STOCHASTIC SUPEROPTIMIZATION

eric schkufza, rahul sharma, alex aiken
stanford university

asplos 2013

MICRO-OPTIMIZATION

llvm -O0 (100 LOC)

```
L0:
  movq rdi, -8(rsp)
  movq rsi, -16(rsp)
  movl edx, -20(rsp)
  movl ecx, -24(rsp)
  movq r8, -32(rsp)
  movq -16(rsp), rsi
  movq rsi, -48(rsp)
  movq -48(rsp), rsi
  movabsq 0xffffffff, rdi
  andq rsi, rdi
  movq rdi, -40(rsp)
  movq -48(rsp), rsi
  shrq 32, rsi
  movabsq 0xffffffff, rdi
  andq rsi, rdi
  movq rdi, -48(rsp)
  movq -40(rsp), rsi
  movq rsi, -72(rsp)
  movq -48(rsp), rsi
  movq rsi, -80(rsp)
  movl -24(rsp), esi
  imulq -72(rsp), rsi
  movq rsi, -56(rsp)
  movl -20(rsp), esi
  imulq -72(rsp), rsi
  movq rsi, -72(rsp)
  movl -20(rsp), esi
  ...
```

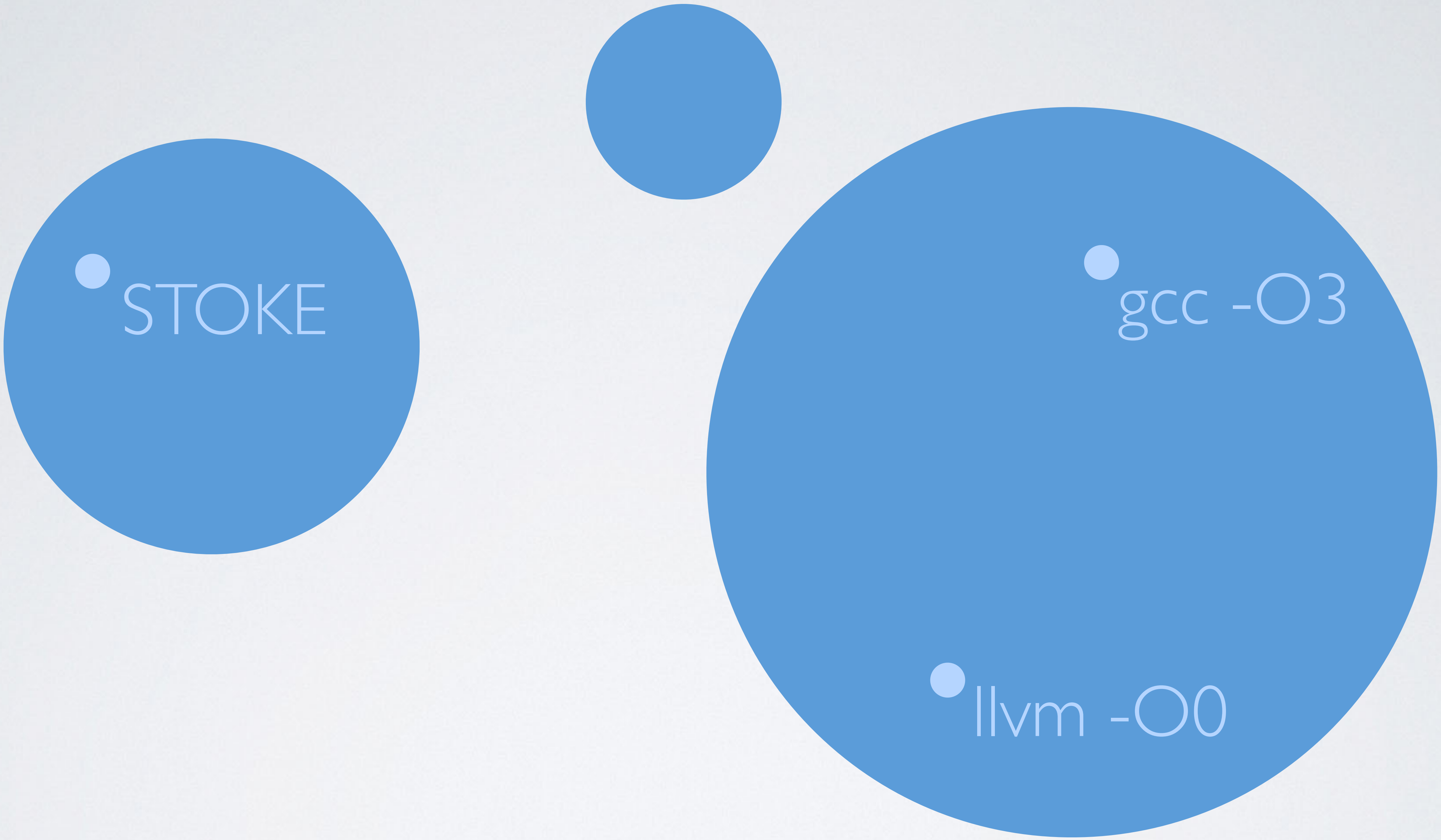
gcc -O3 (29 LOC)

```
.L0:
  movq rsi, r9
  movl ecx, ecx
  shrq 32, rsi
  andl 0xffffffff, r9d
  movq rcx, rax
  movl edx, edx
  imulq r9, rax
  imulq rdx, r9
  imulq rsi, rdx
  imulq rsi, rcx
  addq rdx, rax
  jae .L2
  movabsq 0x100000000, rdx
  addq rdx, rcx
.L2:
  movq rax, rsi
  movq rax, rdx
  shrq 32, rsi
  salq 32, rdx
  addq rsi, rcx
  addq r9, rdx
  adcq 0, rcx
  addq r8, rdx
  adcq 0, rcx
  addq rdi, rdx
  adcq 0, rcx
  movq rcx, r8
  movq rdx, rdi
```

STOKE (11 LOC)

