Search Combinators

Tom Schrijvers

UNIVERSITEIT GENT

with Guido Tack, Pieter Wuille, Horst Samulowitz, Peter Stuckey

Search heuristics are crucial.

Support for Search?

General Purpose Programming Language

"everthing is possible, nothing is easy"

Solver-Provided Options

"everything is easy, nothing is possible"

State-of-the-Art Modularity: Prolog

- ✓ label1(Vars1) , label2(Vars2)
- ✓ label1(Vars1) ; label2(Vars2)
- ✓ once(label1(Vars1))
- ✓ once((label1(Vars1),label2(Vars2)))

Lack of Modularity

- ✘ lds(label1(Vars1))

Lack of Modularity

- ✘ lds(label1(Vars1))
- ✘ lds((label1(Vars1) , label2(Vars2)))

Lack of Modularity

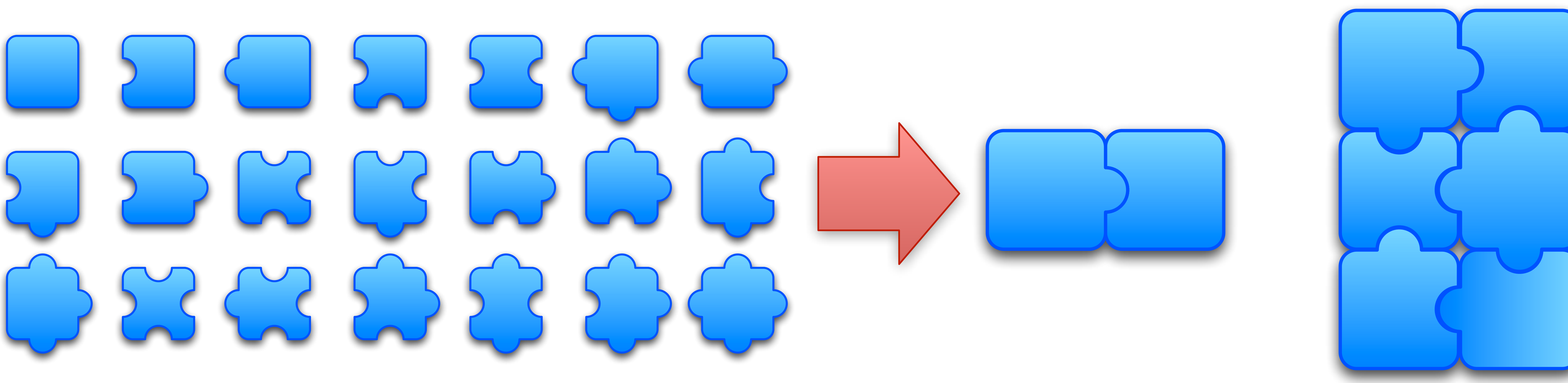
- ✘ lds(label1(Vars1))
- ✘ lds((label1(Vars1) , label2(Vars2)))
- ✓ lds_label1(Vars1)
- ✓ lds_label1_label2(Vars1,Vars2)

Can we do better?

- ✓ Lots of expressivity and flexibility
- ✓ Lots of productivity through high-level specifications
- ✓ Modular reuse of search specifications

Yes: Search Combinators

High-level modular building blocks



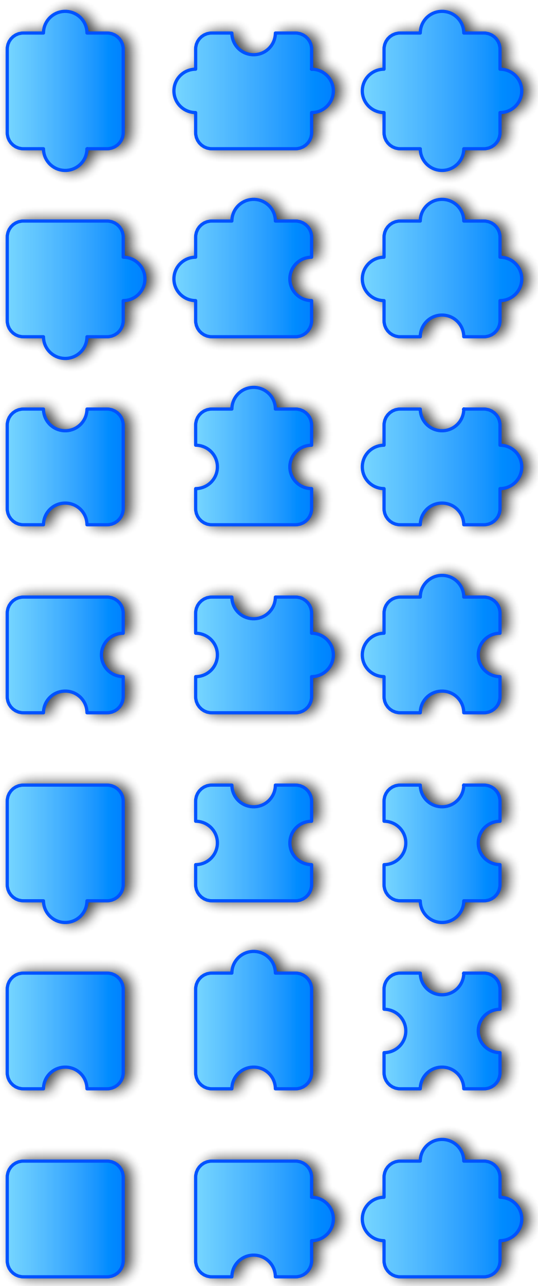
"Everything is possible and easy"

