LEVERAGING DSLS (ALMOST) FOR FREE

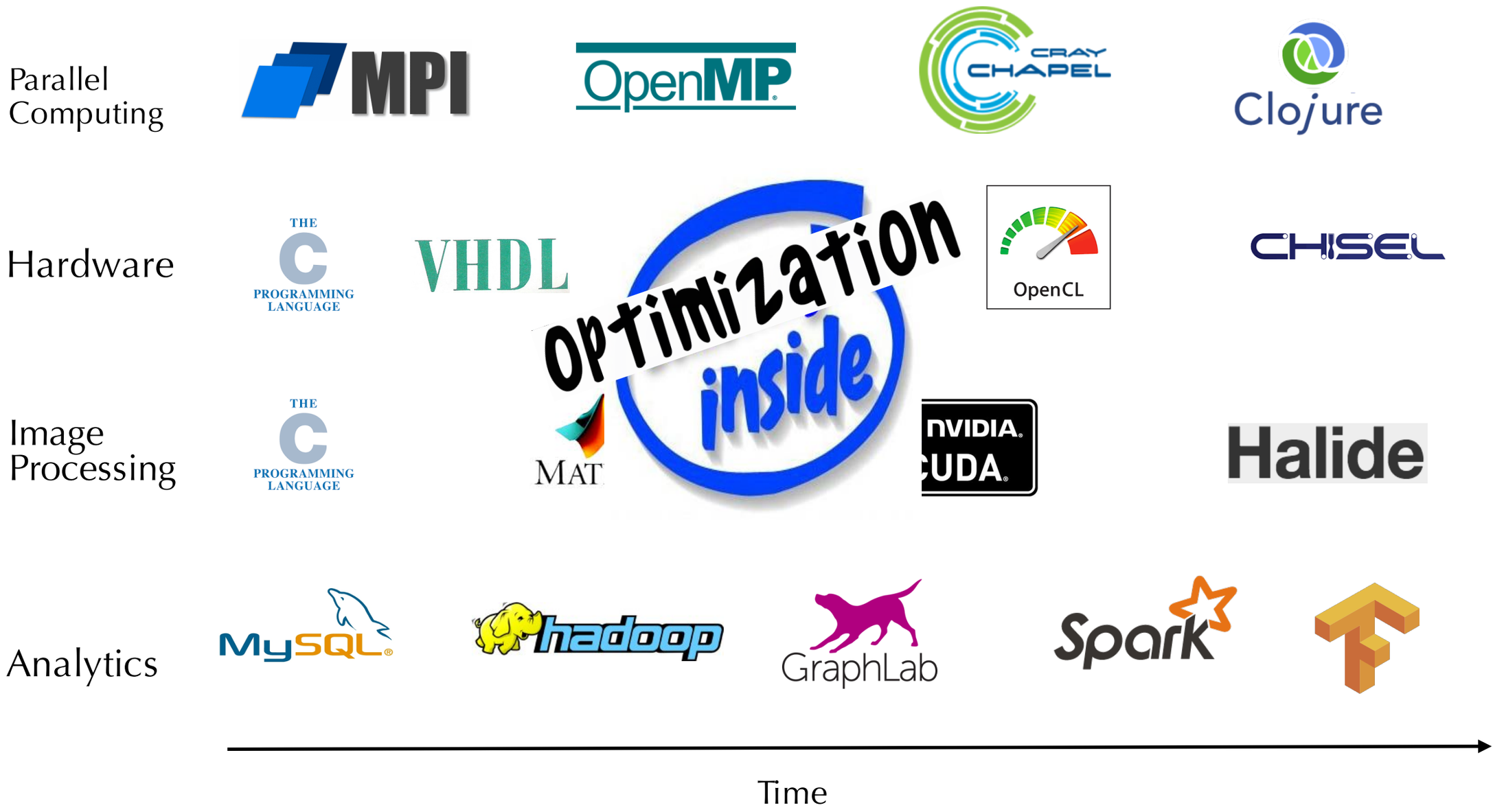
@alvinkcheung

uwdb.io

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

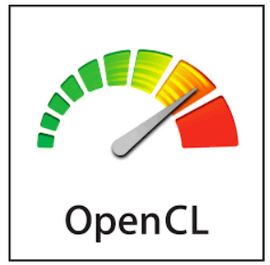
PLSE

uwplse.org


Parallel Computing
MPI
OpenMP
CRAY CHAPEL
Clojure
Hardware
THE C PROGRAMMING LANGUAGE
VHDL
Optimization inside
OpenCL
CHISEL
Image Processing
THE C PROGRAMMING LANGUAGE
MAT
NVIDIA CUDA
Halide
Analytics
MySQL
hadoop
GraphLab
Spark
Time


Java


THE
C
PROGRAMMING
LANGUAGE


OpenCL


OpenMP


python

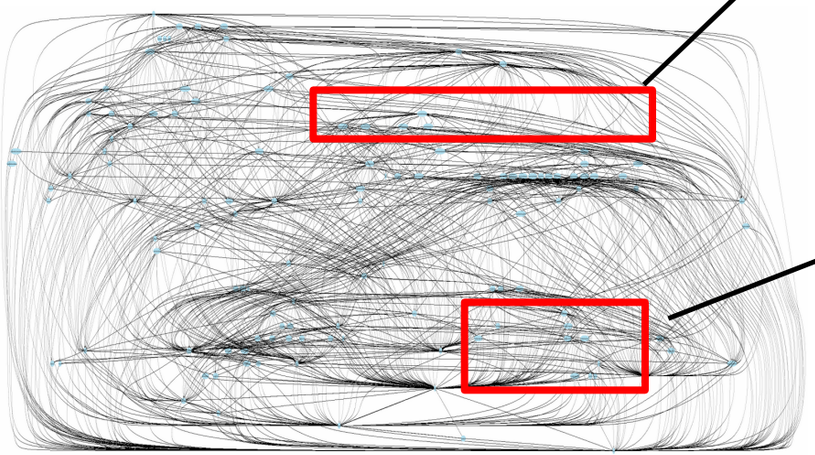
?


NVIDIA
CUDA


COBOL


fortran


Spark



Rotate → Blur ✓

Hash Partitioning → Join ✓

Caching ✗
Pipelining ✗

NVIDIA CUDA

Hash implementation ✗

hadoop

Embedded algorithmic details ✓

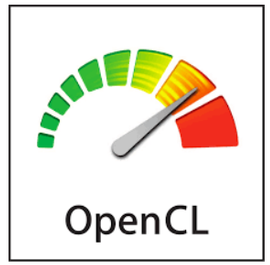
✗ Architecture-specific optimizations

THE C PROGRAMMING LANGUAGE

COBOL

fortran

?



OpenMP

Your DSL here

Spark

Annuals of Code Rewrites

- Overhaul of the Mozilla Gecko layout engine (2 years), and now to Servo
- Rewrite of INGRES database into PostGRES (3 years)
- Twitter search engine rewrite (2 years)

All of these were major engineering efforts!

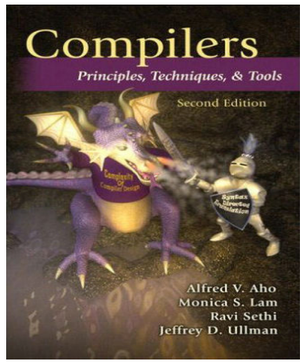
Annuals of Code Rewrites

- Overhaul of the Mozilla Gecko layout engine (2 years), and now
- Rewrite
- Twitter s

Vision:
Perform code rewrites automatically

Current focus:
Performance-critical code fragments

All of these were major engineering efforts!