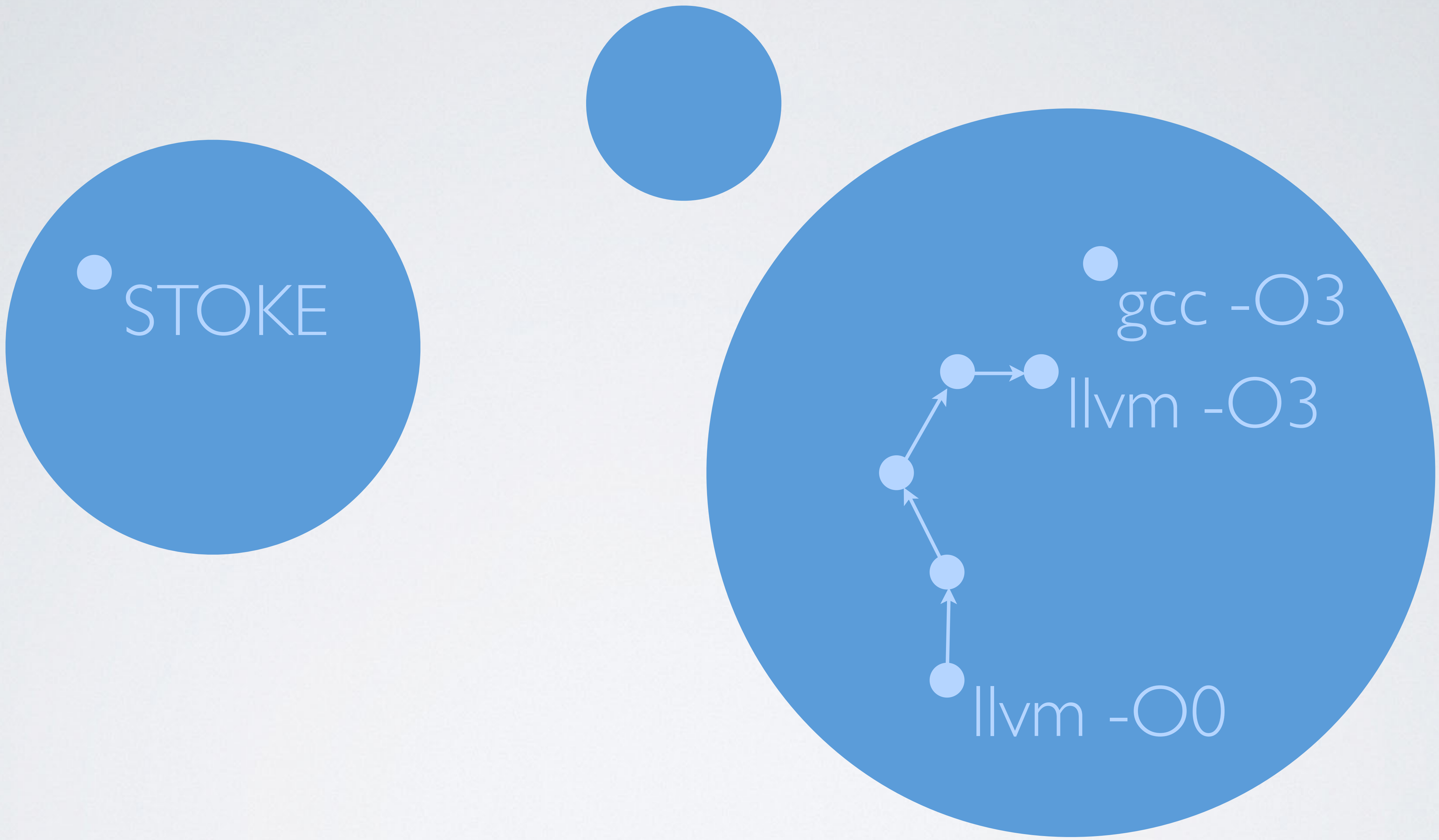
```
.L0:
  shlq 32, rcx
  movl edx, edx
  xorq rdx, rcx
  movq rcx, rax
  mulq rsi
  addq r8, rdi
  adcq 9, rdx
  addq rdi, rax
  adcq 0, rdx
  movq rdx, r8
  movq rax, rdi
```