Base Search

```
s ≡
int_search(vars,var_sel,val_sel)
```

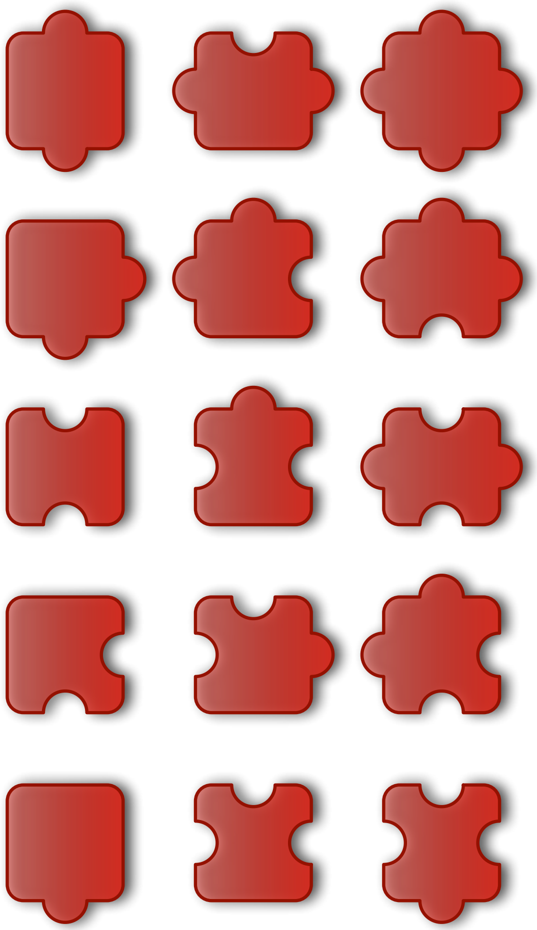
- provided by the solver
- augment with combinators

Combinators



prune
let(v,e,s)
assign(v,e)
post(c,s)
if(c,s_1,s_2)
and([s_1, s_2,..., s_n])
or([s_1, s_2,..., s_n])
portfolio([s_1, s_2,..., s_n])
restart(c,s)

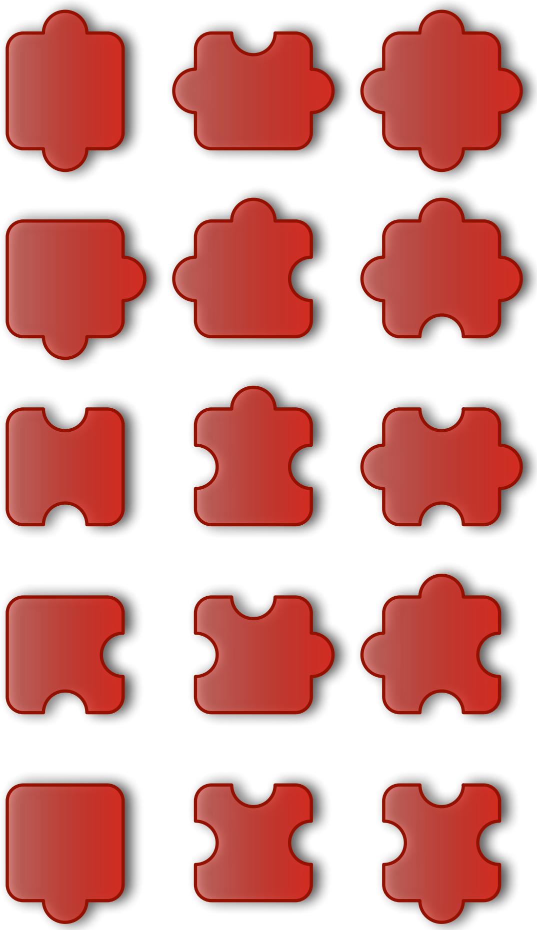
Statistics Combinators



depth(v,s)
discrepancy(v,s)

nodes(v,s)
failures(v,s)
time(v,s)

Statistics Combinators



depth(v,s)

discrepancy(v,s)

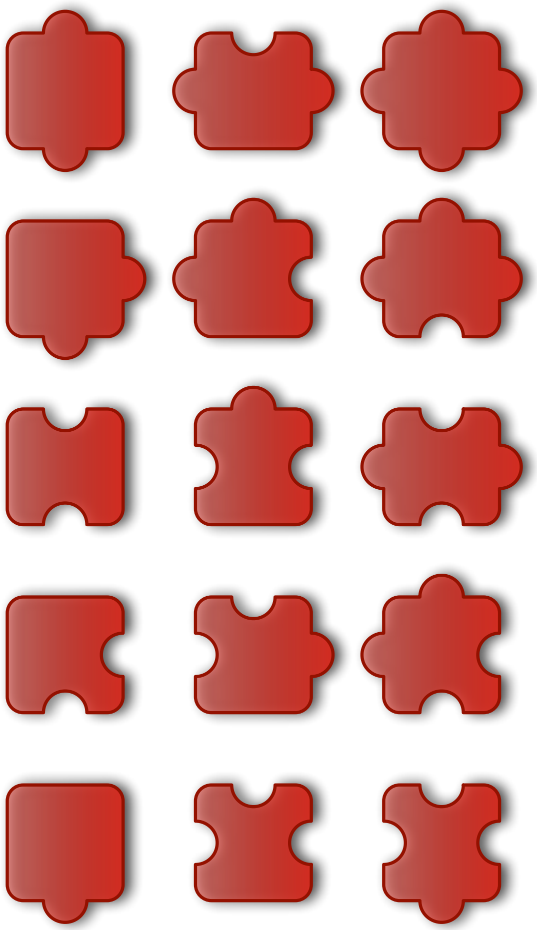
nodes(v,s)

failures(v,s)

time(v,s)

$$\texttt{if}(v < 5, \texttt{depth}(v, s_1), s_2)$$

Statistics Combinators



depth(v,s)
discrepancy(v,s)

nodes(v,s)
failures(v,s)
time(v,s)

if(depth < 5,s_1,s_2)

Reusable Abstractions

```
limit(c,s) ≡ if(c,s,prune)

for(v,l,u,s) ≡ ...

lds(s) ≡
  for(n,0,∞,
    limit(discrepancy ≤ n,s)
  )
```

Reuse Examples

```
✓ lds(int_search(vars,...))

✓ lds(and([int_search(vars1,...)
          ,int_search(vars2,...)])

✓ lds(or([int_search(vars1,...)
         ,int_search(vars2,...)])
  ...
```

More Abstractions

`bab`(*obj*,*s*)

`restart_bab`(*obj*,*s*)