Syntax-Driven Compilation

```
for ($i = 0; $i < $l1.size(); ++$i) {
    $l2.append($l1[$i] + $c);
}
```



```
$l2 = $l1.map(e -> e + $c);
```

```
List getUsersWithRoles () {
  List users = this.userDao.getUsers();
  List roles = this.roleDao.getRoles();
  List results = new ArrayList();
  for (User u : users) {
     for (Role r : roles) {
       if (u.roleId == r.id)
          results.add(u);
     }}
  return results; }
```


?

```
for (int i = 0; i < docs.length; i++) {
  String[] split = docs[i].split(" ");
  for (int j = 0; j < split.length; j++) {
    String word = split[j];
    Integer prev = m.get(word);
    if (prev == null) prev = 0;
    m.put(word, prev + 1);
  }
}
```

[Radoi et al, OOPSLA 14]

⇓ E:localize-fold-accsses

⇓ E:map-horizontal-fission

⇓ S:eliminate-null-check

⇓ E:map-vertical-fission

⇓ E:identify-map-monoid-plus

⇓ E:swap-map-with-fold

⇓ S:reduce-monoid-identity-op

⇓ E:pull-map-from-fold-by-subexpression-extract

⇓ E:localize-group-by-accesses

$$\mathcal{L}(x = E \prec R) \to \texttt{let}\ x = \mathcal{L}(E)\ \texttt{in}\ \mathcal{L}(R)$$

$$\mathcal{L}(a[x := y]) \to a[x := y]$$

$$\mathcal{L}(\texttt{return}\ x) \to x$$

$$\mathcal{L}(\text{branch instruction} \prec R) \to \mathcal{L}(R)\ \text{(handled when reaching its } \phi$$

$$\mathcal{L}\left(\begin{array}{l}\texttt{for}\ \ i = \phi(i', i''),\\ \quad\quad r_1 = \phi(r'_1, r''_1), \ldots, r_n = \phi(r'_n, r''_n)\\ \quad\quad i < l\\ \{\ E\ \} \prec R\end{array}\right) \to \mathcal{L}\left(\begin{array}{l}\texttt{let}\ f = \lambda\, r_1\ r_2 \ldots i\,.\,\mathcal{L}(E \prec\\ \quad \texttt{let}\ r = \texttt{fold}\langle r'_1, \ldots, r'_n\rangle\ j\\ \quad\quad \texttt{let}\ r_1, \ldots, r_n = r\ \texttt{in}\ \mathcal{L}(\end{array}\right.$$

$$\mathcal{L}\left(\begin{array}{ll}x = \phi(x_0, x_1) & \\ \text{generated by the } \texttt{if} & \prec R\\ \text{with branch condition } C & \end{array}\right) \to \texttt{let}\ x = \text{if } C \text{ then } x_0 \text{ else } x_1\ \texttt{in}\ \mathcal{L}(1$$

$$\mathcal{L}(\ldots) \to \ldots$$

(localizc-map-acccsscs)

$$\frac{\texttt{map}\,\lambda\, K V\,.\,E}{\lambda\, a\,.((\texttt{map}\,\lambda\, K\ V v\,.\,E[v/c[i]])\ \circ(\texttt{zip}\, a\ c))}$$

$c \in \{c \subset E \mid$
$(\exists! i \in K.c[i] \subset E) \wedge$
$(\text{free}(c) \setminus \text{free}(E) = \emptyset) \wedge$
$(\nexists v \in K \cup V\,.\,v \in c)\}$

(localize-group-by-accesses)

$$\frac{\texttt{groupBy}(\lambda\, k\,.\,E)\ D}{\texttt{groupBy}(\lambda\, kv\,.\,E[v/c[k]])c}$$

D is the domain (set of keys) of the c Map

Brittle
Difficult to be correct
Hard to maintain

$$\frac{}{a[k]}$$

a is a monoid with 0 as identity

(identify map monoid plus)

$$\frac{\begin{array}{c}\texttt{map}(\lambda\, i\, x\, y\,.\,x \oplus y)\\ (\texttt{zip}\ A\ B)\end{array}}{A \boxplus B}$$

A and B are $\texttt{M}[T]$ monoids
$\oplus$ is the plus for T
$\boxplus$ is the plus for $\texttt{M}[T]$

(swap map and fold)

$$\frac{(\texttt{fold}\, r_0\ \oplus) \circ (\texttt{map}\, f)}{\lambda\, c\,.(r_0 \oplus f(\texttt{fold}\ 0_{\boxplus}\ \boxplus)\ c))}$$

$\texttt{map}$ over monoid $\boxplus, 0_{\boxplus}$
$\forall ab.f(a \boxplus b) = f(a) \oplus f(b)$

(flatMap)

$$\frac{(\texttt{fold}\, r_0\ \oplus) \circ (\texttt{map}\, f)}{r_0 \oplus \text{flatMap}\, f}$$

$\texttt{fold}$ over the monoid $\oplus, 0_{\oplus}$

(extract map from fold)

$$\frac{\texttt{fold}\langle r_0^0, \ldots, r_n^0\rangle\ \lambda\langle r_0, \ldots, r_n\rangle\ K\ V\,.\,E}{(\texttt{fold}\langle r_0^0, \ldots, r_n^0\rangle\ \lambda\langle r_0, \ldots, r_n\rangle\ K\ \langle v_0^f, \ldots, v_m^f\rangle V_{\cap\,\text{free}(F)}\,.\,F)\ \circ\ (\texttt{map}\,\lambda\, K\ V\,.\langle G[r_\_^0/r\_], V_{\cap\,\text{free}(F)}\rangle)}$$

$E = (\lambda\langle v_0^f, \ldots, v_m^f\rangle\,.\,F) \circ G$

F is $\arg\max \mathcal{C}(G)$ with the condition:

$\nexists i \in [0..n]\ .r_i \in G \wedge r_i \in E[r_\_^0/r\_]$ whe

$r_\_^0/r\_ = r_i^0[k]/r_i[k]$ applied for all $i \in$

(fold to group by)

$$\frac{\texttt{fold}\, r_0\, \lambda\, r\, V\,.\, r[E := B]}{(\texttt{map}\,\lambda\, k\, l\,.(\texttt{fold}\, r_0[k]\,\lambda\, g\, V\,.\, C)\ l)\ \circ\ (\texttt{groupBy}\,\lambda\, V\,.\, E)}$$