EXISTING APPROACHES



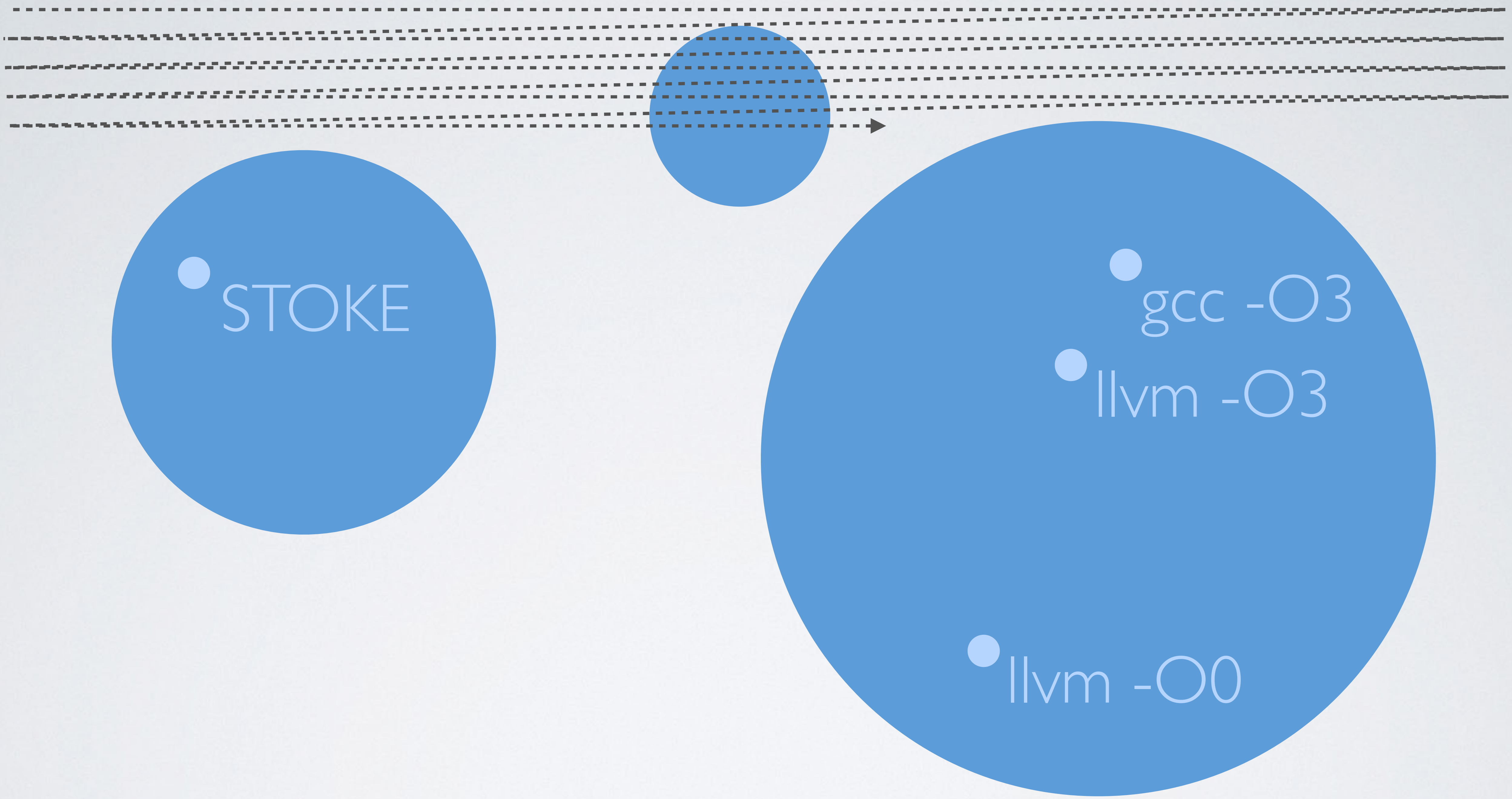
- **The Abstract Program Space**

EXISTING APPROACHES



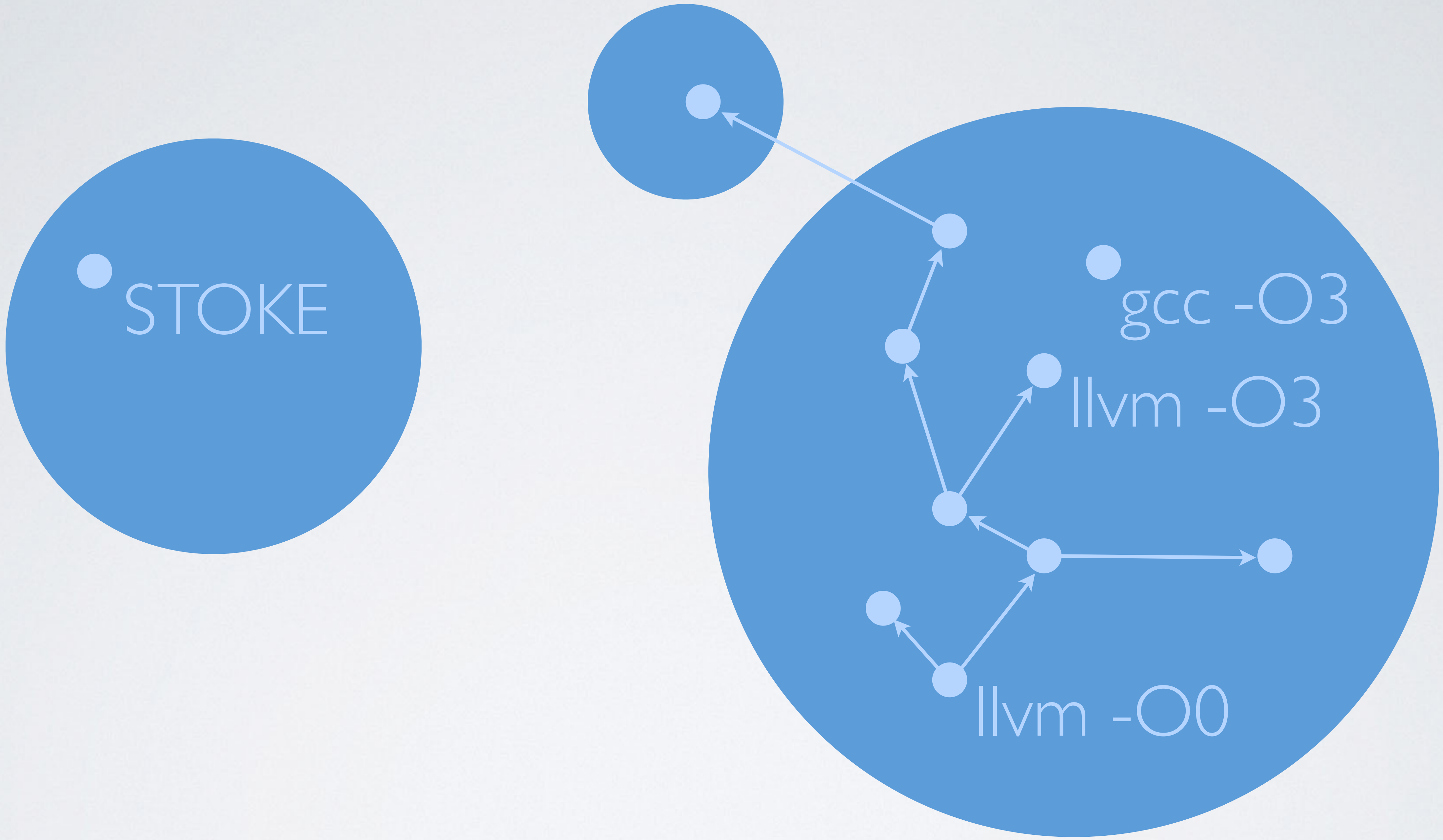
- **Traditional Compilers:** Consistently good, but not optimal