`dicho`(*obj*,*s*,*lb*,*ub*)

`id`(*s*)

`hot_start`(*c*,s_1,s_2)

see paper

Radiotherapy Planning

```
bab(k,
  and([int_search(N,...)]
       ++
      [once(int_search(rowᵢ,...))
      | i in 1..n
      ]
     )
   )
```

Modular Syntax

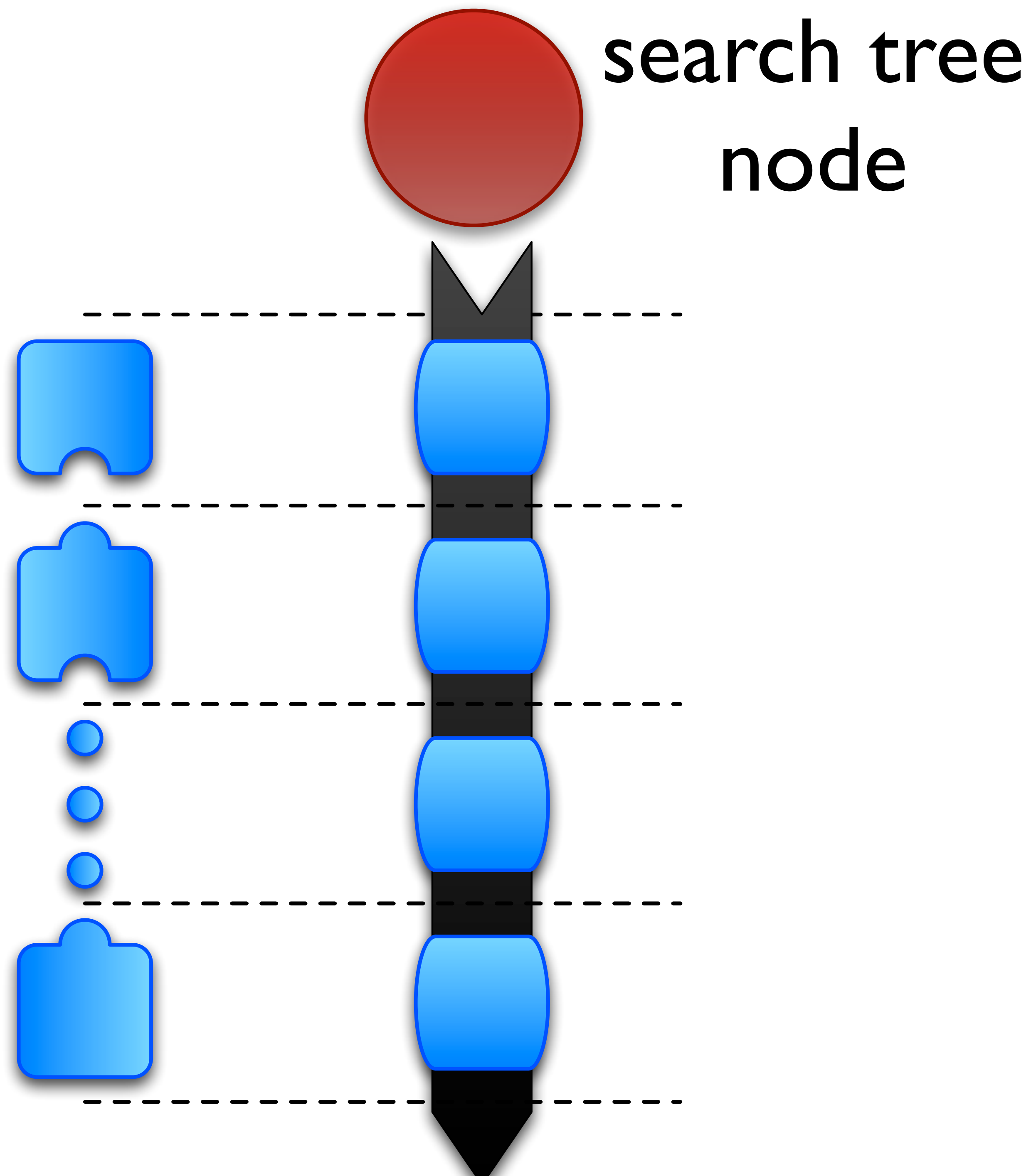
vs.