$C = B[g/r[E]]$

$r \notin C \wedge r \notin E \wedge \exists\, v \in V.v \in E$

we cannot prove E is distinct across the fol

SEARCH

Target code Proof of translation

SEARCH

Target code ∧ DSL

Proof of translation

SEARCH

PROGRAM SYNTHESIS

Target code ∧ DSL

Proof of translation

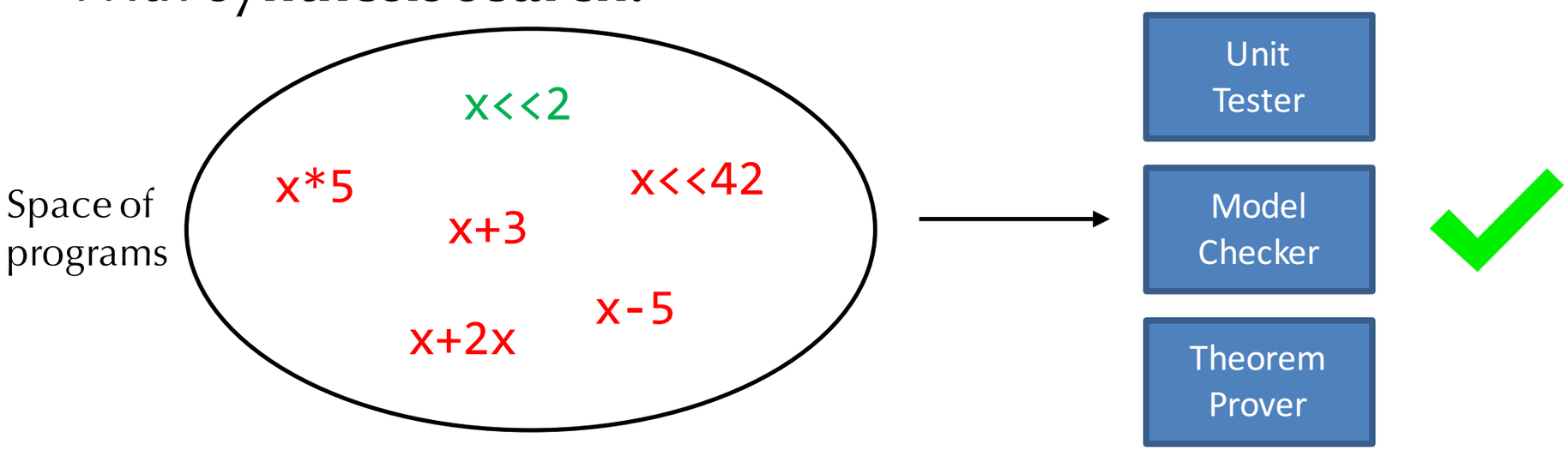
Program Compilation vs. Synthesis

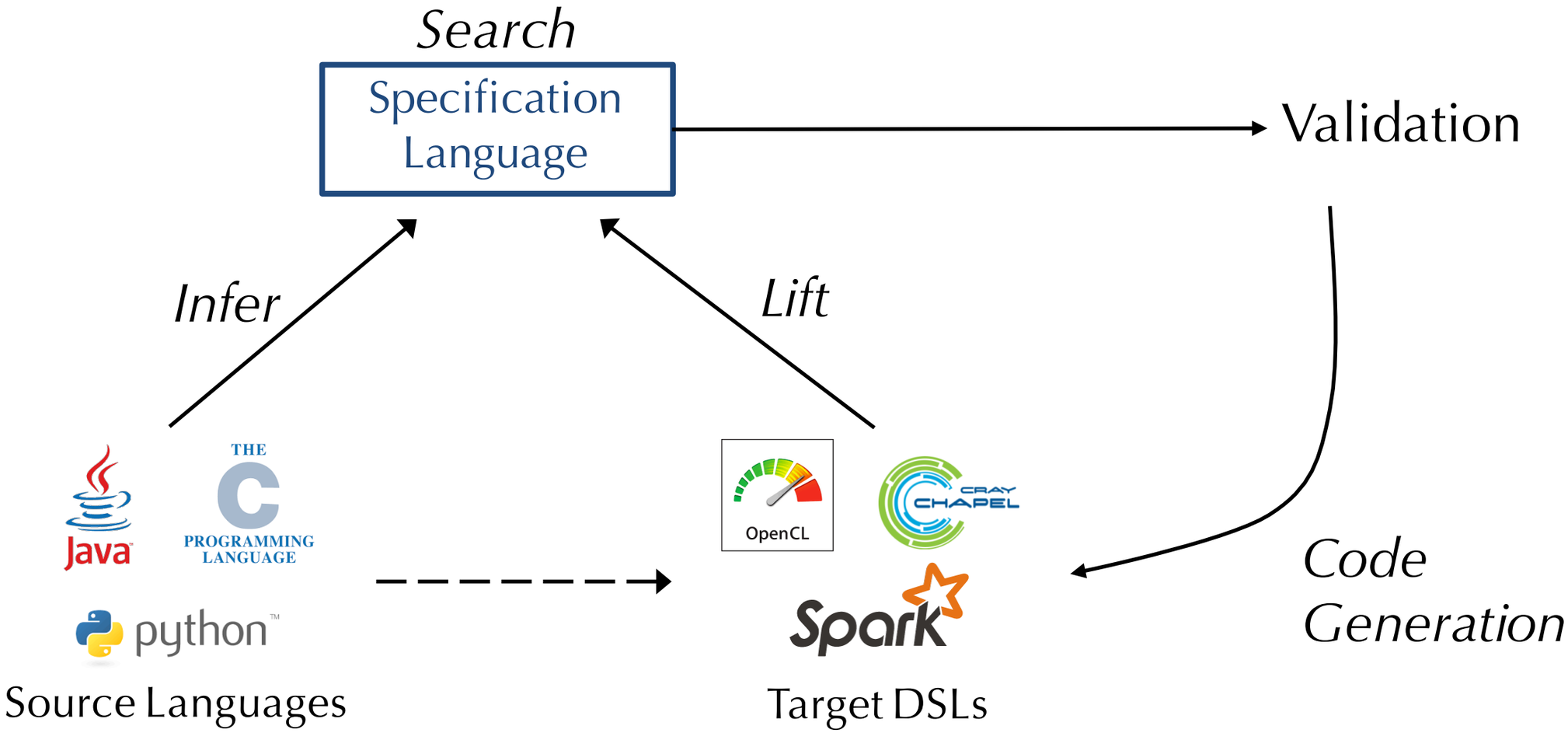
Suppose we want to optimize **x+x+x+x**

With compiler **rewrite rules**:

x+x+x+x ⟶ 4*x ⟶ 2^2*x ⟶ x<<2 ✓

With **synthesis search:**





Verified Lifting

MetaLift

Spec Language

- Boolean
- Arithmetic

- Classes
- Lists
- Arrays
- …

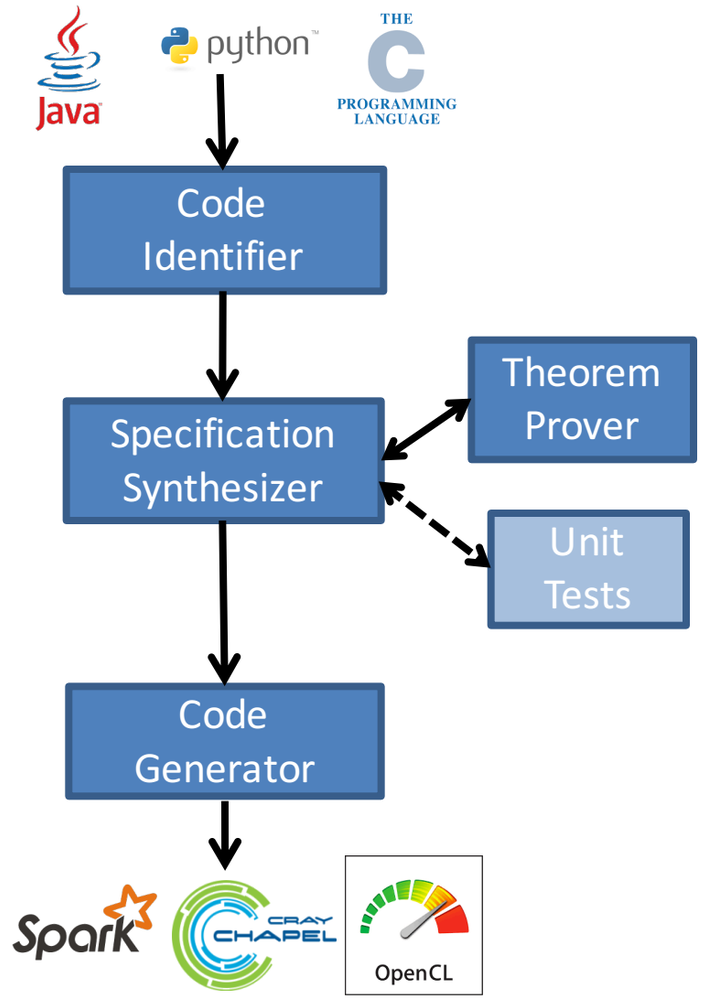
Compiler Generator

Target code fragments

DSL semantics

Search space description

Codegen rules



MetaLift

Spec Language

- Boolean
- Arithmetic

- Classes
- Lists
- Arrays
- …

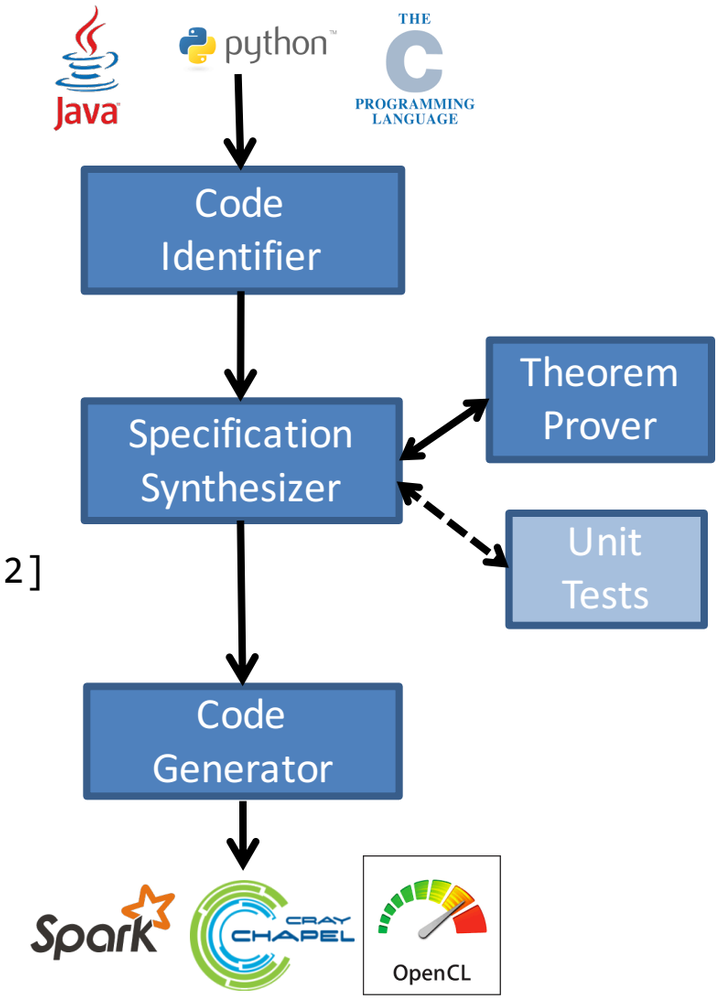
Compiler Generator

Target code fragments

```
select :: [(a)] -> ((a)-> Bool) -> [(a)]
select l f = [t | t <- l, f(t)]

join :: [(a)] -> [(b)] -> [(a,b)]
join l1 l2 = [(t1, t2) | t <- l1, t <- l2]
...
```

Codegen rules



MetaLift

Spec Language

- Boolean
- Arithmetic

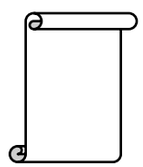
- Classes
- Lists
- Arrays
- …

Compiler Generator

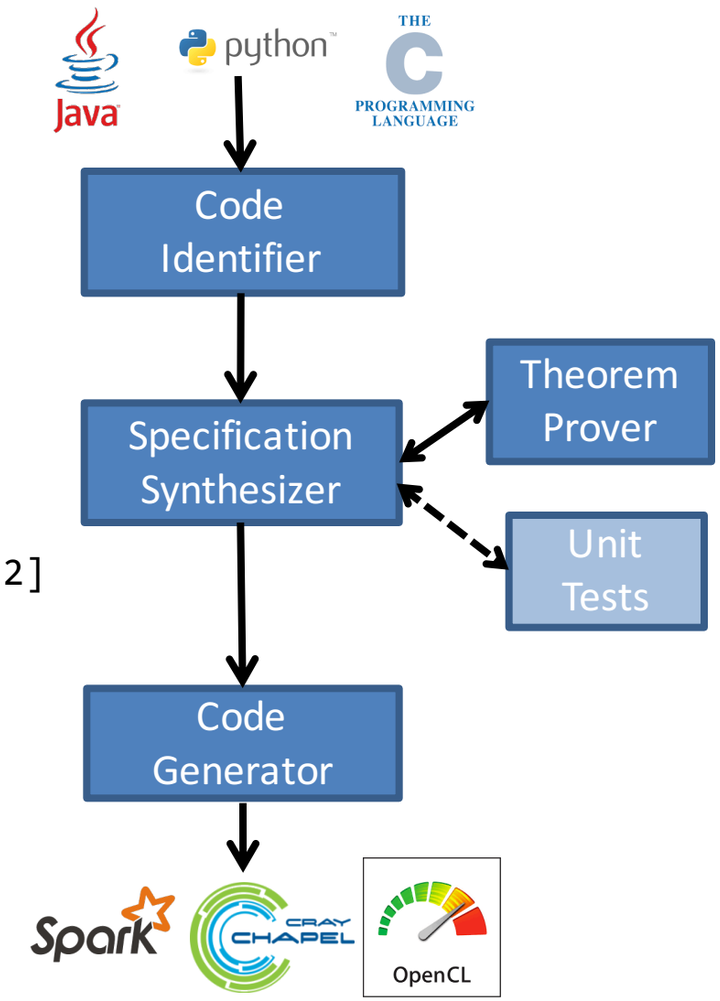
```
Java: while (*) { * }

select :: [(a)] -> ((a)-> Bool) -> [(a)]
select l f = [t | t <- l, f(t)]

join :: [(a)] -> [(b)] -> [(a,b)]
join l1 l2 = [(t1, t2) | t <- l1, t <- l2]
...
```