EXISTING APPROACHES



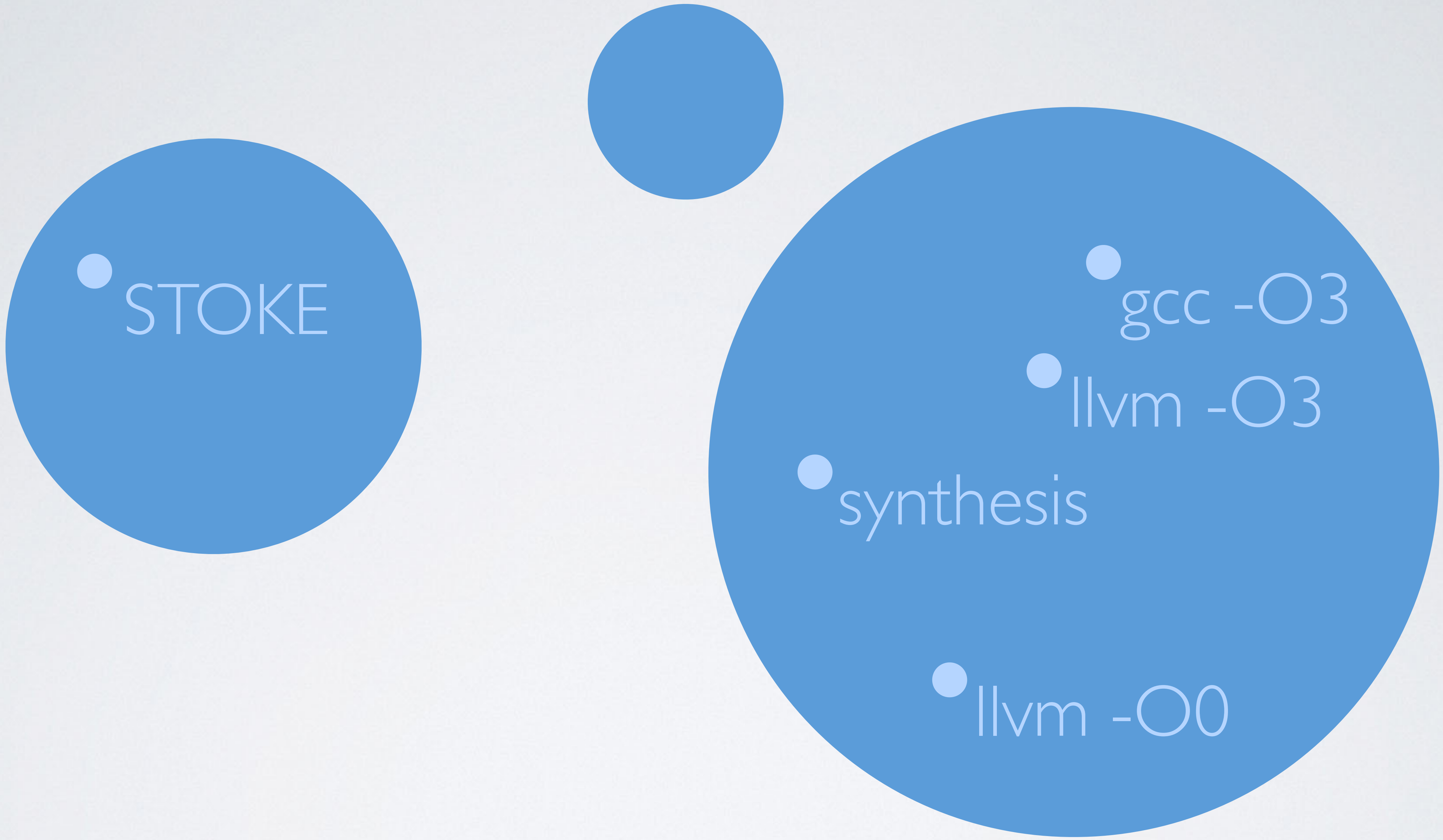
- **Bruteforce Enumeration:** Guess and check all possible implementations [massalin 87][bansal 06][bansal 08]

EXISTING APPROACHES



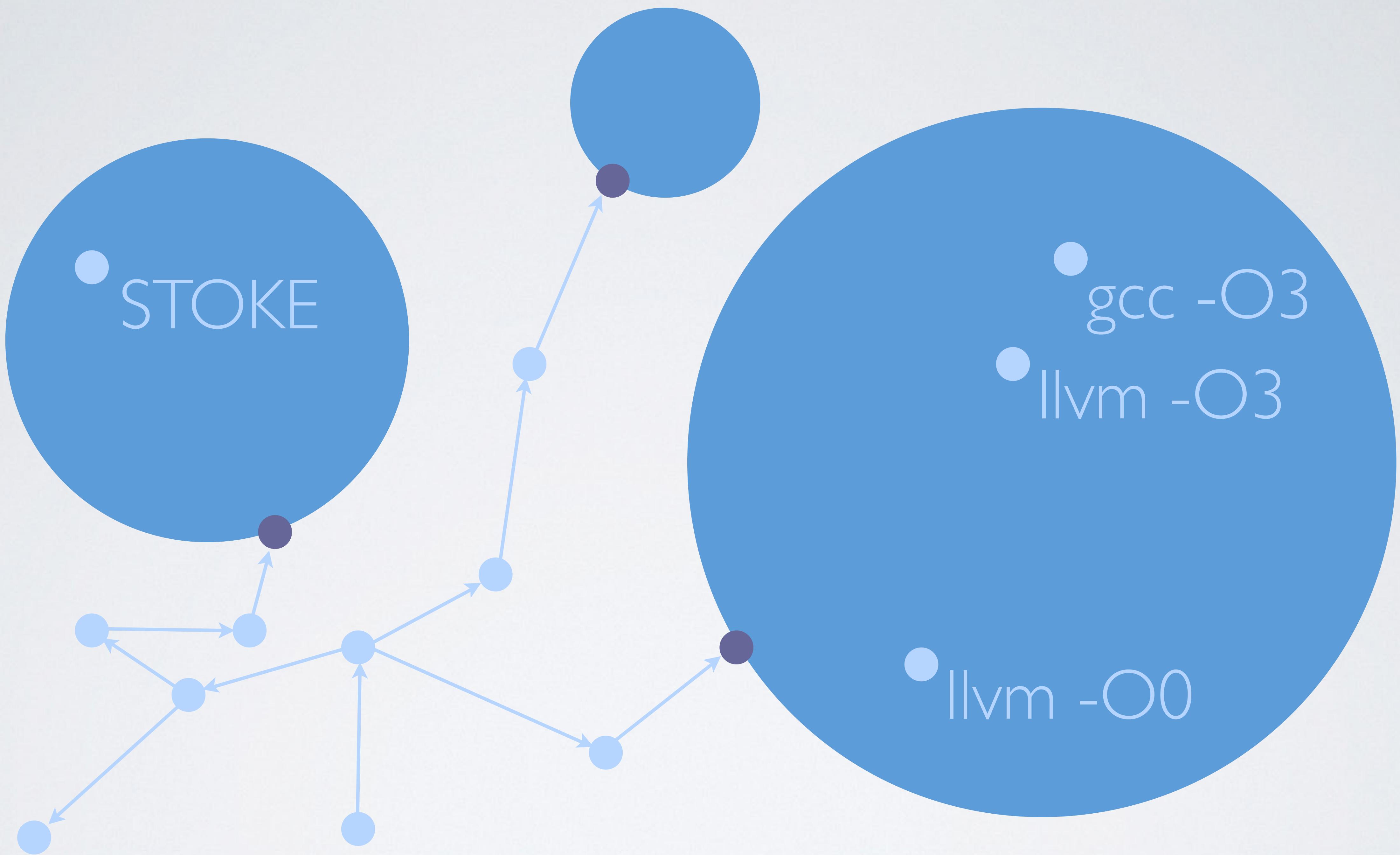
- **Equality Preservation:** Expert-written rules for traversing the smaller space of correct implementations [joshi 02][tate 09]

