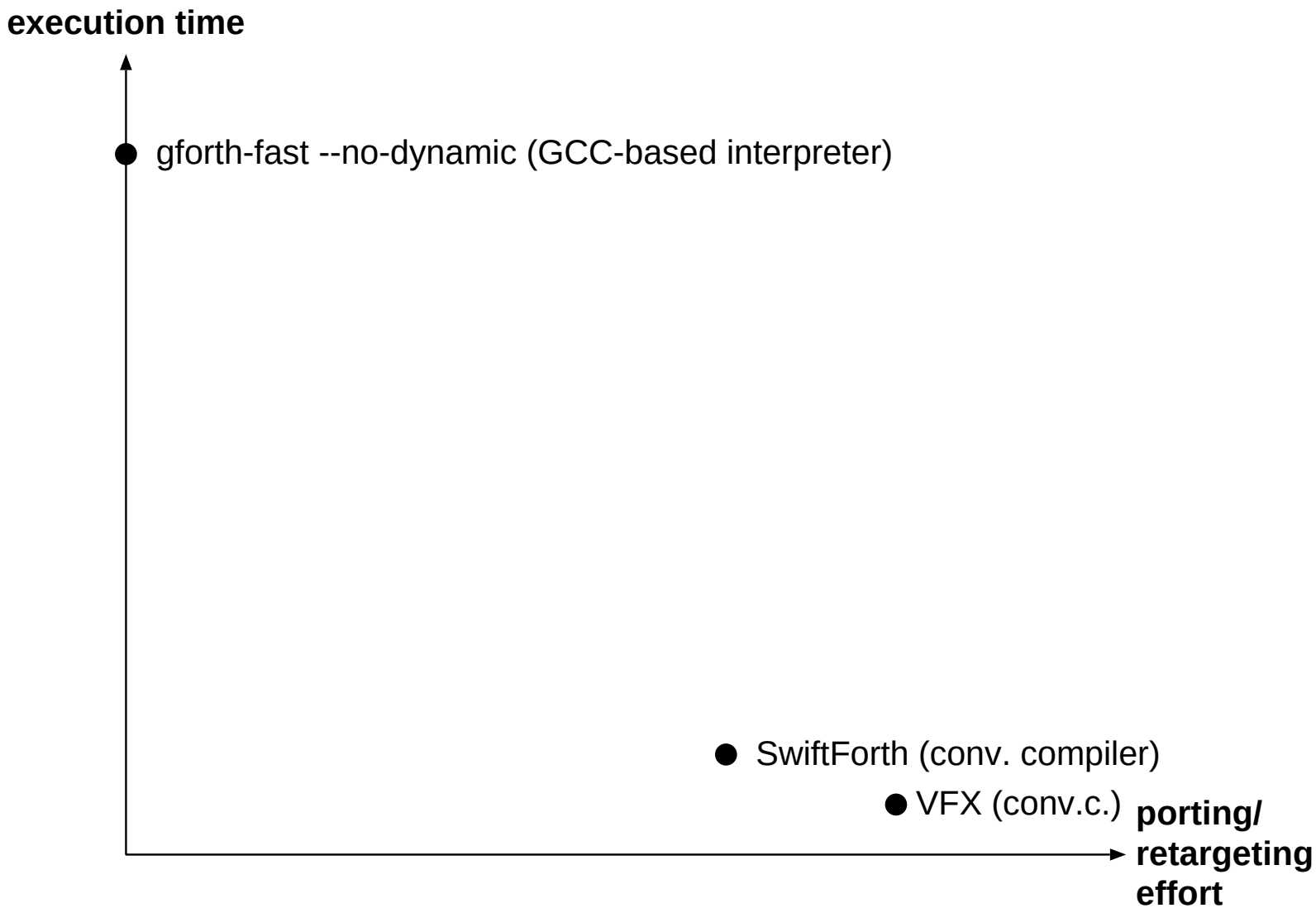
Code-Copying Compilation in Production

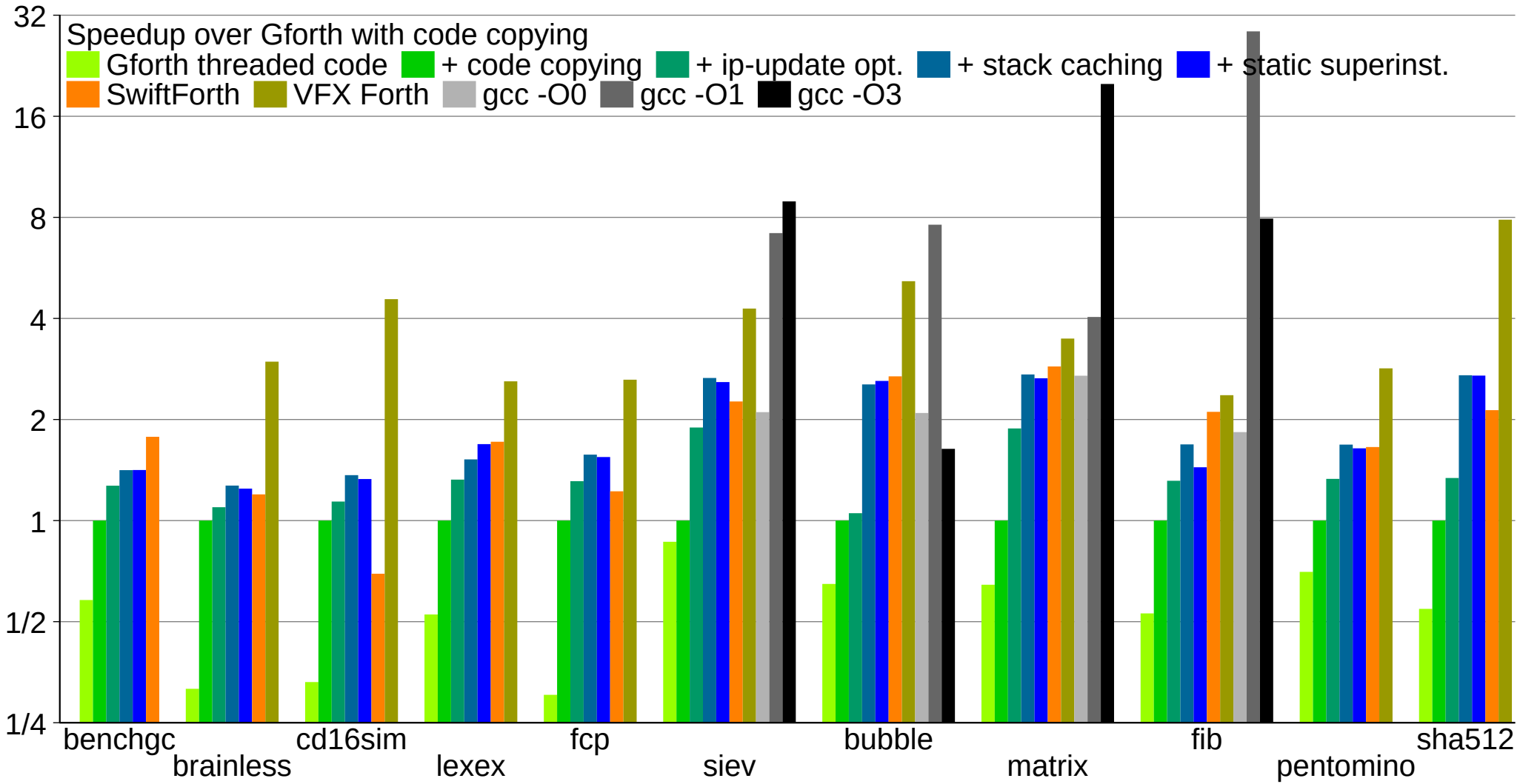
An Experience Report

M. Anton Ertl, TU Wien
Bernd Paysan, net2o

Objectives


execution time
gforth-fast --no-dynamic (GCC-based interpreter)
SwiftForth (conv. compiler)
VFX (conv.c.)
porting/
retargeting
effort