Codegen rules



MetaLift

Spec Language

- Boolean
- Arithmetic

- Classes
- Lists
- Arrays
- …

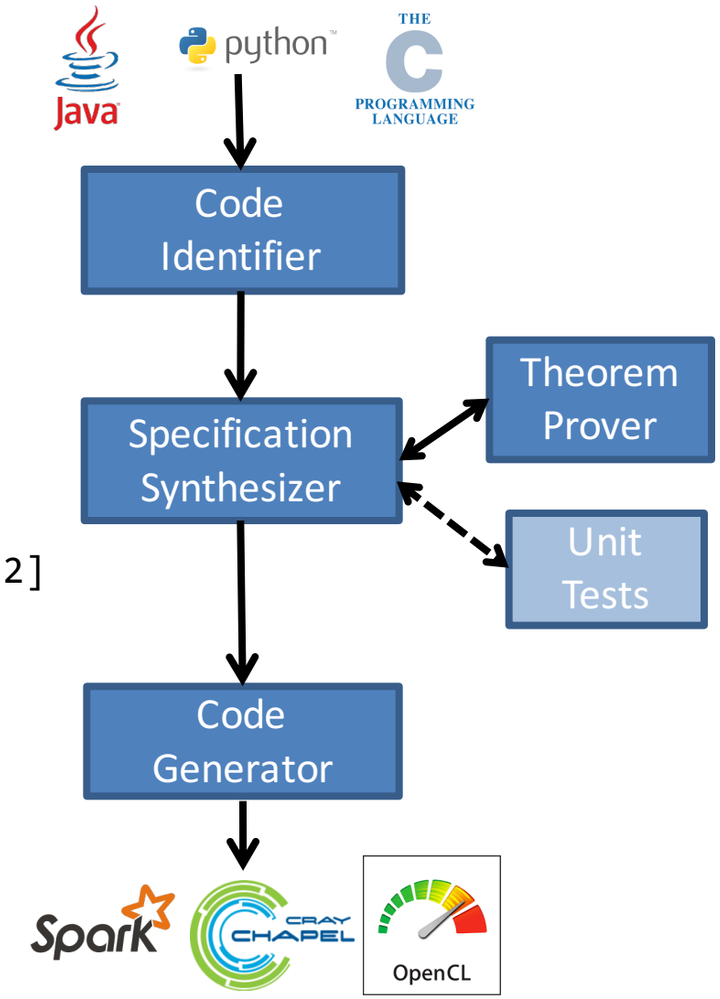
Compiler Generator

```
Java: while (*) { * }
```

```
select :: [(a)] -> ((a)-> Bool) -> [(a)]
select l f = [t | t <- l, f(t)]

join :: [(a)] -> [(b)] -> [(a,b)]
join l1 l2 = [(t1, t2) | t <- l1, t <- l2]
...

    translate (select l f) =
     "select * from" + translate(l) +
     "where" + translate(f)
     ...
```



Data Analytics: Java → SQL

[PLDI 13, CIDR 14]

1. Define ordered lists and their operations

```
select :: [(a)] -> ((a)-> Bool) -> [(a)]
select l f = [t | t <- l, f(t)]

join :: [(a)] -> [(b)] -> [(a,b)]
join l1 l2 = [(t1, t2) | t <- l1, t <- l2]
```

2. Synthesizer infers spec from source

```
List getUsersWithRoles () {
  List users = this.userDao.getUsers();
  List roles = this.roleDao.getRoles();
  List results = new ArrayList();
  for (User u : users) {
     for (Role r : roles) {
       if (u.roleId == r.id)
          results.add(u);
     }}
  return results; }
```

3. Retarget synthesized spec to SQL

codegen

```
List getUsersWithRoles () {
  return executeQuery(
   "SELECT    u FROM users u, roles r
    WHERE     u.roleId == r.id
    ORDER BY u.roleId, r.id");
}
```

Lifted code can be optimized by DBs 100-1000x speedup

Leveraging GPUs: Fortran → Halide

[SNAPL 15, PLDI 16]

1. Define arrays and their operations

```
get a i = a i
store a i e = a//[(i,e)]
```

2. Synthesizer infers spec from source

```
procedure sten(imin,imax,jmin,jmax,a,b)
  real,dim(imin:imax,jmin:jmax) :: a
  real,dim(imin:imax,jmin:jmax) :: b
  do j=jmin,jmax
    t = b(imin, j)
    do i=imin+1,imax
      q = b(i,j)
      a(i,j) = q + t
      t=q
    enddo
  enddo
end procedure
```

3. Retarget synthesized spec to Halide

codegen

```
int main() {
  ImageParam b(type_of<dbl>(),2);
  Func func;
  Var i, j;
  func(i,j) = b(i-1,j) + b(i,j);
  func.compile_to_file("ex1", b);
  return 0;
}
```

Lifted code can be executed on GPUs
17x speedup

Hardware: Domino → Programmable Switches

BAREFOOT NETWORKS

[SIGCOMM 16]

1. Define hardware building blocks as instructions

```
mux c v1 v2 = if c then v1 else v2
pairExec x y f g = (f x, g y)
inc x e = x + e
```

2. Synthesizer infers spec from source

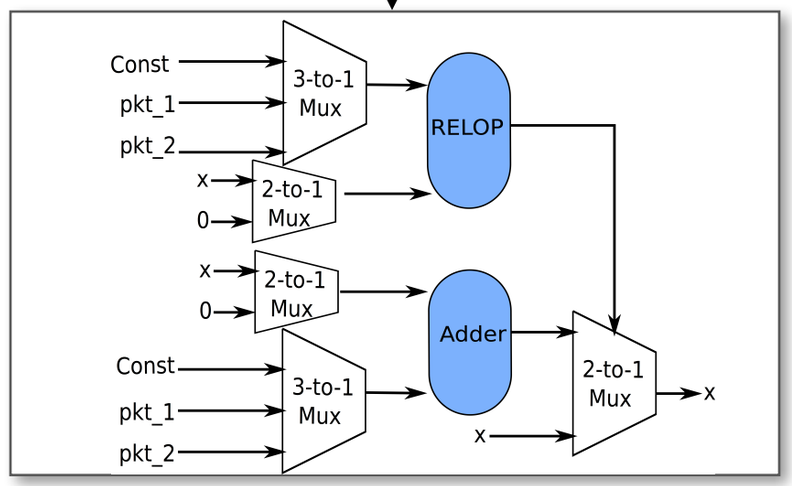
```
void flowlet (Packet p) {
  p.new_hop = hash3(p.sport, p.dport,
                    p.arrival) % HOPS;
  p.id = hash2(p.sport, p.dport) % FLOWS;

  if (p.arrival – last_time[p.id]>THRESHOLD)
    saved_hop[pkt.id] = p.new_hop;

  ...
}
```

3. Retarget spec to programmable switches

codegen



Compile 10 well-known data plane algorithms to switches & run at line rate

Parallel Frameworks: Sequential Java → Hadoop

[SYNT 16, SIGMOD 17]

1. Define semantics of map and reduce

```
map l f = [f t | t <- l]
reduce l f i = foldl f i l
```

2. Synthesizer infers spec from source

```
// sequential implementation
int regress(Point [] points)
{
   int SumXY = 0;
   for (Point p : points){
      SumXY += p.x * p.y;
   }

   return SumXY;
}
```

3. Retarget synthesized spec to Hadoop and Spark

codegen

```
void map(Object key, Point [] value)
{  for (Point p : points)
     emit("sumxy", p.x * p.y);
}}

void reduce(Text key, int [] vs)
{  int SumXY = 0;
   for (Integer val : vs)
     SumXY = SumXY + val;
   emit(key, SumXY); }
```

Lifted code can be optimized by Hadoop/Spark
32x speedup

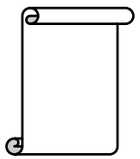
Casper

Compiling Sequential Java to Hadoop

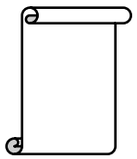
An Illustration of the MetaLift Toolchain

