SIGMOD 2018

Automatically Leveraging MapReduce Frameworks for Data-Intensive Applications

Maaz Bin Safeer Ahmad
(University of Washington)

Alvin Cheung
(University of Washington)

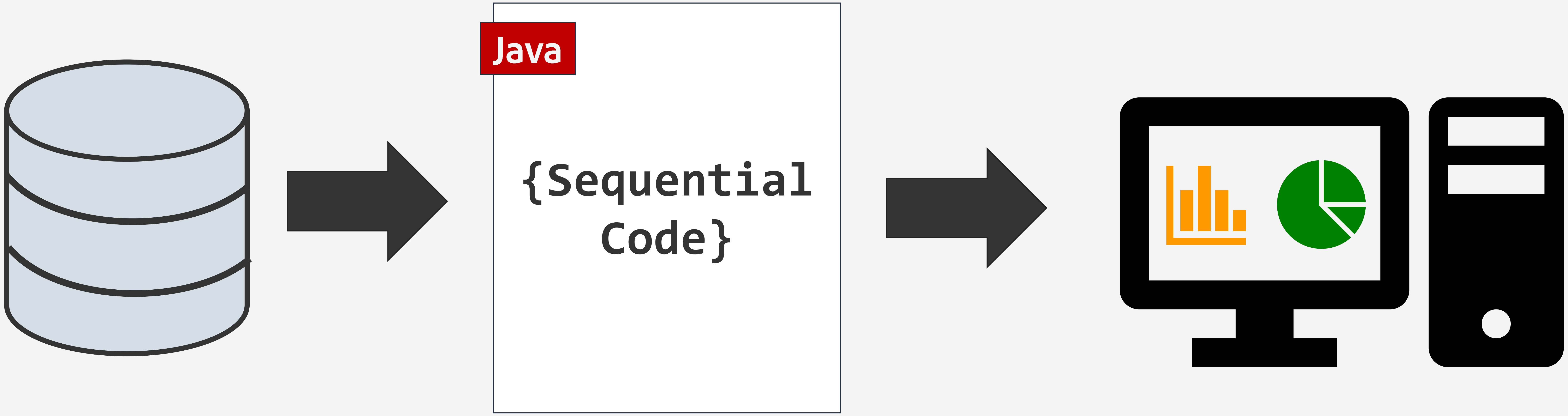
PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