Modular Semantics

Modular Semantics

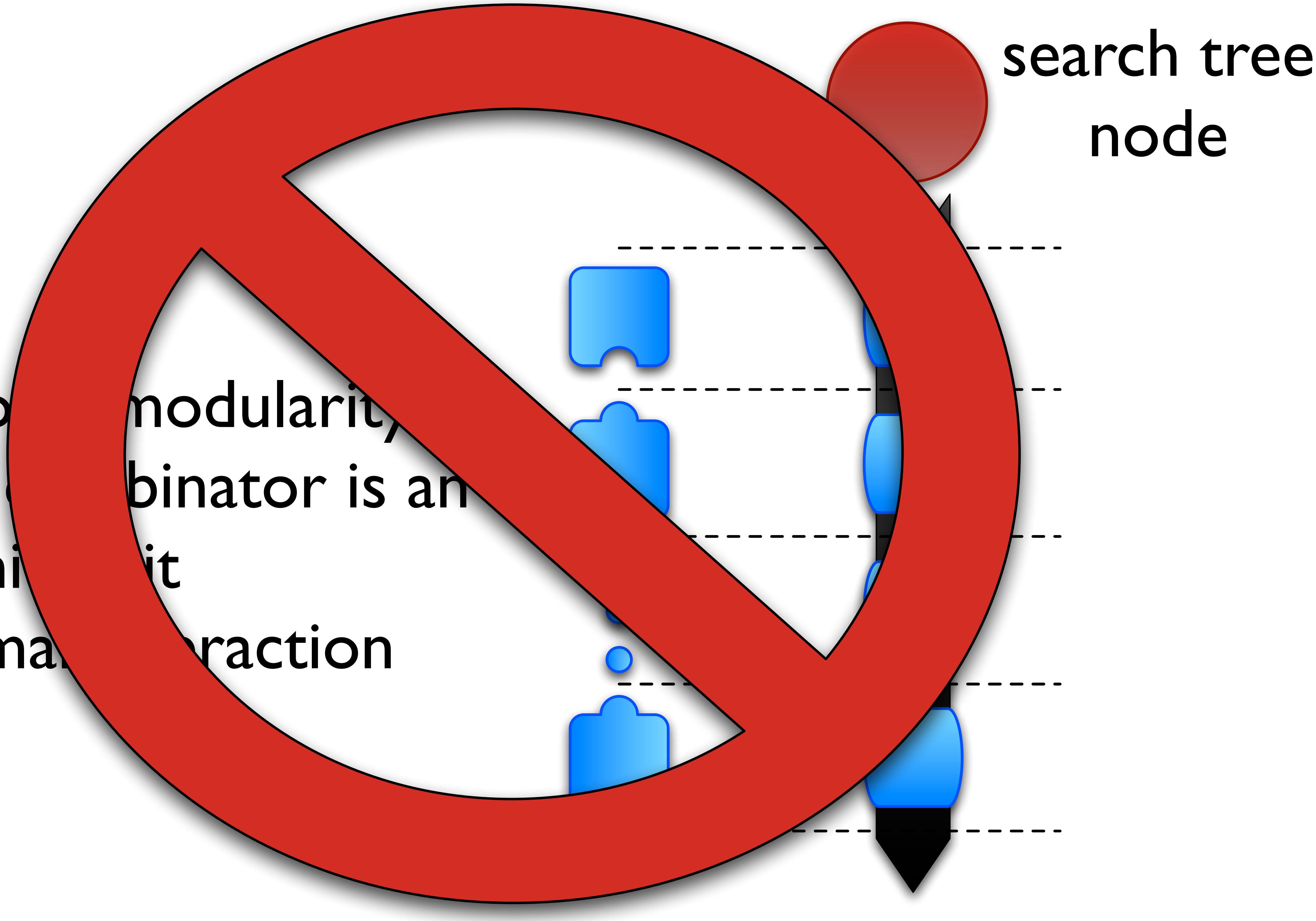
traditional modularity:

- each combinator is an atomic unit
- minimal interaction



Modular Semantics

traditio modularit
- each binator is an
  atomi it
- minima raction



Modular Semantics

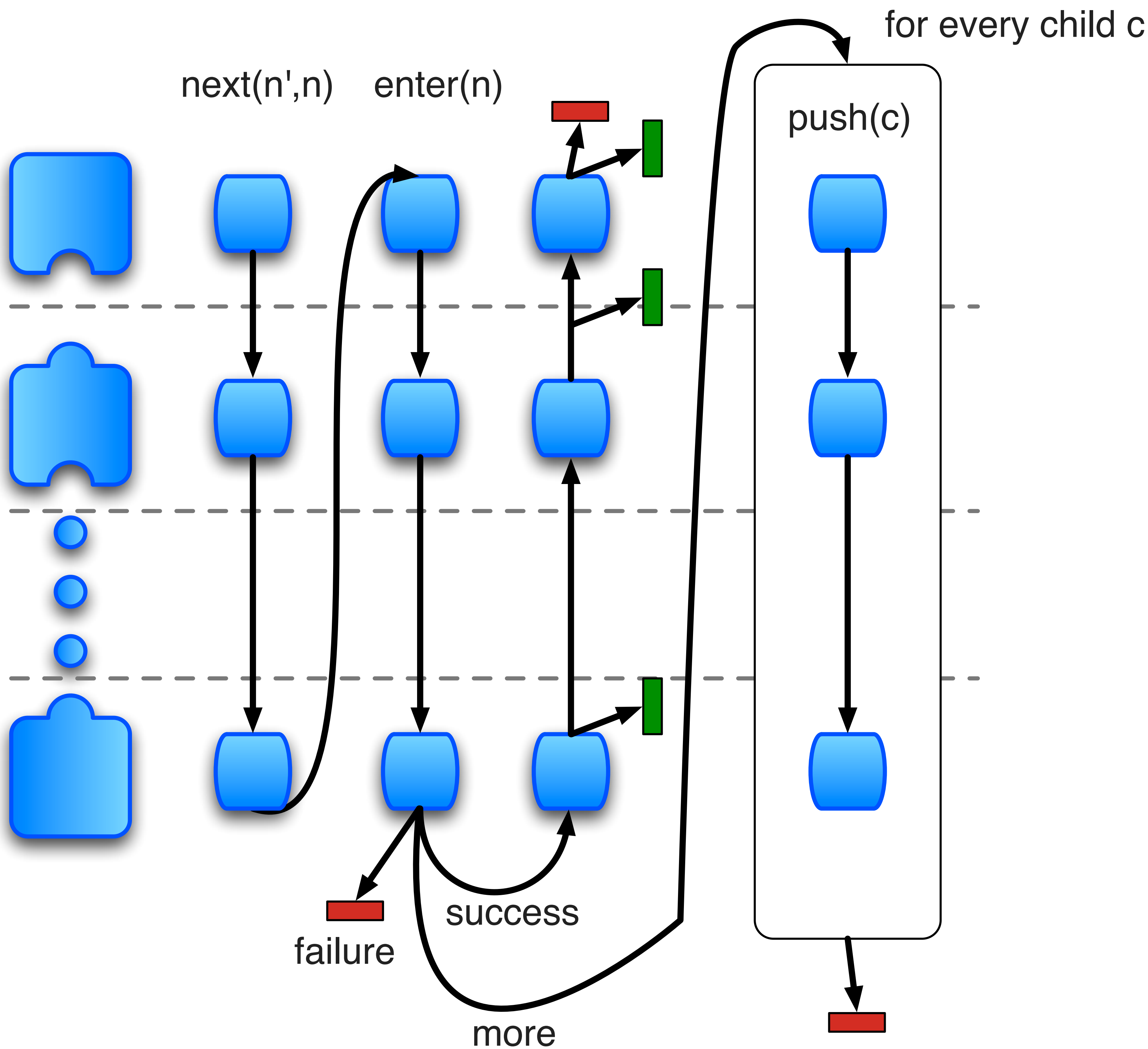
- cross-cutting behavior
- highly entangled in monolithic code

