

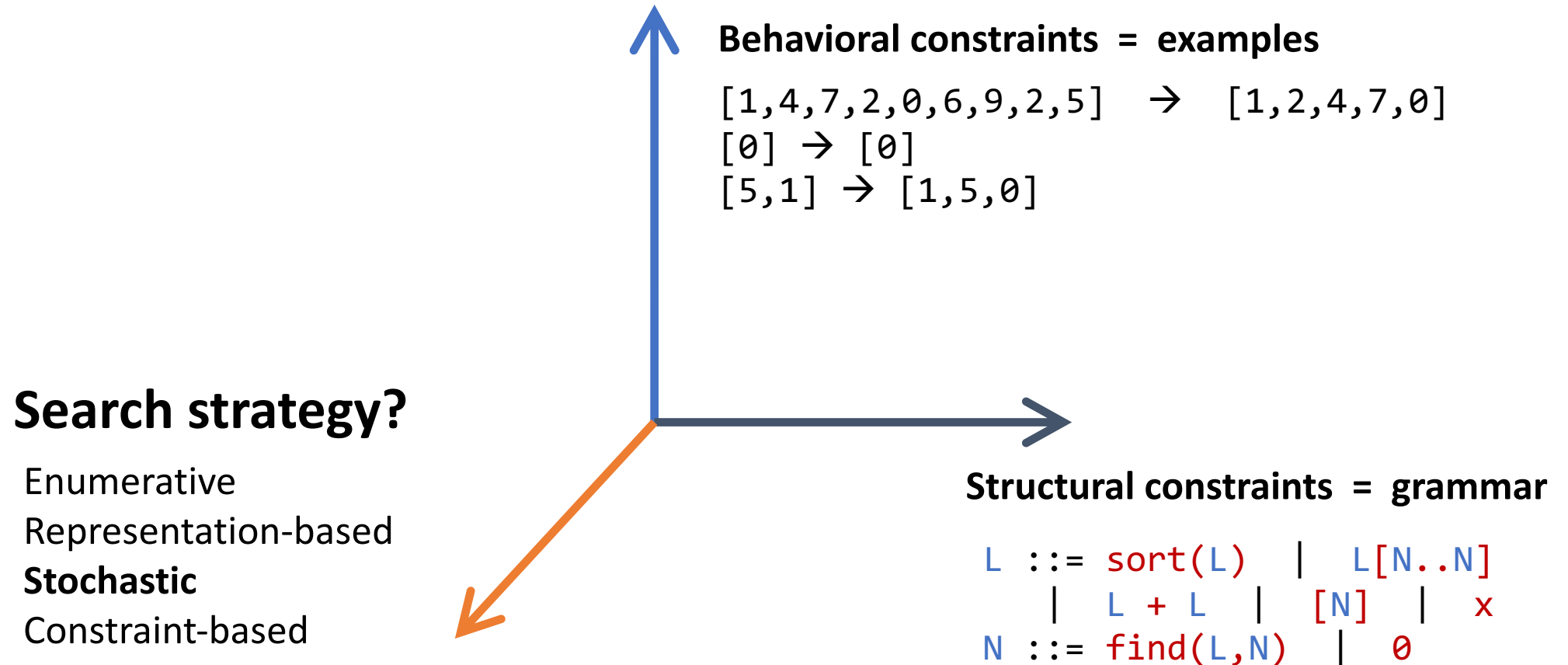
#11: Stochastic Search

Sankha Narayan Guria

EECS 700: Introduction to Program Synthesis



The problem statement



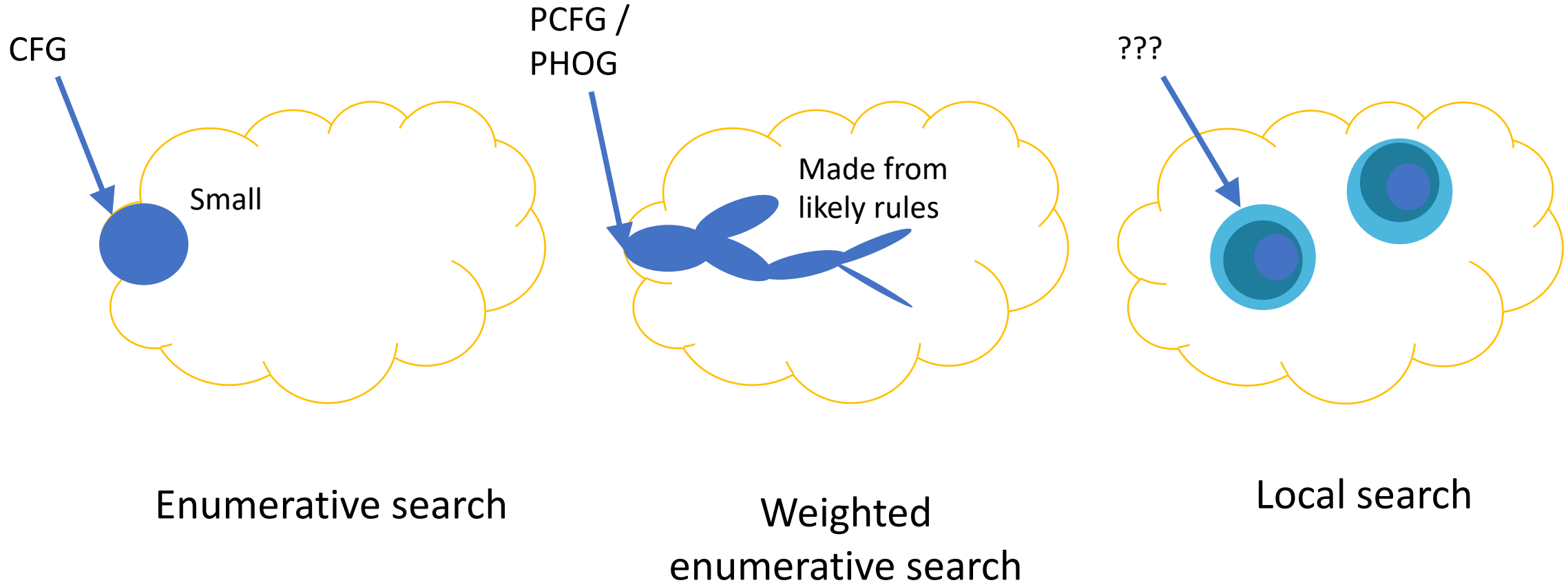
Stochastic search in synthesis

- Weimer, Nguyen, Le Goues, Forrest. *Automatically Finding Patches Using Genetic Programming*. ICSE'09
- Gissurarson, Applis, Panichella, van Deursen, Sands. *PropR: Property-Based Automatic Program Repair*. ICSE'22
- Schkufza, Sharma, Aiken: *Stochastic superoptimization*. ASPLOS'13
- Shi, Steinhardt, Liang: *FrAngel: Component-Based Synthesis with Control Structures*. POPL'19

Stochastic search in synthesis

- Weimer, Nguyen, Le Goues, Forrest. *Automatically Finding Patches Using Genetic Programming*. ICSE'09
- Gissurarson, Applis, Panichella, van Deursen, Sands. *PropR: Property-Based Automatic Program Repair*. ICSE'22
- Schkufza, Sharma, Aiken: *Stochastic superoptimization*. ASPLOS'13
- Shi, Steinhardt, Liang: *FrAngel: Component-Based Synthesis with Control Structures*. POPL'19