Performance (Core i5-1135G7)



Porting effort

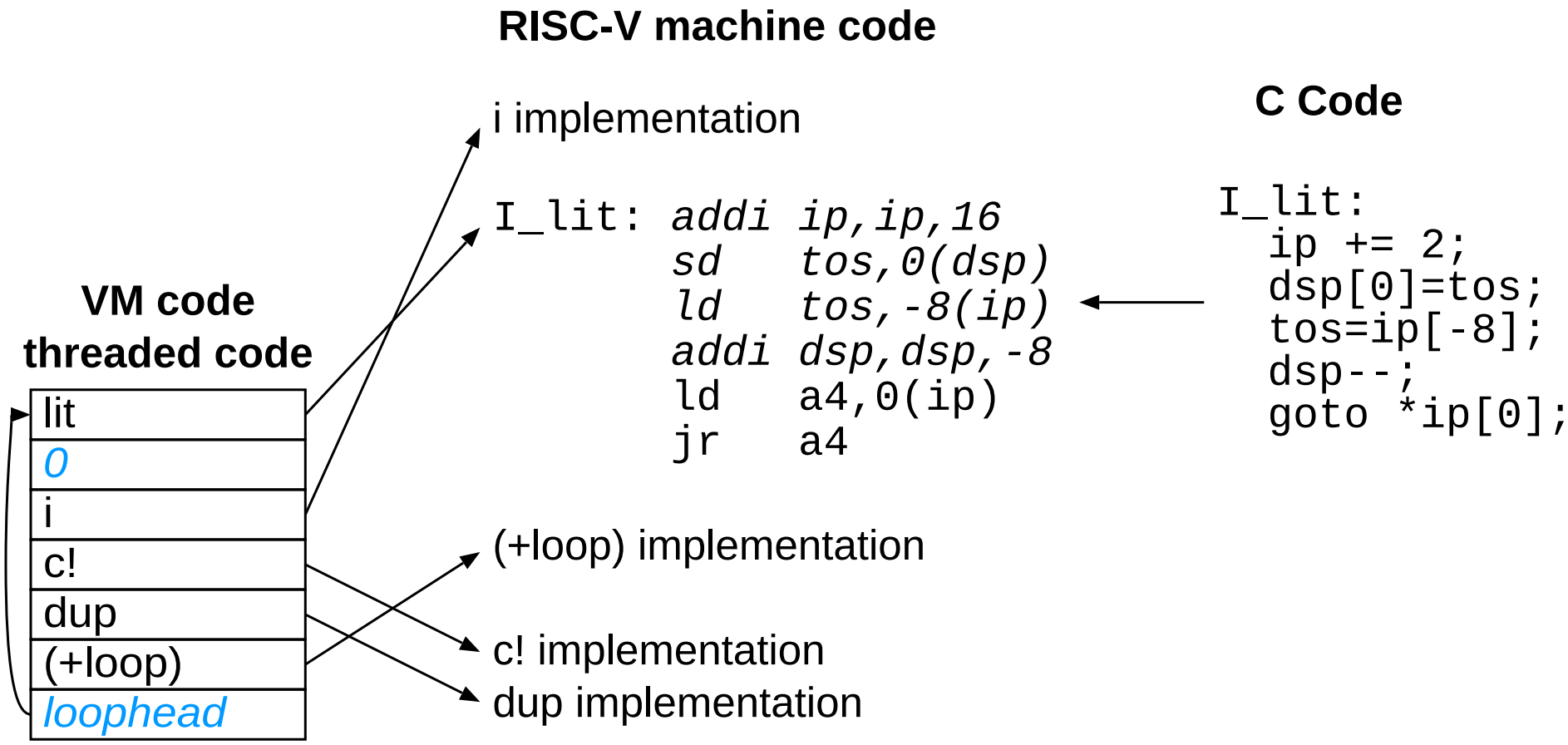
Gforth

AMD64 ≈ 50 SLOC, 2003
RISC-V 6 SLOC, 2017
not counted Assembler, Disassembler
more Alpha, ARM A32/T32, ARM A64, HPPA, IA-32, IA-64, Loongarch, SPARC, PowerPC, PowerPC64
only threaded code 68000, MIPS, unknown

