspace

File Edit View Options Encoding Language


```
1 strings := {'hello'.
2 'ESUG'.
3 '\u{20AC}'} collect: [:e | UnicodeString value: e ].
4 stream := UnicodeString new writeStream.
5 (EsbSampler spyOn: [
6     1 to: 1000000 do:
7         [ix | stream
8             nextPutAll: (strings at: (ix \\ strings size) + 1) ] ]
9 ) browse
```


Other edge cases

- VM performance tests regressions
 - New LLVM versions
- Same hardware, same machine code, different results
 - Virtualization
 - Security Settings

Challenges

- Profilers work for known stack frame layouts
- Requires debug symbols

Function Stack	CPU Time: Total 	Module
▼ Total	100.0%	
▶ [Unknown stack frame(s)]	0.1%	
▼ RtlUserThreadStart	99.9%	ntdll.dll
▼ BaseThreadInitThunk	99.9%	KERNEL32.DLL
▼ func@0x140002481	98.6%	abt.exe
▼ func@0x140001fa0	98.6%	abt.exe
▼ func@0x140001820	98.6%	abt.exe
▼ EsExecuteImage	98.6%	esvm40.dll
▼ func@0x180008020	98.6%	esvm40.dll
▼ func@0x180066db0	98.6%	esvm40.dll
▼ func@0x180066eb0	98.6%	esvm40.dll
▼ [Unknown stack frame(s)]	71.1%	
[Outside any known module]	71.1%	
▶ func@0x1800906b0	27.4%	esvm40.dll

Solutions

- Build your own!
- Or leverage existing tools
 - Intel VTune JIT Profiling API (x86/x64 Windows/Linux)
 - jitdump file format + perf tool (x86/x64/aarch64 Linux)

Demo

Thanks for attending!

Questions?

Henrik Johansen
Senior Software Developer

✉ hjohansen@instantiations.com

Contact

General Inquiry
info@instantiations.com

Sales
sales@instantiations.com

VAST Support Portal
vast-support.instantiations.com

North America, Toll Free
855 476 2558

International
+1 503 263 0058