➡ "Aspect-Oriented Programming"

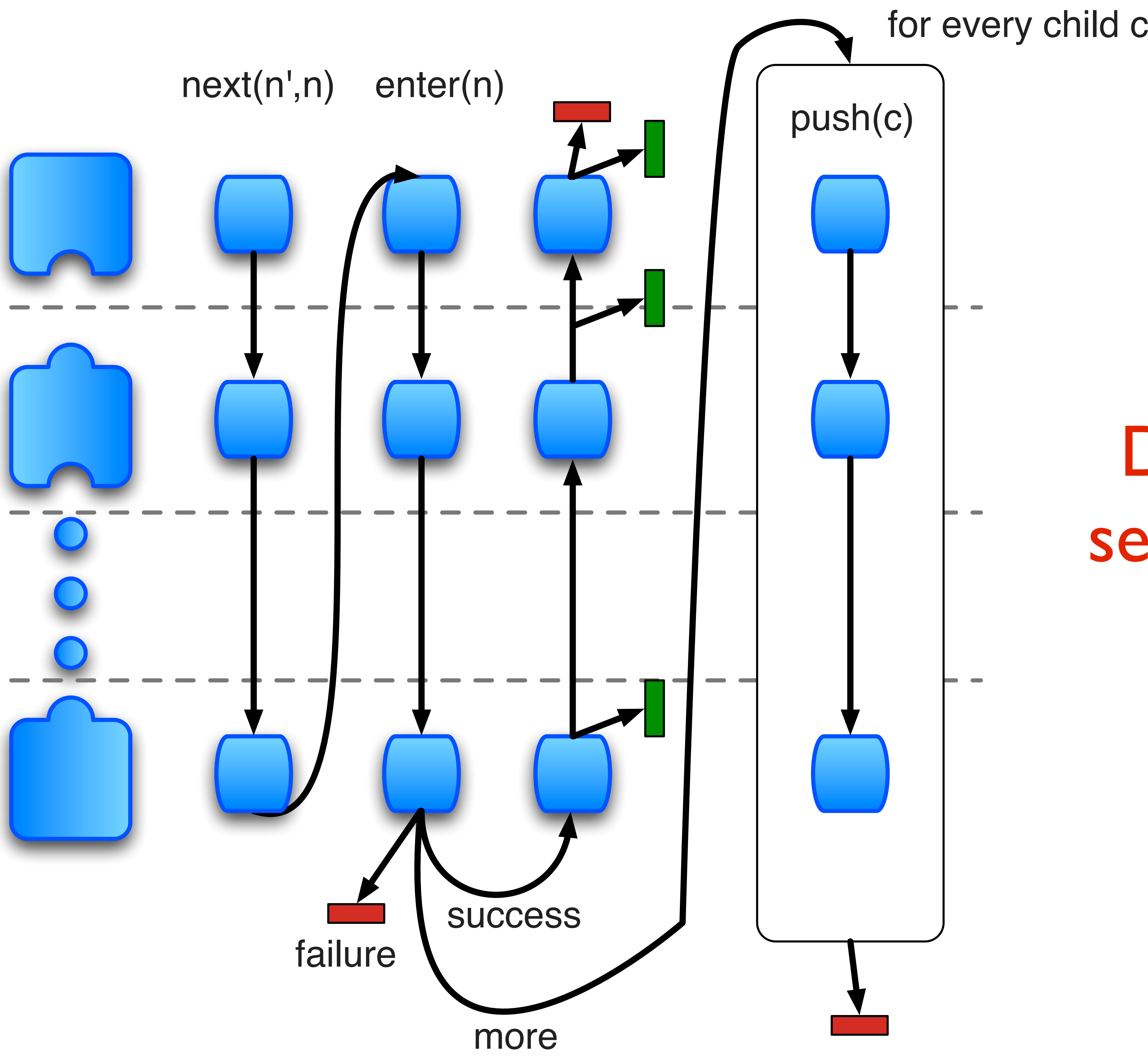
Functional Mixin Inheritance

- disciplined form of AOP
  - ★ meaningful and pre-defined set of interaction points
- no AOP system needed