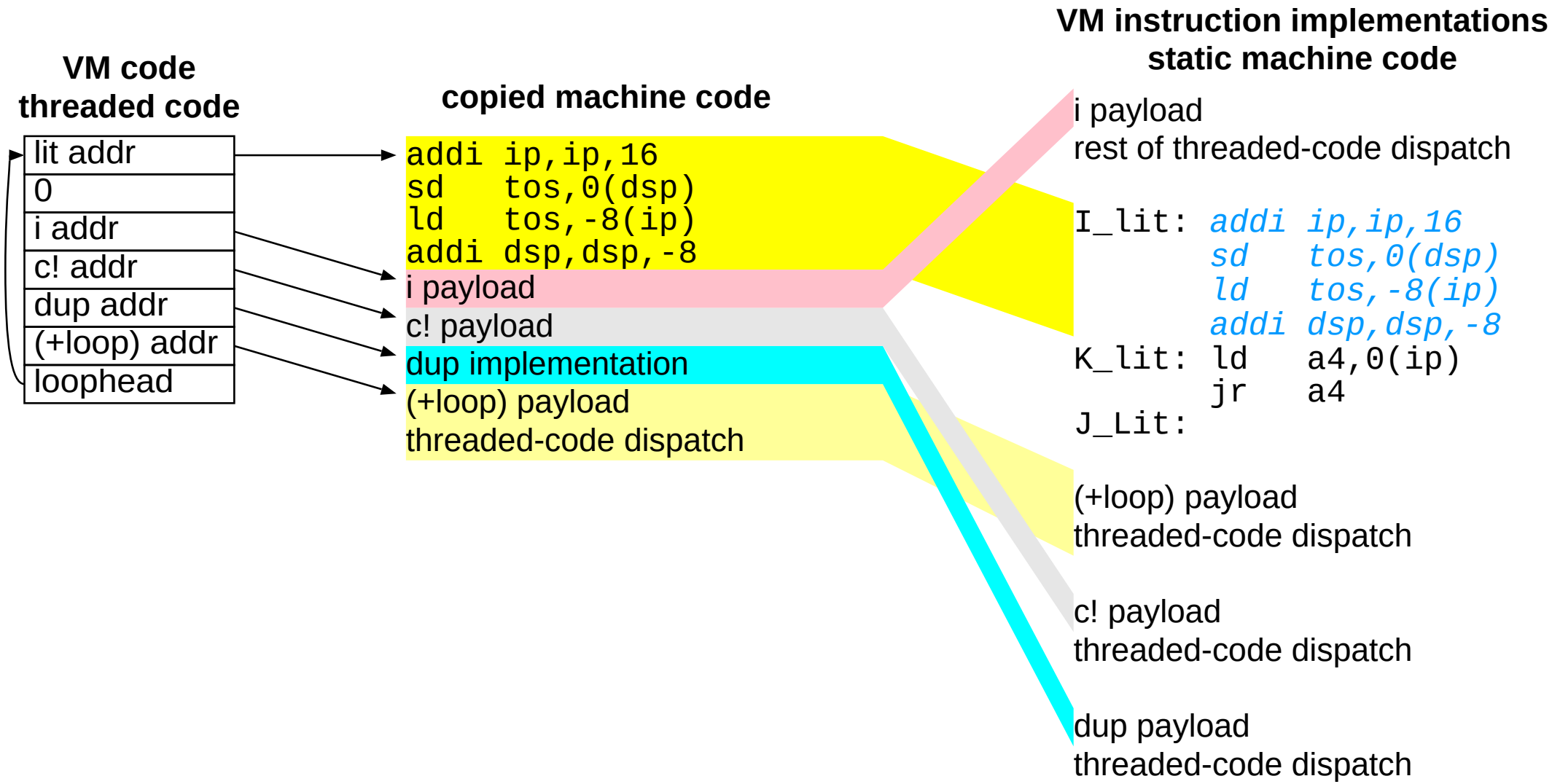
SwiftForth

AMD64 ≈ 7000 lines
IA-32 ≈ 7000 lines

Threaded code (`gforth-fast -no-dynamic`)



Code copying (`gforth-fast`, disabled other optimizations)



Relocatability

engine()

```
i implementation
I_lit:  addi  ip,ip,16
        sd    tos,0(dsp)
        ld    tos,-8(ip)
        addi  dsp,dsp,-8
K_lit:  ld    a4,0(ip)
        jr    a4
J_lit:
(+loop) implementation
c! implementation
dup implementation
```

engine2()

```
i implementation
        .skip 4
I_lit:  addi  ip,ip,16
        sd    tos,0(dsp)
        ld    tos,-8(ip)
        addi  dsp,dsp,-8
K_lit:  ld    a4,0(ip)
        jr    a4
J_lit:  .skip 4
(+loop) implementation
        .skip 4
c! implementation
        .skip 4
dup implementation
```

Why does it work?

- Register allocation
  has to be the same at every `goto *` and label
- Instruction sets
  Instructions are independent (with exceptions)
  Compiler does not insert labels inside the exceptions
- If all else fails
  Fall back to threaded code

Hurdles and workarounds

- Code reordering
  No loops and few ifs in VM instruction implementations
  Extract such cases into separate functions
- Code deduplication (one indirect branch for all `goto *`)
  Have only one `goto *`, and copy that
- Code duplication
  Insert empty `asm` statements: duplication appears expensive
- Bad copy propagation
  wait for better compiler version
  `-fno-tree-vectorize`
  use a different compiler
- `__builtin___clear_cache()` buggy
  Use machine-specific code

Does it work with security features?

- No RWX memory (`W^X`)
  Jump through OS-specific hoops
  RX mapping and RW mapping?
- Spectre
  `-mindirect-branch=thunk-inline`
  Slowdown factor 2.1–7.6 (without code copying 7.5–18.1) on Ryzen 3900X
- Control-flow protection
  `-fcf-protection=full` works
  1.45× more instructions, 1.04× more cycles on Ryzen 8700G
  Pointless in Gforth