EXISTING APPROACHES



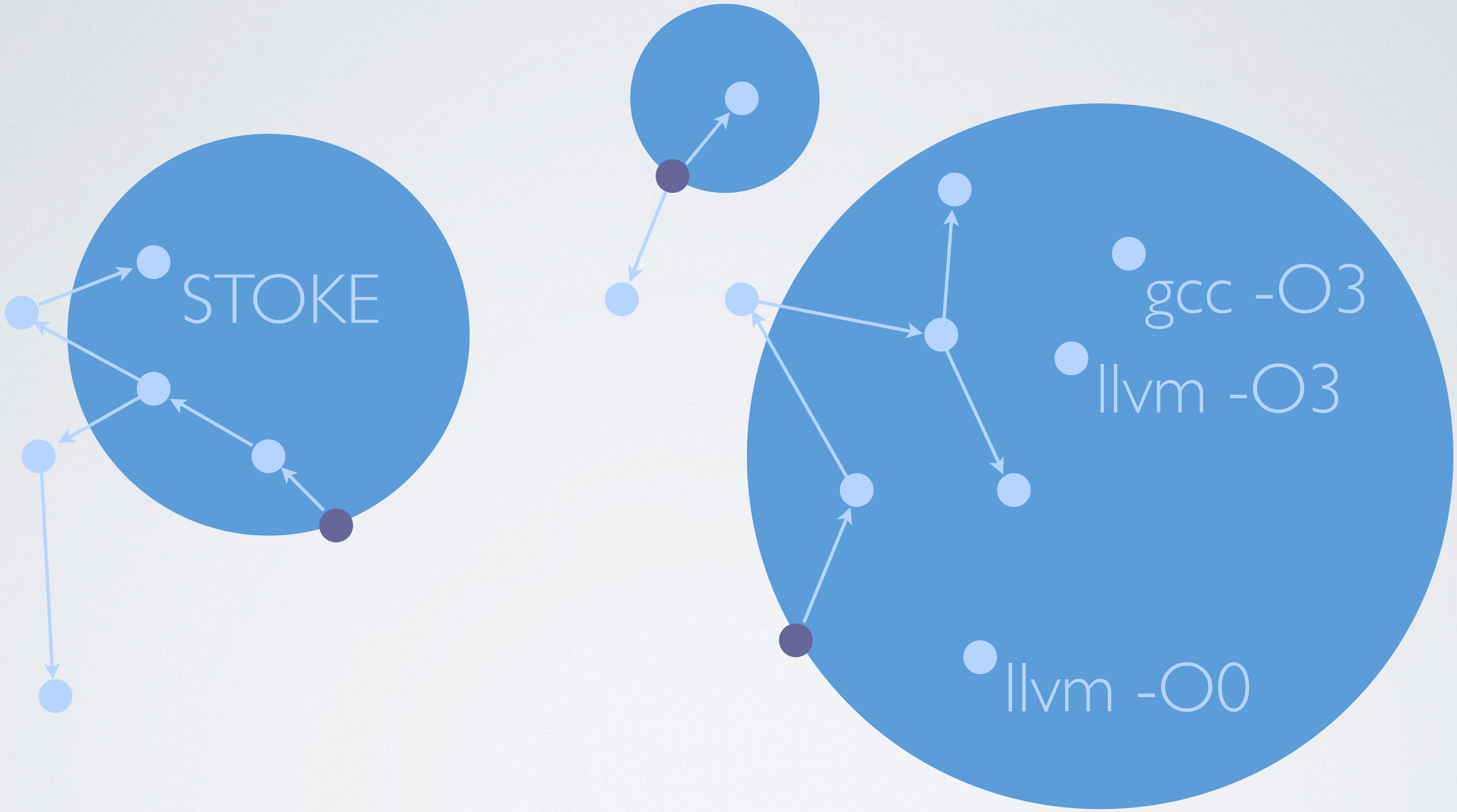
- **Program Synthesis:** Intermediate-level RTL, produce one correct implementation [gulwani 11][solar-lezama 06][liang 10]