Mixin-based Interaction



Mixin-based Interaction



Details: see paper

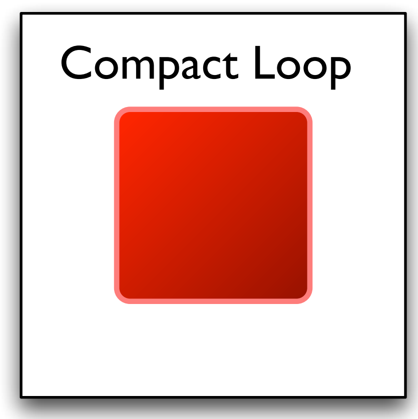
Implementations

DSL | **Haskell** | **C++**

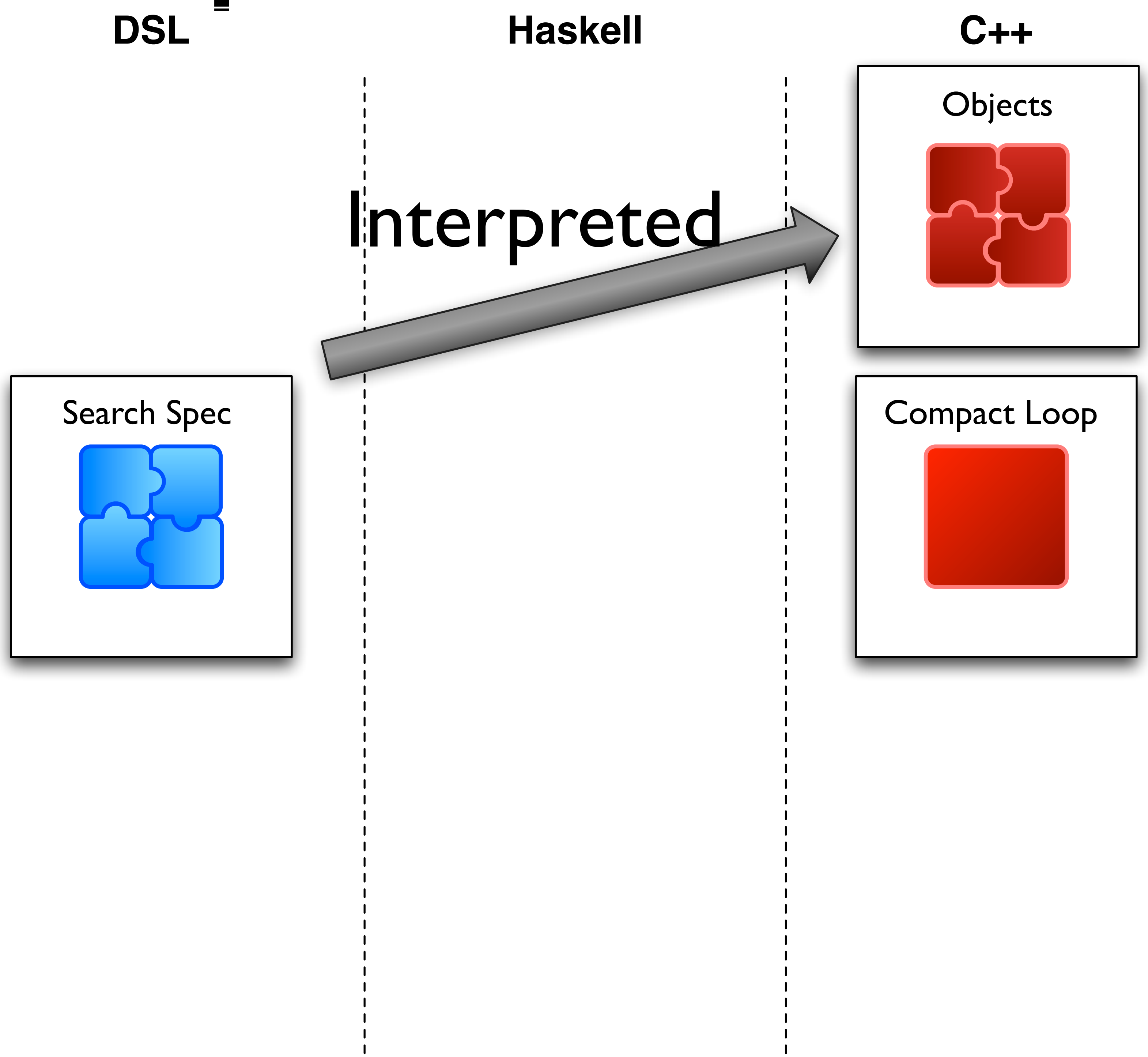
Compact Loop

Implementations

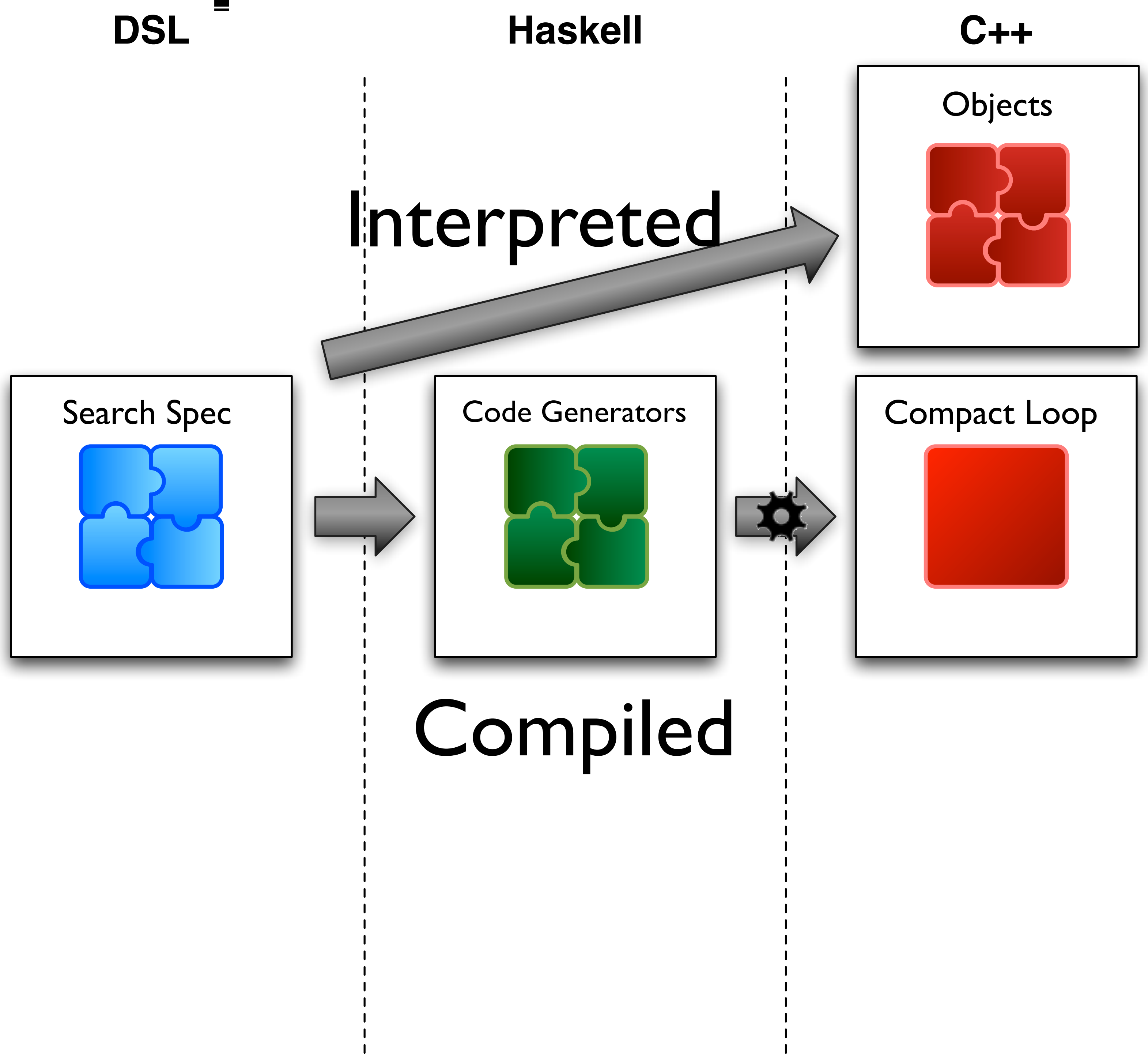
DSL | **Haskell** | **C++**



Implementations

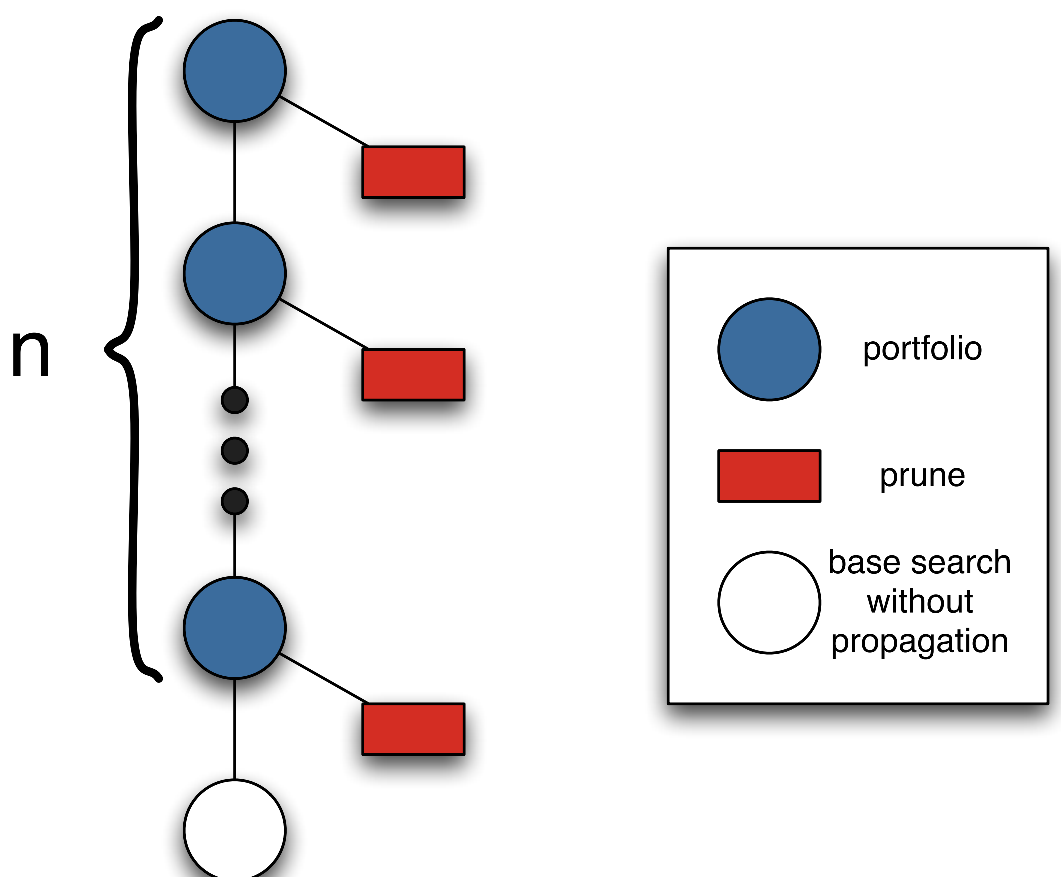