Search space

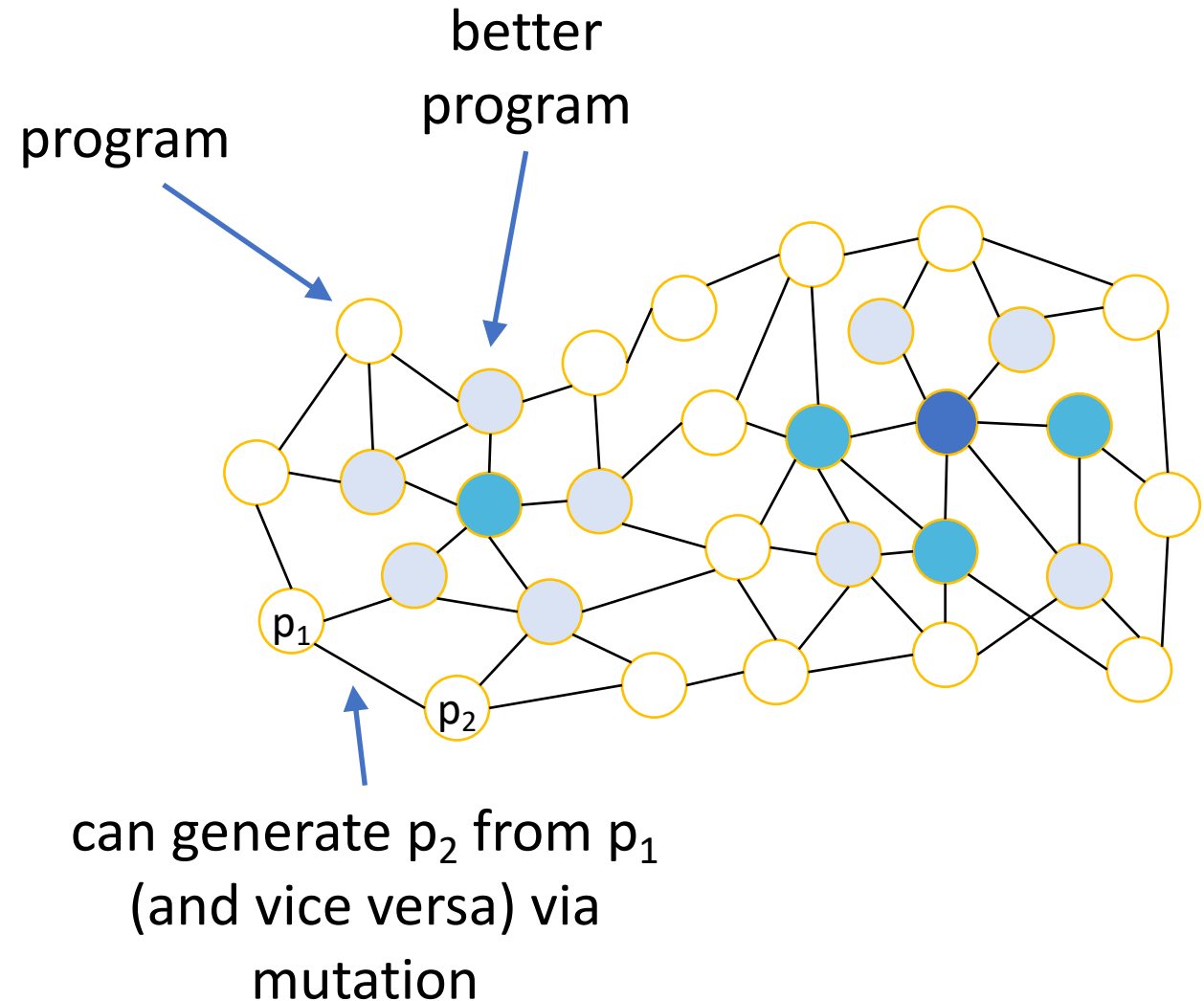


Naïve local search

- To find the best program:

```
p := random()
while (true) {
  p' := mutate(p);
  if (cost(p') < cost(p))
    p := p';
}
```

- Will never get to  from p_1 !