Demo

Casper Architecture

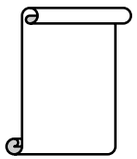


Target code fragments

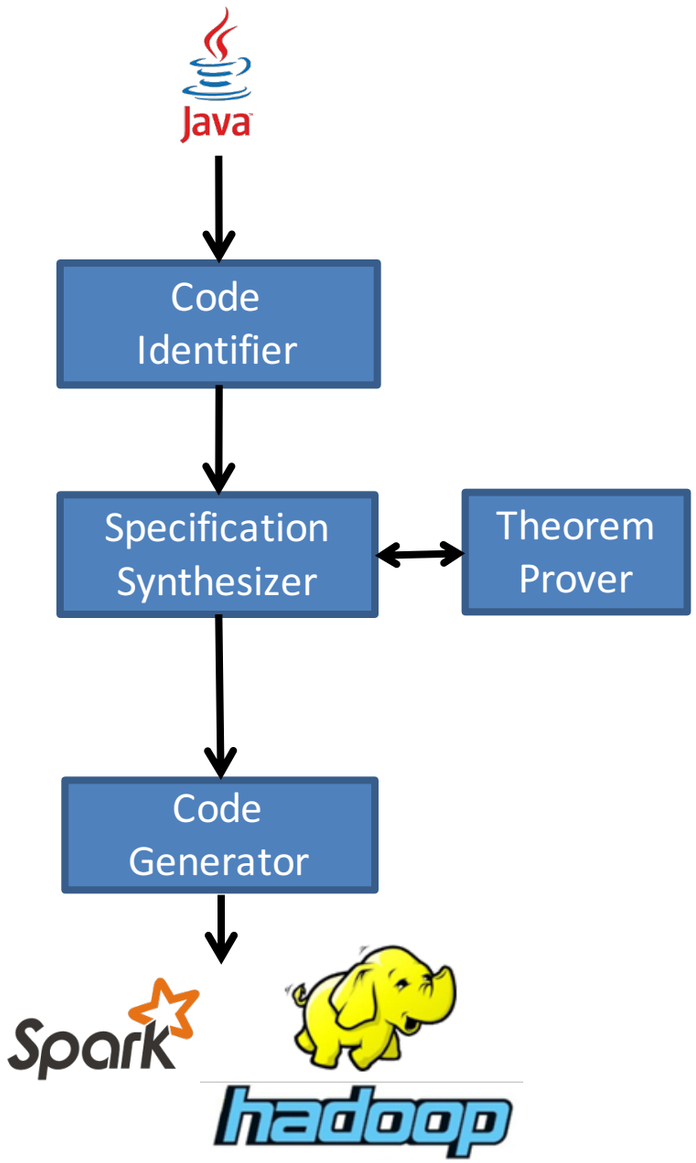


DSL semantics

Search space description

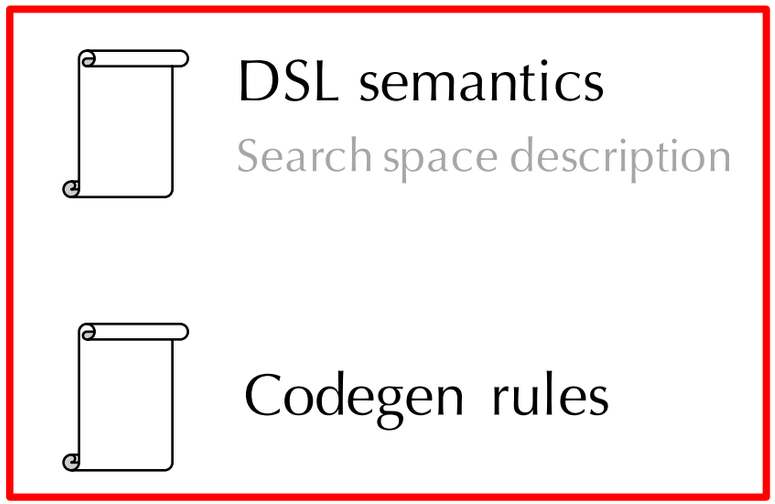


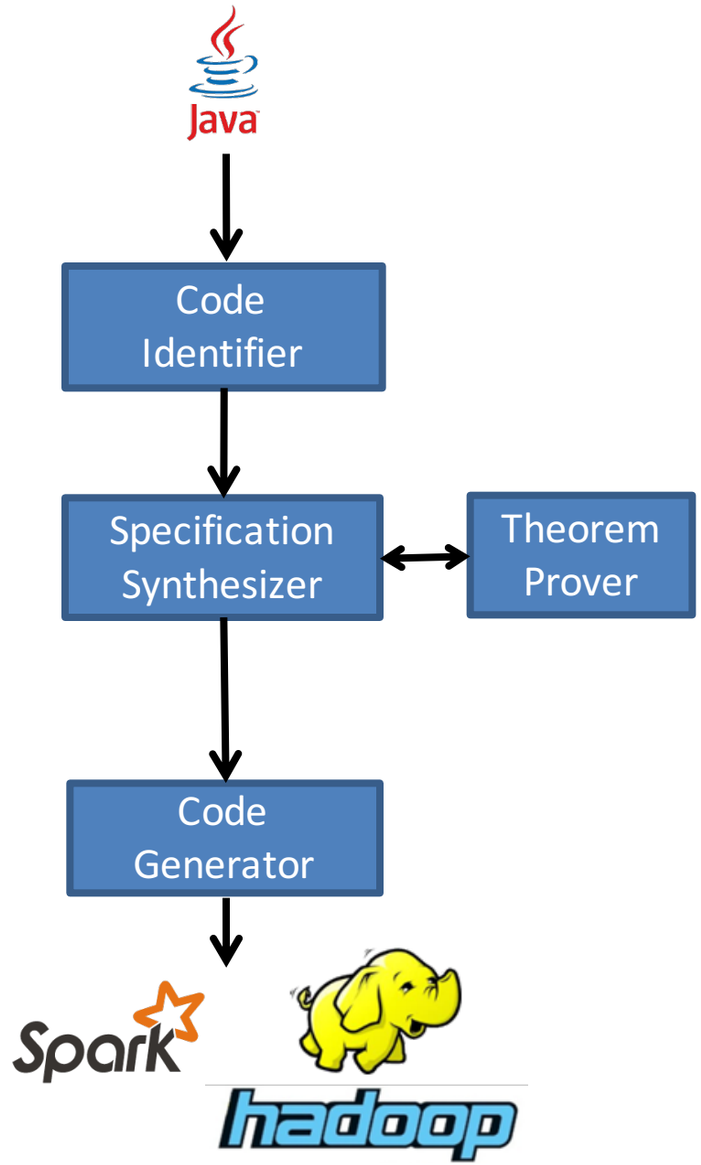
Codegen rules

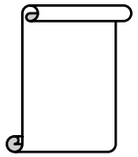


Casper Architecture

```
Java: while (*) { * }
```



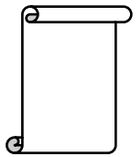




DSL semantics

```
map :: [(a)] -> (a -> (k,v)) -> [(k,v)]
map l f = [f t | t <- l]

reduce :: [(k,[v])] -> (v -> v -> v) -> [(k,v)]
reduce l f i = [(k, foldl f i v) | (k,v) <- l]
```



Code generation

```
translate (map l f) =
  "void map(...) { for (t : l) emit(" + translate(f) + "; }"

translate (reduce l f i) = ...
```