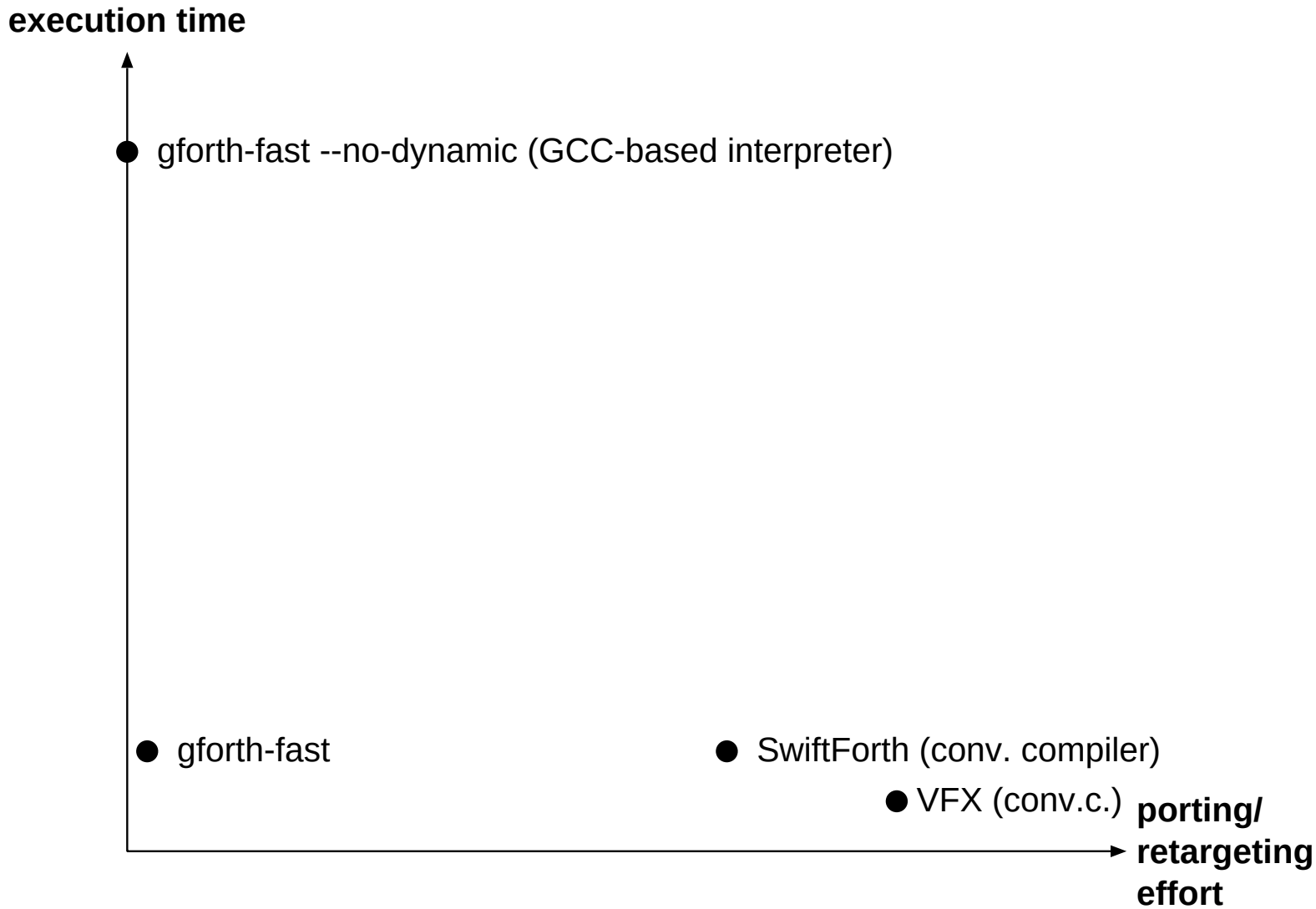
Alternative approaches

- Functions with tail calls instead of `goto *` in big function
  Allows more code snippets
  How to determine the end of a code snippet?
- Extract code snippets from object files at build time
  instead of from the executable code at run-time
- Patch the code (copy-and-patch) instead of accessing VM code
  Shorter code, fewer indirect jumps
- No production system has used any such approaches and stuck with them
  Yet

Optimizations

	Gforth	SwiftForth
code snippets	GCC generated	assembly
code generator	code copying ($\approx$ 500 lines)	code copying
literal operands	from threaded code	patched
control flow	through threaded code	patched
threaded-code IP updates	optimized ($+834/-316$ lines)	none exist
multi-state stack caching	3 registers	x
static superinstructions	56	346 (1819 lines)
tail-call optimization	x	✓
	$\approx$ 5000 lines overall	

Objectives


execution time
gforth-fast --no-dynamic (GCC-based interpreter)
gforth-fast
SwiftForth (conv. compiler)
VFX (conv.c.)
porting/
retargeting
effort

Conclusion

- Objectives: good performance, small porting effort
- Concatenate machine code snippets
  VM-level immediate operands from VM code
  Control flow through VM (threaded) code
  fall back to threaded code if anything is amiss
- Determine relocatability by comparing two copies
- Workarounds for all hurdles to date have been found
- Objectives: competetive with SwiftForth, 6–50 SLOC/port, many ports
- In production since 2003