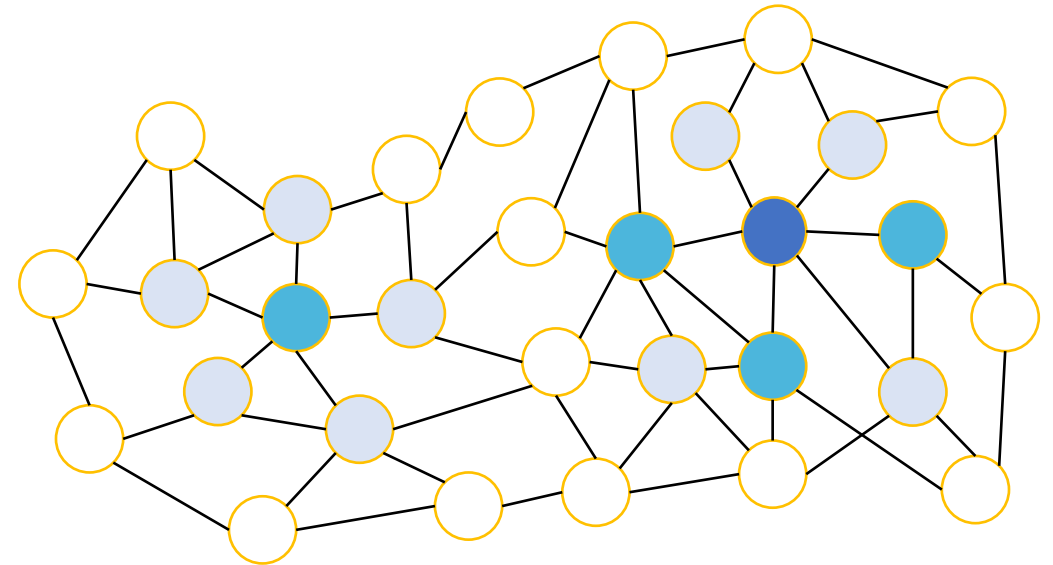
MCMC sampling

- Avoid getting stuck in local minima:

```
p := random()
while (true) {
  p' := mutate(p);
  if (random( $A(p \rightarrow p')$ ))
    p := p';
}
```

- where

- if p' is better than p : $A(p \rightarrow p') = 1$
- otherwise: $A(p \rightarrow p')$ decreases with difference in cost between p' and p



MCMC sampling

- Metropolis algorithm:

$$A(p \rightarrow p') = \min(1, e^{-\beta(C(p') - C(p))})$$

- The theory of Markov chains tells us that in the limit we will be sampling with the probability proportional to

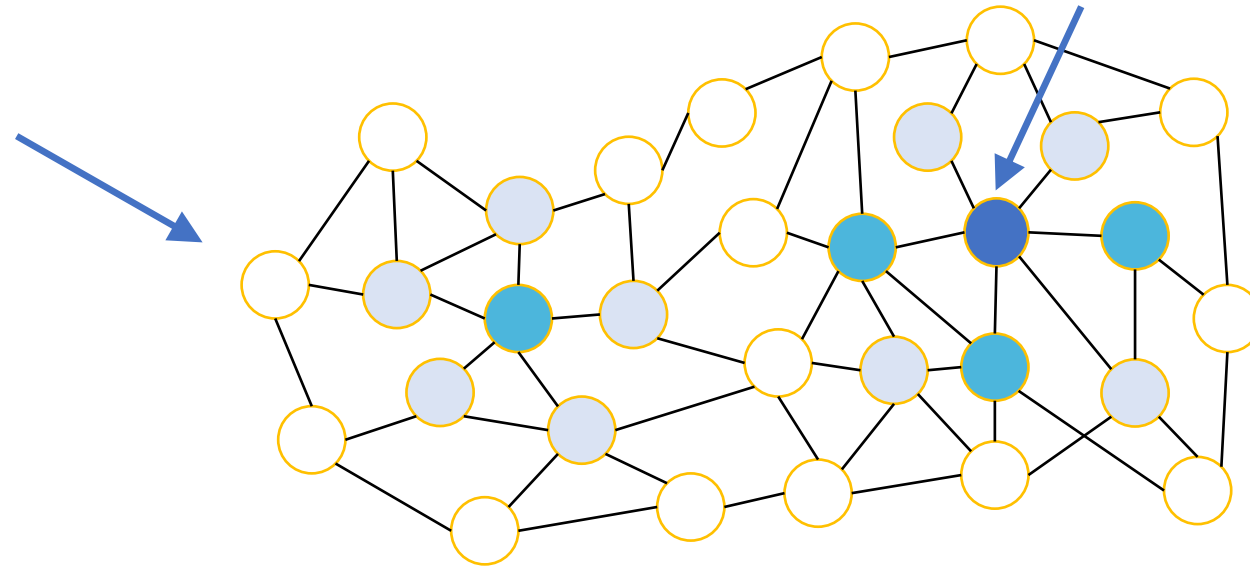
$$e^{-\beta * C(p)}$$

MCMC for superoptimization

[Schkufza, Sharma, Aiken '13]