OUR APPROACH



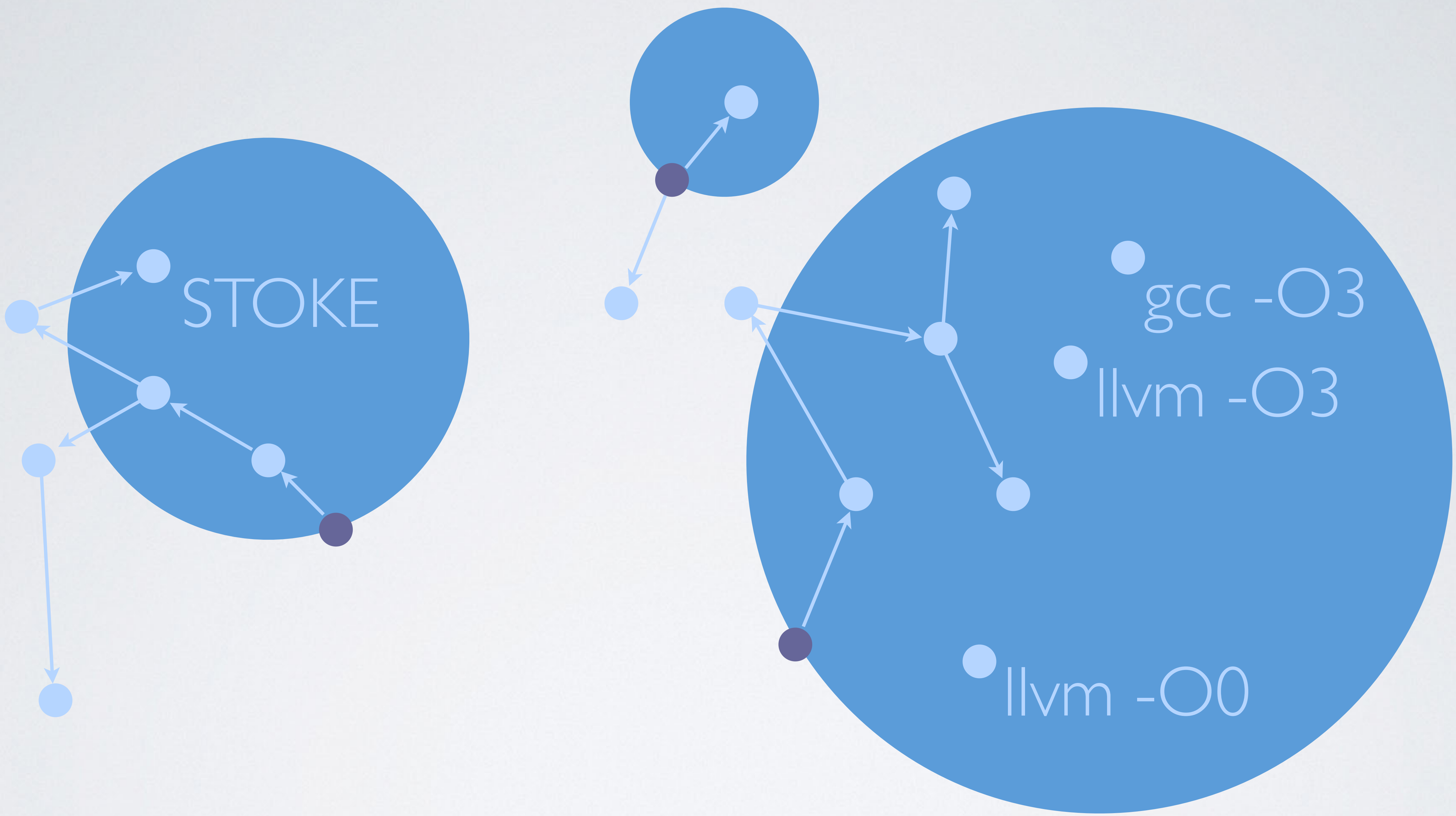
- **Random Search:** Synthesis threads begin from random codes and attempt to discover regions of correct implementations

OUR APPROACH



- **Random Search:** Optimization threads search each region and improve code, sometimes ignoring correctness

OUR APPROACH



- **Result:** A superoptimization technique that scales beyond all previous approaches to interesting real world kernels

WHAT'S REQUIRED

- **Search procedure:** Markov Chain Monte Carlo (MCMC) sampling: the only known tractable solution method for high dimensional irregular search spaces [andrieu 03][chenney 00]
- **Cost Function:** Measures the quality of a **rewrite** with respect to the **target** code
  - **Synthesis:** $\text{cost}(r; t) = \text{eq}(r; t)$
  - **Optimization:** $\text{cost}(r; t) = \text{eq}(r; t) + \text{perf}(r; t)$

MCMC SAMPLING

- **Algorithm:**

  1. Select an initial program
  2. Repeat (millions, even billions of times)
     1. Propose a random modification and evaluate cost
     2. If (cost decreased) { accept }
     3. If (cost increased) { with some probability accept anyway }

TECHNICAL DETAILS

- **Ergodicity:** Random transformations should be sufficient to cover entire search space.

- **Throughput:** Runtime cost to propose and evaluate should be minimal

- **Symmetry:** Probability of transformation equals probability of undoing it

- **Result:** Intelligent hill climbing method, robust against local minima, desirable limiting properties

TRANSFORMATIONS

original

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
 ...
```

TRANSFORMATIONS

original

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
 ...
```

insert

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
imulq rsi, rdx
...    ^
```

TRANSFORMATIONS

original

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
 ...
```

insert

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
imulq rsi, rdx
...    ^
```

delete

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
...
```

TRANSFORMATIONS

original

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
 ...
```

insert

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
imulq rsi, rdx
...    ^
```

delete

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
...
```

instruction

```
...
movl ecx, ecx
shrq 32, rsi
salq 16, rcx
movq rcx, rax
movl edx, edx
imulq r9, rax
...
```

TRANSFORMATIONS

original

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
 ...
```

insert

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
imulq rsi, rdx
...    ^
```

opcode

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
subl edx, edx
imulq r9, rax
...
```

delete

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
...
```

instruction

```
...
movl ecx, ecx
shrq 32, rsi
salq 16, rcx
movq rcx, rax
movl edx, edx
imulq r9, rax
...
```

TRANSFORMATIONS

original

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
 ...
```

insert

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
imulq rsi, rdx
...    ^
```

opcode

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
subl edx, edx
imulq r9, rax
...
```

delete

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
~~movq rcx, rax~~
movl edx, edx
imulq r9, rax
...
```

instruction

```
...
movl ecx, ecx
shrq 32, rsi
salq 16, rcx
movq rcx, rax
movl edx, edx
imulq r9, rax
...
```

operand

```
...
movl ecx, ecx
shrq 32, rcx
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
...
```

TRANSFORMATIONS

original

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
 ...
```

insert

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
imulq rsi, rdx
...    ^
```

opcode

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
subl edx, edx
imulq r9, rax
...
```

delete

```
...
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
~~movq rcx, rax~~
movl edx, edx
imulq r9, rax
...
```

instruction

```
...
movl ecx, ecx
shrq 32, rsi
salq 16, rcx
movq rcx, rax
movl edx, edx
imulq r9, rax
...
```

swap

```
...
movl ecx, ecx
movl edx, edx
andl 0xffffffff, r9d
movq rcx, rax
shrq 32, rsi
imulq r9, rax
...
```