UW DB

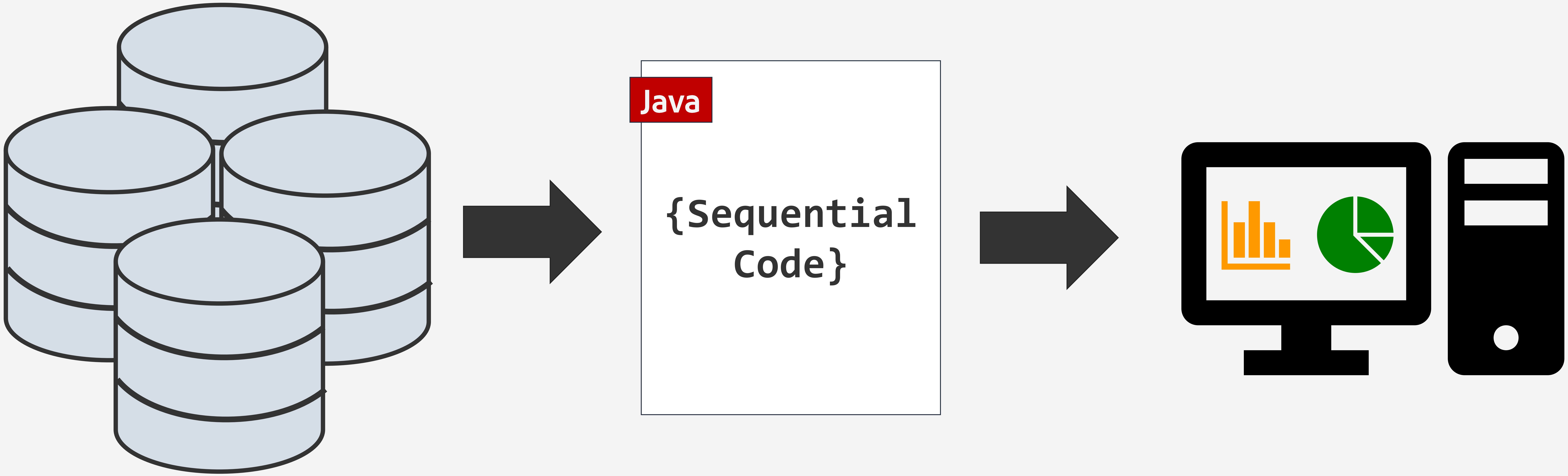
PLSE

Why translate sequential code to MapReduce?