- .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

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



Cost function

$$C_s(p) = eq_s(p) + perf(p)$$

source program

penalty for wrong results

penalty for being slow

$$eq_s(p) = \sum_{t \in Tests} reg_s(p, t) + mem_s(p, t) + err(p, t)$$

of different bits in registers/memory

of segfaults etc

when $eq_s(p) = 0$, use a symbolic validator

Cost function

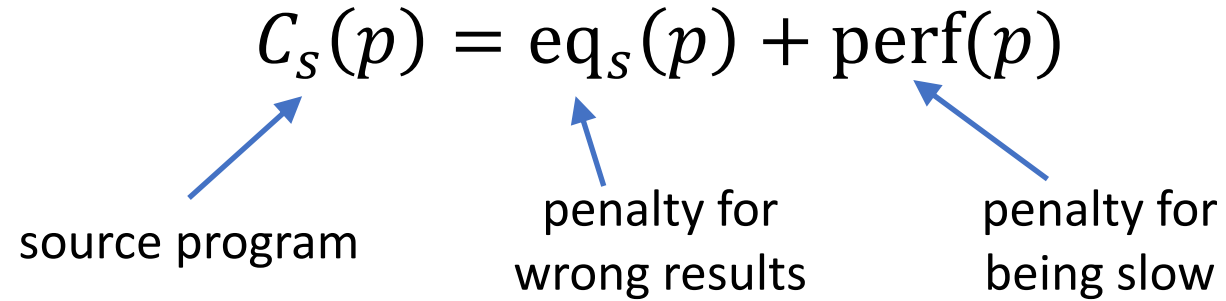
$$C_s(p) = eq_s(p) + perf(p)$$


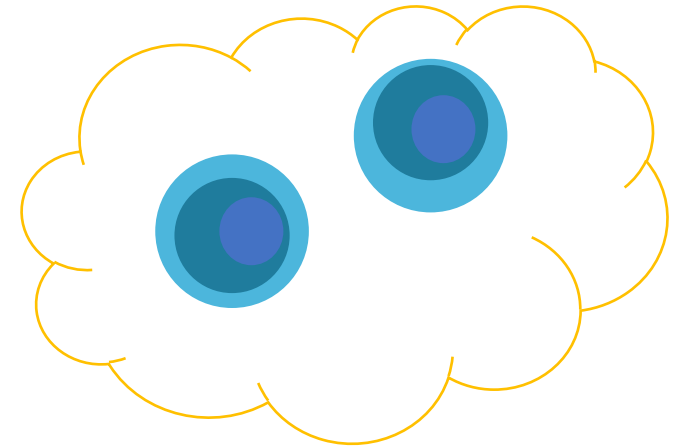
Diagram illustrating the cost function $C_s(p)$ as the sum of two components:

- $C_s(p)$: source program
- $eq_s(p)$: penalty for wrong results
- $perf(p)$: penalty for being slow

$$perf(p) = \sum_{i \in instr(p)} latency(i)$$

Local search: discussion

- Strengths:
 - can explore program spaces with no a-priori bias
- Limitations?
 - only applicable when there is a cost function that faithfully approximates correctness
 - Counterexample: round to next power of two
 - 0011 -> 0100
 - 0101 -> 1000
 - 0111 -> 1000



Stochastic search in synthesis

- Weimer, Nguyen, Le Goues, Forrest. *Automatically Finding Patches Using Genetic Programming*. ICSE'09
- Gissurarson, Applis, Panichella, van Deursen, Sands. *PropR: Property-Based Automatic Program Repair*. ICSE'22
 - Similar but for program repair, uses genetic programming
- Schkufza, Sharma, Aiken: *Stochastic superoptimization*. ASPLOS'13
- Shi, Steinhardt, Liang: *FrAngel: Component-Based Synthesis with Control Structures*. POPL'19
 - Samples from a grammar with bias towards partial solutions