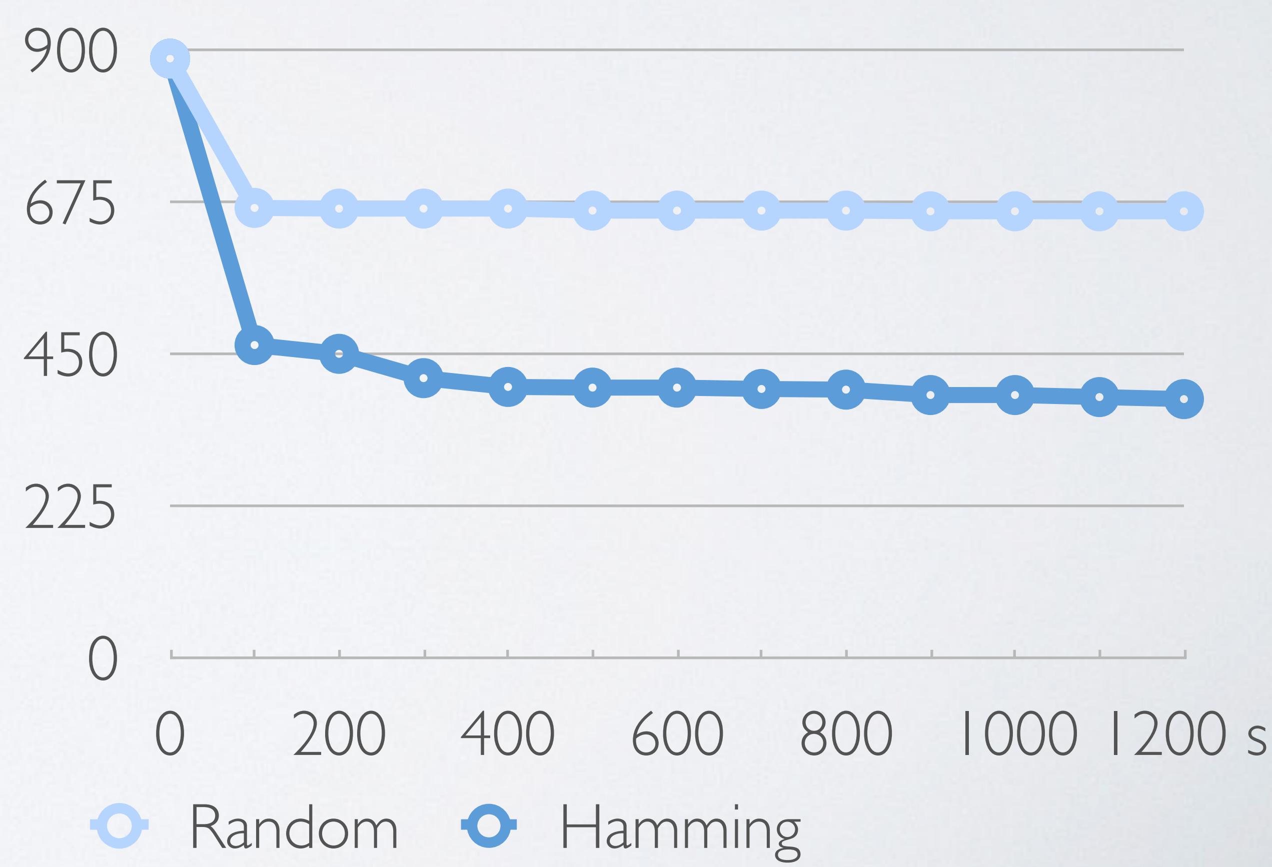
operand

```
...
movl ecx, ecx
shrq 32, rcx
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
...
```

EQUALITY FUNCTION

- **Smooth Metric:** Provides a useful notion of partial correctness which helps smooth the search space
- **Hamming Distance:** Simulate target and rewrite on test vectors and count error bits in live out registers

	ax	bx	cx	dx
T	1111	0000	0000	0000
R	0010	1000	1100	1111
	3			



EQUALITY FUNCTION

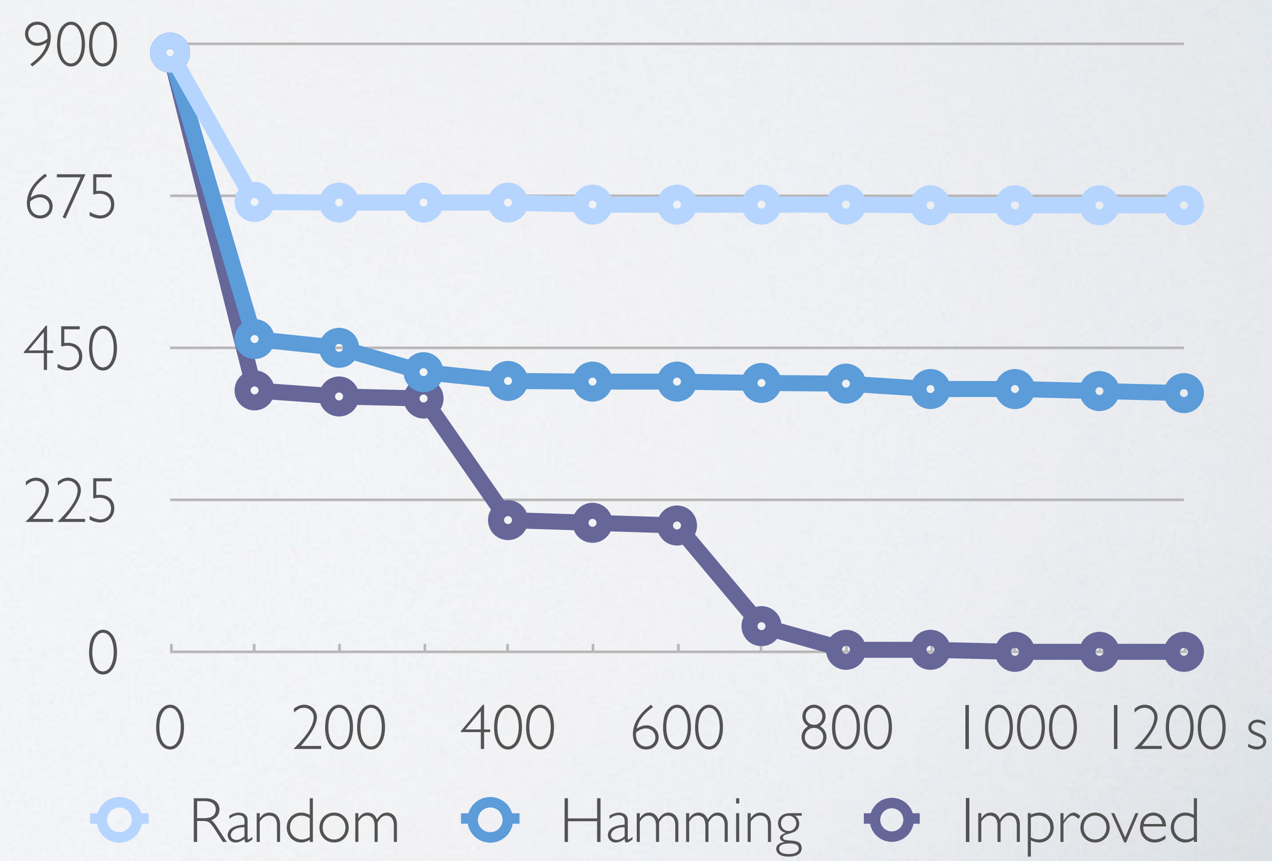
- **Improved Metric:** Relaxes that correct results appear in the correct registers