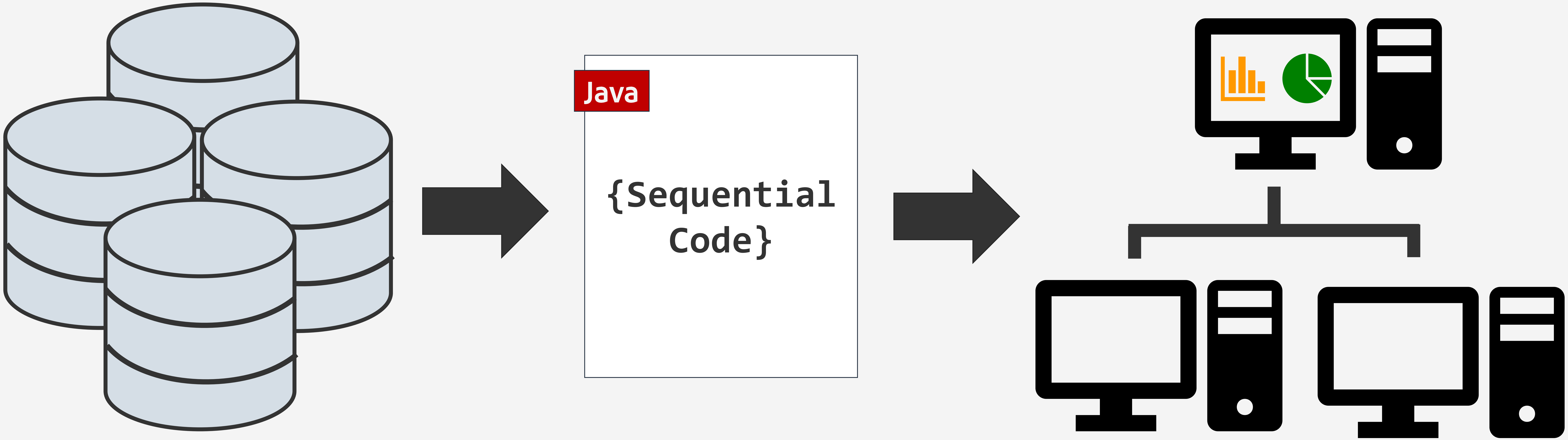
Optimizing Existing Applications



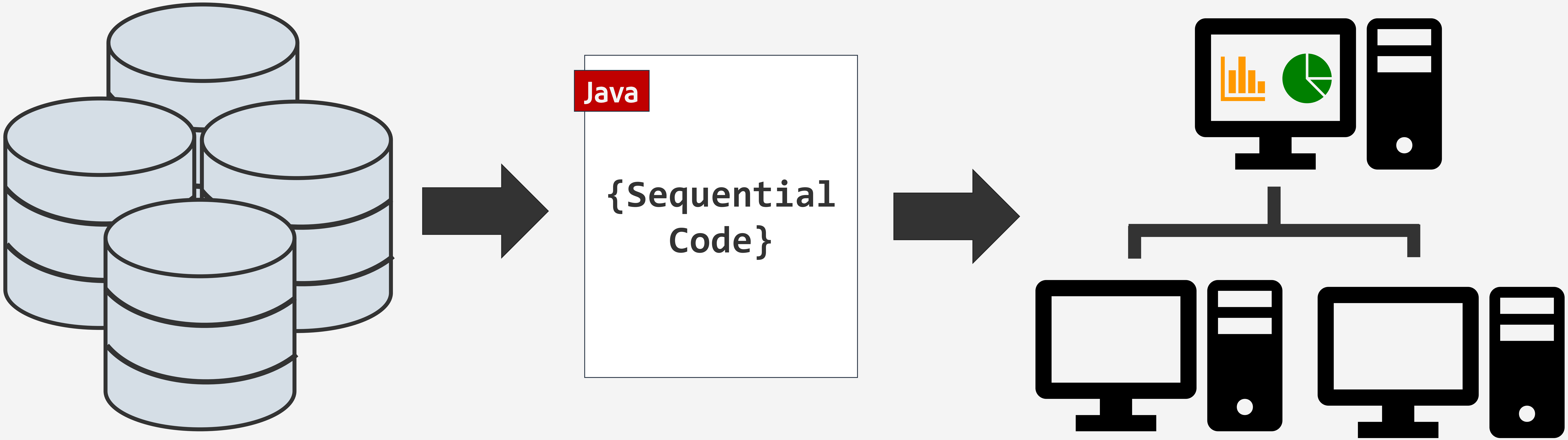
Optimizing Existing Applications



Optimizing Existing Applications

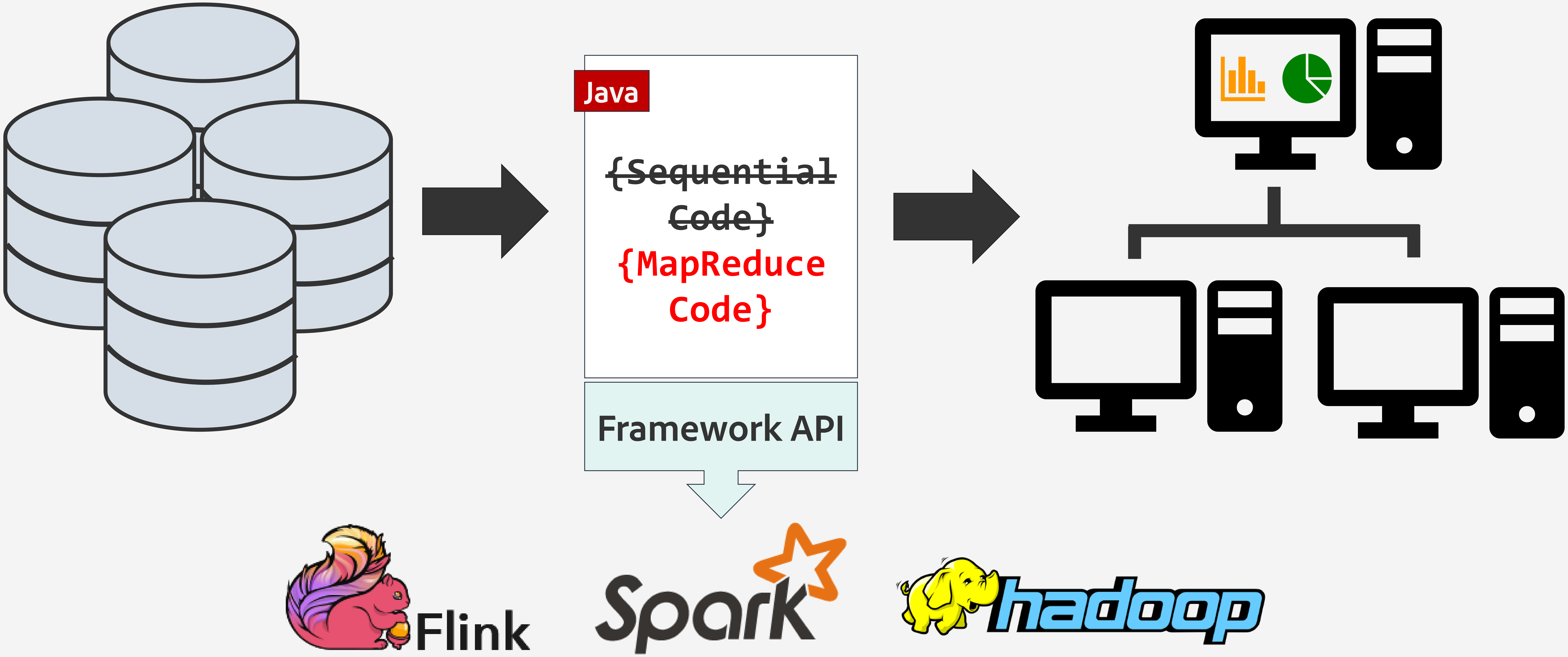


Optimizing Existing Applications



Flink Spark hadoop

Optimizing Existing Applications



Option 1: Manual Re-write

- Recurring
- Error-prone
- Tedious
- Requires Expertise

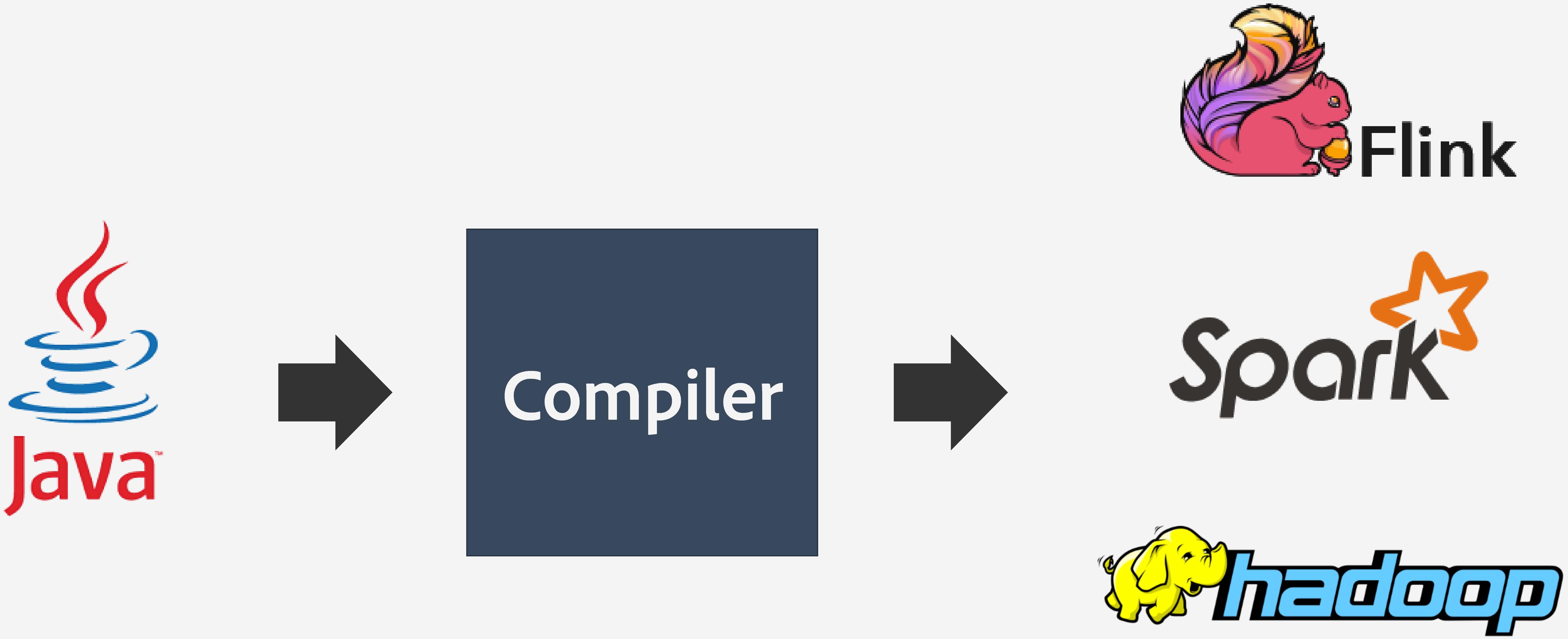
Java

Flink

Spark

hadoop

Option 2: Build a Compiler



Why is the sequential to MapReduce re-write difficult to automate?

Syntax Directed Translation

- Traditionally compilers use pattern-matching rules to do code transformations.

```
for (int i=0; i < $in.size(); ++i)
{
  if ($in.get(i) > $c)
    $out.add($in.get(i));
}
```

Syntax Directed Translation

- Traditionally compilers use pattern-matching rules to do code transformations.

```
for (int i=0; i < $in.size(); ++i)
{
  if ($in.get(i) > $c)
    $out.add($in.get(i));
}
```

```
$out.union($in.filter(e -> e > $c));
```

Syntax Directed Translation