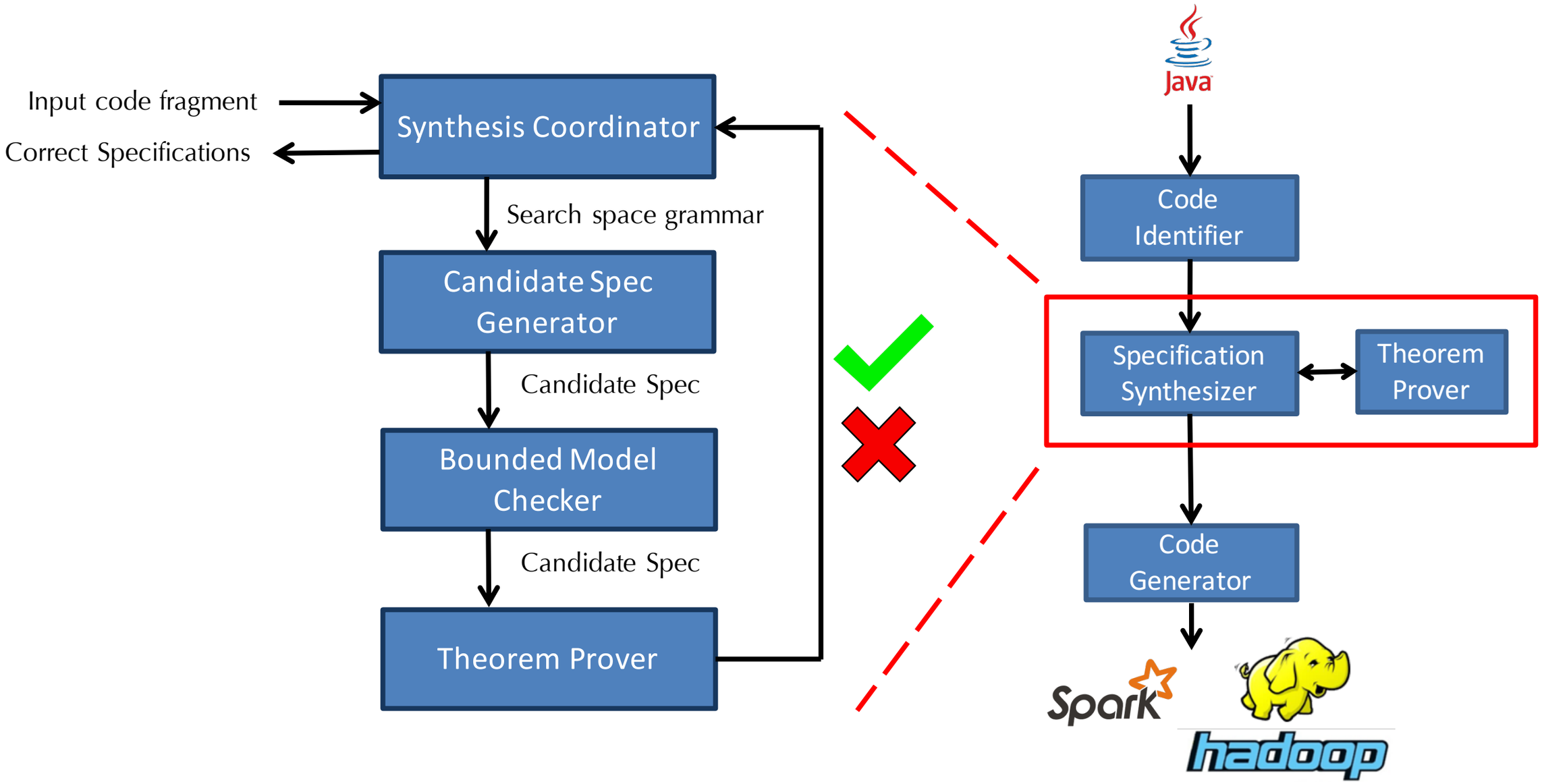
Search Space Grammar

$$e := (e_1, e_2) \mid e + e \mid e - e \mid var \mid lit$$

$$\mid \; e * e \mid e / e \mid \text{if } e_1 \text{ then } e_2 \text{ else } e_3$$

Automatically generated for each input code fragment

Increases expressivity incrementally


Input code fragment
Correct Specifications
Synthesis Coordinator
Search space grammar
Candidate Spec Generator
Candidate Spec
Bounded Model Checker
Candidate Spec
Theorem Prover
Java
Code Identifier
Specification Synthesizer
Theorem Prover
Code Generator
Spark
hadoop

Evaluation: Benchmarks

60 benchmarks collected from **5 benchmark suites**

Phoenix: Classic MapReduce problems such as WordCount, 3D Histogram, Linear Regression and StringMatch.

Bigλ: Big-Data analytical benchmarks including basic sentiment analysis, database operations and Wikipedia log processing.

Arithmetic: Mathematical functions such as Max, Delta and Conditional Sum.

Statistical: Statistical analysis such as Mean, Covariance and Standard Error.

Fiji: Three image processing kernels: RedToMagenta, Temporal Median and Trails.

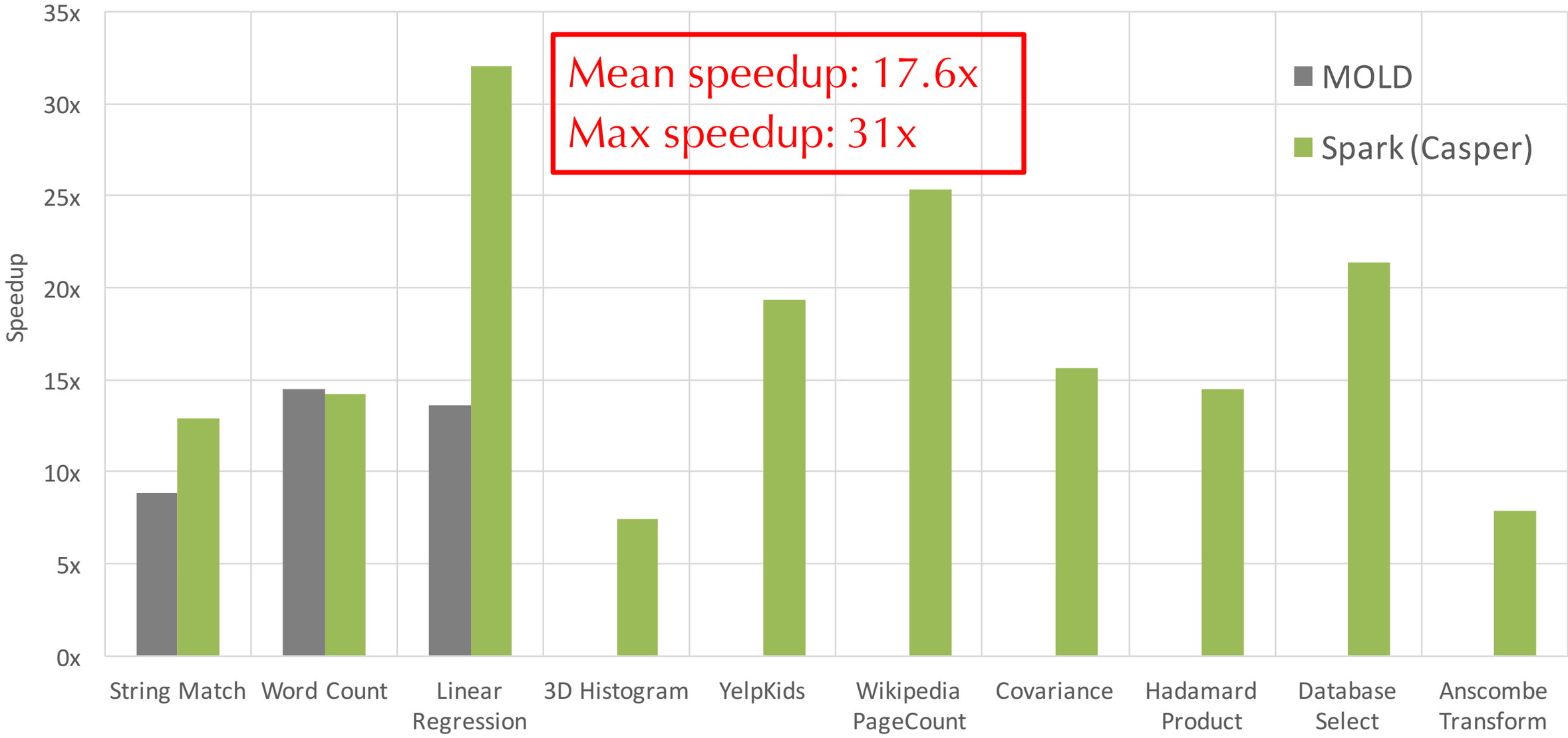
Feasibility: Does Casper work?