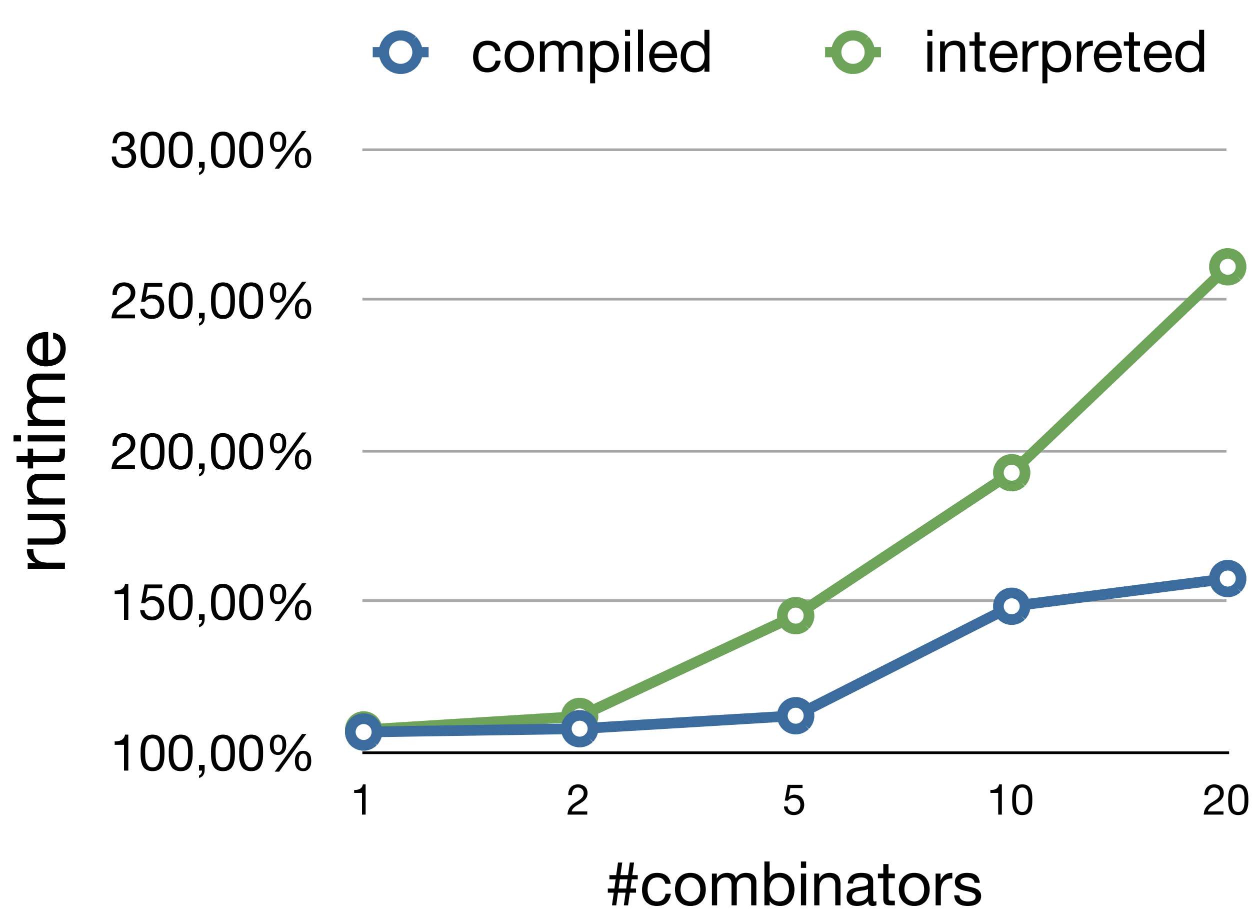
Implementations



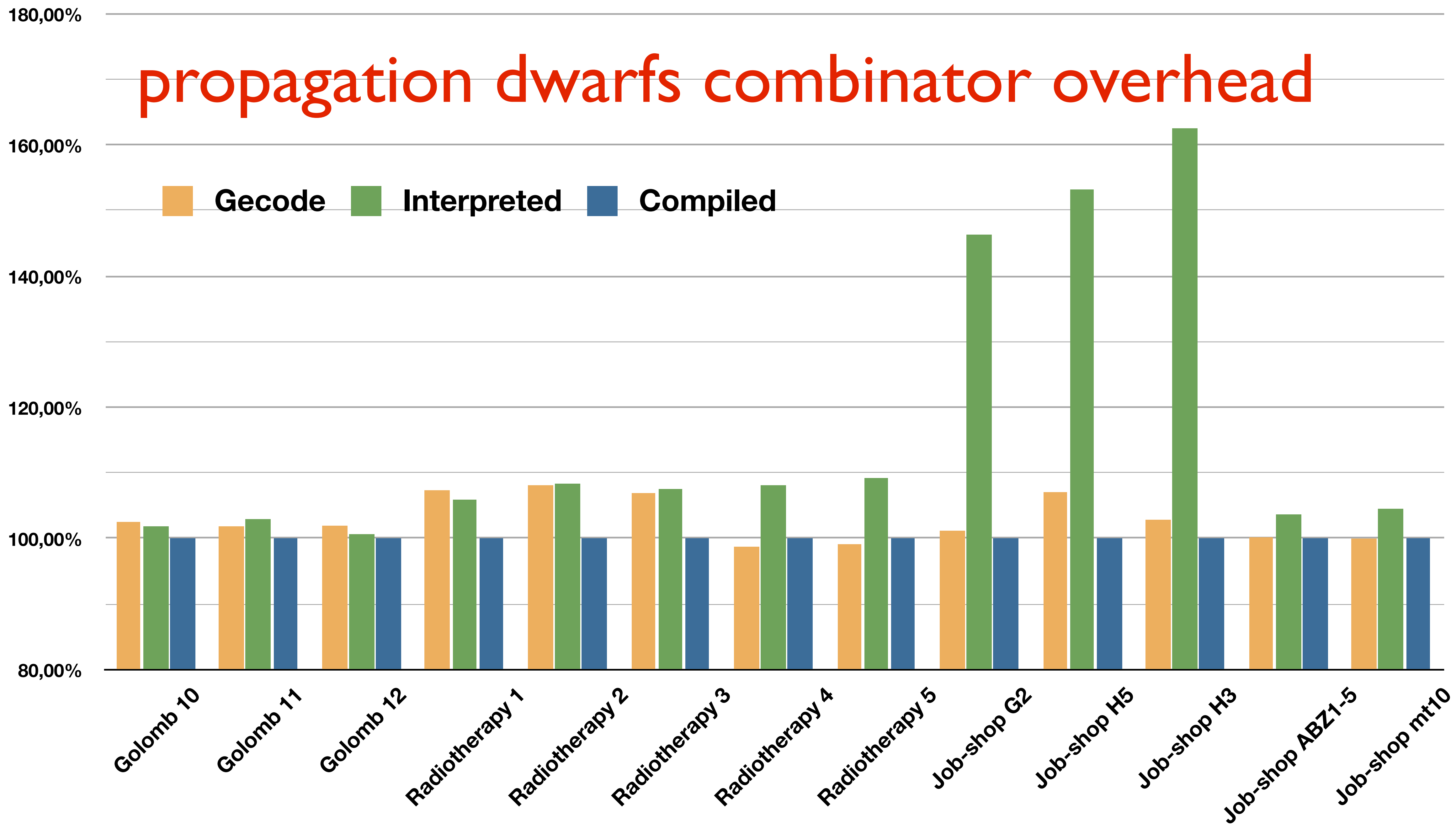
Combinator Overhead?

Worst-case Scenario





In Practice


180,00%
propagation dwarfs combinator overhead
160,00%
Gecode
Interpreted
Compiled
140,00%
120,00%
100,00%
80,00%
Golomb 10
Golomb 11
Golomb 12
Radiotherapy 1
Radiotherapy 2
Radiotherapy 3
Radiotherapy 4
Radiotherapy 5
Job-shop G2
Job-shop H5
Job-shop H3
Job-shop ABZ1-5
Job-shop mt10

Summary

- high-level modular modeling of search
- low-level modular implementation
- competitive performance compared to hand-coded algorithm

Future Work

- Optimizations
- MiniZinc integration
- Combinators for parallel search
- Extend other solving technology (e.g., LP)

➡ Combinators for hybrid search

Thank You!

Combinator Overhead