- **Implicit parallelization:** Allows search to experiment with multiple rewrites simultaneously

	`ax`	`bx`	`cx`	`dx`
`T`	`1111`	`0000`	`0000`	`0000`
`R`	`0010`	`1000`	`1100`	`1111`

min (3 + 0   3 + 1   2 + 1   0 + 1)

1



FORMAL CORRECTNESS

- **Periodic Checkpoints:** Verify last observed correct rewrite using an SMT solver
- **Proof Technique:** Encode both codes as SMT formulae and query the existence of inputs which will produce distinct outputs
- **Conflict-driven Testcases:** Formal correctness failures produce counter-examples which refine testcases [cadar 08]

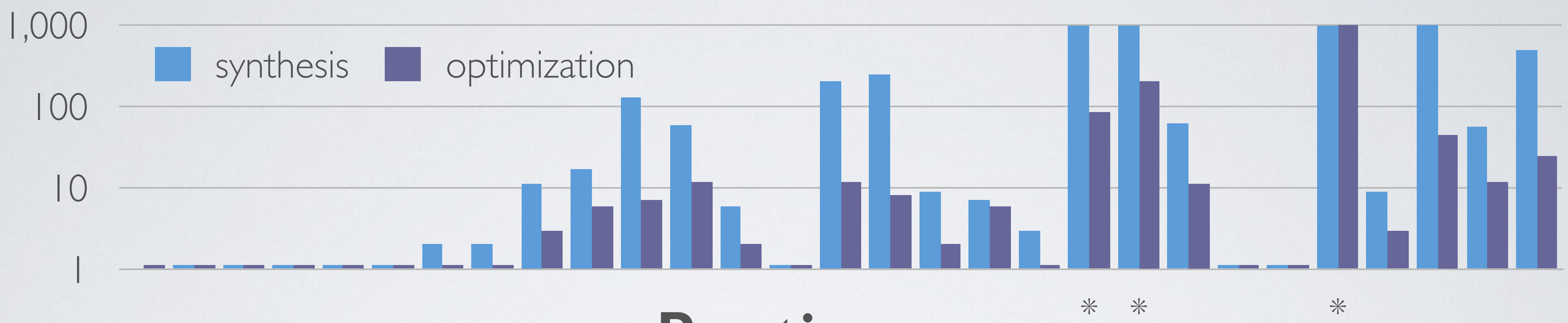
PERFORMANCE METRIC

- **Fast Approximation:** Approximate the runtime of a program by summing the average latencies of its instructions
- **Slow Precise Version:** Preserve the top-n most performant results discovered during search and rerank using compilation and evaluation on representative workloads

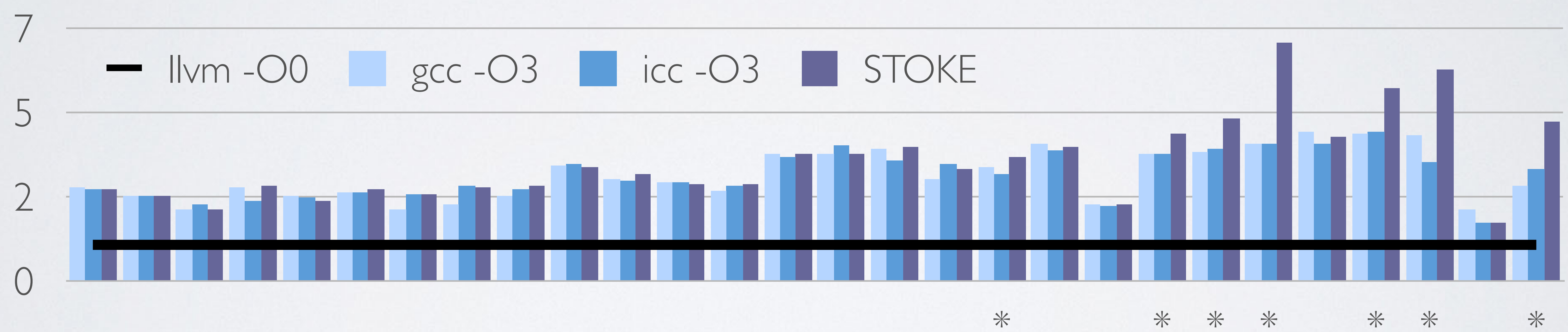
BENCHMARKS

- **Synthesis Kernels:** 25 loop-free kernels taken from A Hacker's Delight [gulwani 11]
- **Real World:** OpenSSL 128-bit integer multiplication montgomery multiplication kernel
- **Vector Intrinsics:** BLAS Level 1 SAXPY
- **Heap Modifying:** Linked List Traversal [bansal 06]

RESULTS



Runtime



Speedup

- **Experiments:** Target codes compiled using llvm -O0, STOKE matches or outperforms gcc and icc with full optimizations enabled

FUTURE WORK

- **Limitations:** Synthesis is ineffective for functions which distill inputs to boolean results; improved equality metric is slow for functions which produce pervasive memory side effects
- **Loops:** Higher throughput using JIT-compilation, more complex formal correctness checks (in submission)
- **Floating Point:** Trading bit-wise correctness for epsilon precision (in progress)
- **Hardware:** HDL optimization (in progress)

CONCLUSION

- **Micro-optimization:** For many interesting application domains, even a single assembly instruction can make a difference
- **Fun History of Related Work:**
  - **Brute-Force:** [massalin 87][bansal 06, 08]
  - **Equality Preservation:** [joshi 02][tate 09]
  - **Synthesis:** [gulwani 11, 13][solar-lezama 06][liang 10]
- **New approach:** Sacrifice completeness in favor of stochastic search and experiment with partially correct code; exciting results