Benchmark	Extracted	Translated
Phoenix	7	11
Bigλ	6	8
Arithmetic	11	11
Statistical	18	19
Fiji	8	11
Total	**50**	**60**

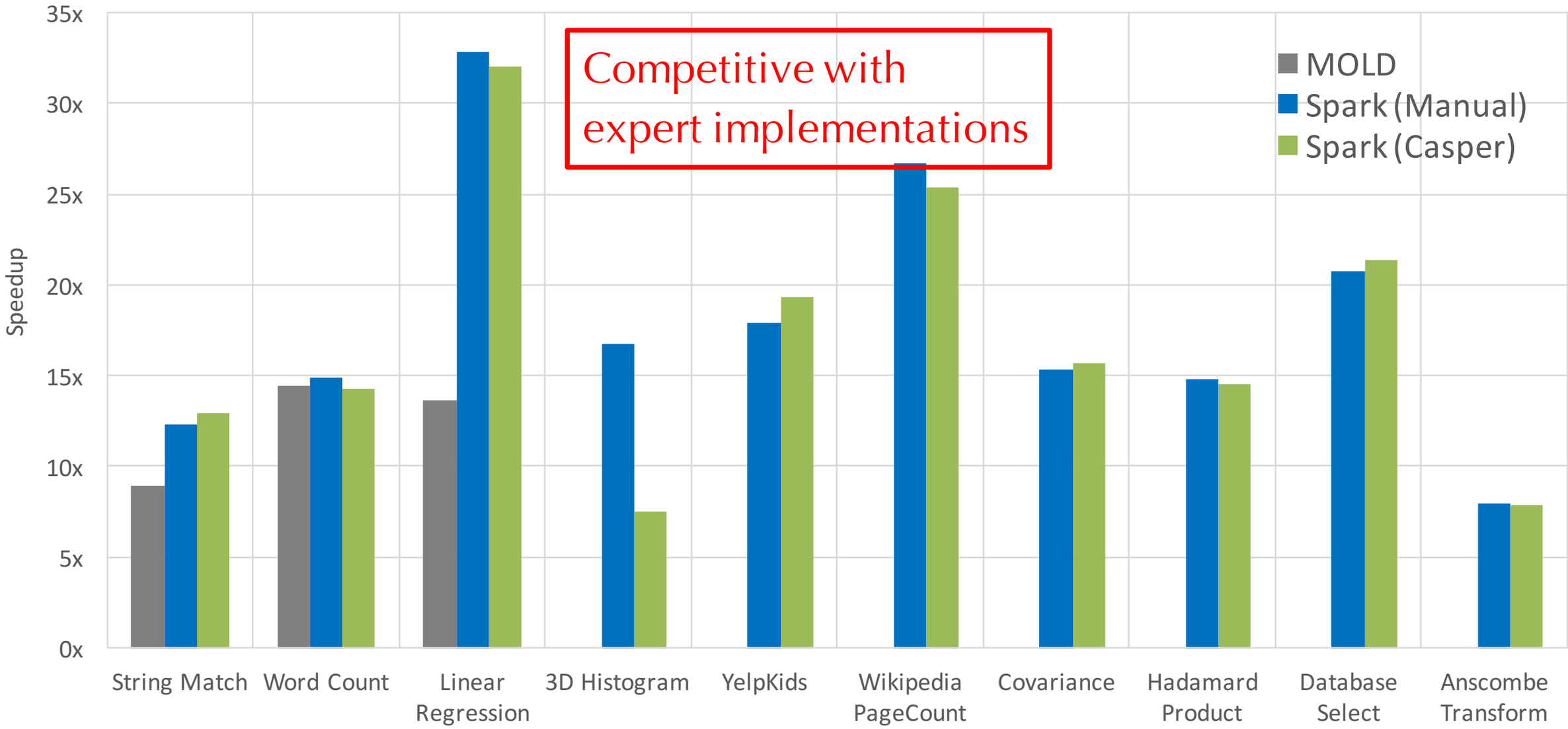
Causes of failures

- 3 caused by calls to unsupported to external library methods
- 7 caused by program constructs not currently expressible using MetaLift

Performance: Is Casper competitive?



Performance: Is Casper competitive?



Performance Analysis: Is Casper Practical?

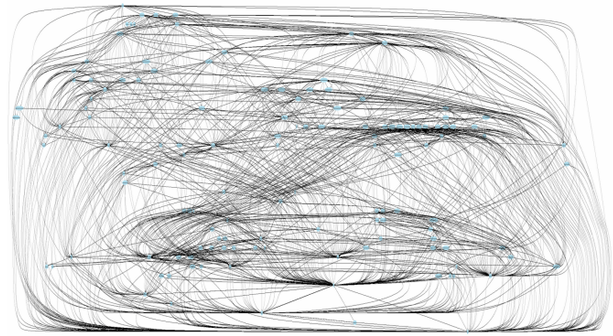
Mean compilation time for one benchmark was **5.8 minutes.**

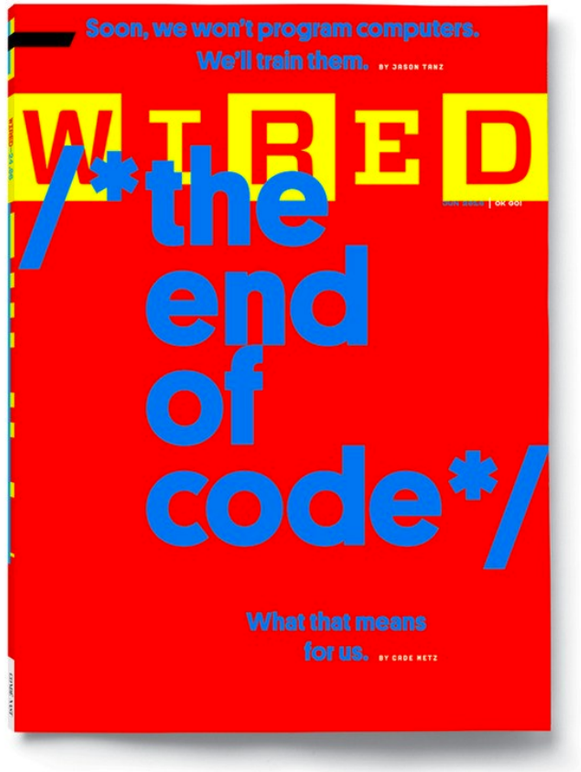
Median compilation time for one benchmark was **36.6 seconds.**

Mean time to reach first correct translation was only **48 seconds!**

Benchmark	Solutions Generated with Pruning	Solutions Generated without Pruning
WordCount	2	827
3D Histogram	5	118
Yelp: Kid Friendly Restaurants	1	286


KEEP
CALM
AND
BANG YOUR HEAD
AGAINST THE WALL




Soon, we won't program computers.
We'll train them. BY JASON TANZ
WIRED
/*the
end
of
code*/
What that means
for us. BY CADE METZ