- Traditionally compilers use pattern-matching rules to do code transformations.

```
for (int i=0; i<N; i++) {
  HashMap<String,Double> contrib = new HashMap<>();
  for (Map.Entry<String,Double> r : ranks.entrySet()) {
    List<String> urls = grouped_links.get(r.getKey());
    if(urls != null) {
      int size = urls.size();
      urls.forEach(dst -> {
        if (!contrib.containsKey(dst))
          contrib.put(dst, 0.0);
        contrib.put(dst, contrib.get(dst) +
                            (r.getValue() / size));
      });}}
  for (String dst : contrib.keySet())
    ranks.put(dst, contrib.get(dst) * 0.85 + 0.15);
}
```

Syntax Directed Translation

- Traditionally compilers use pattern-matching rules to do code transformations.

```
for (int i=0; i<N; i++) {
  HashMap<String,Double> contrib = new HashMap<>();
  for (Map.Entry<String,Double> r : ranks.entrySet()) {
    List<String> urls = grouped_links.get(r.getKey());
    if(urls != null) {
      int size = urls.size();
      urls.forEach(dst -> {
        if (!contrib.containsKey(dst))
          contrib.put(dst, 0.0);
        contrib.put(dst, contrib.get(dst) +
                         (r.getValue() / size));
      });}}
  for (String dst : contrib.keySet())
    ranks.put(dst, contrib.get(dst) * 0.85 + 0.15);
}
```

→ ??

A Synthesis Based Approach

Java

```java
int sum = 0;
for (Integer val : data) {
  sum += val * val;
}
```

Re-write rules

Spark

```java
int sum = 0;
sum = data.map(val -> val*val)
          .reduce((v1,v2) -> v1+v2);
```

A Synthesis Based Approach

$\lambda_{m(v)} \rightarrow v \ * v$

$\lambda_{r(v_1, v_2)} \rightarrow v_1 + v_2$

$sum \ = \ reduce(map(data, \lambda_m), \lambda_r);$

Codegen

Java

```java
int sum = 0;
for (Integer val : data) {
  sum += val * val;
}
```

Re-write rules

Spark

```java
int sum = 0;
sum = data.map(val -> val*val)
          .reduce((v1,v2) -> v1+v2);
```

A Synthesis Based Approach

$$\lambda_{m(v)} \rightarrow v \ * v$$

$$\lambda_{r(v_1, v_2)} \rightarrow v_1 + v_2$$

$$sum \ = \ reduce(map(data, \lambda_m), \lambda_r);$$

??

Codegen

Java

```
int sum = 0;
for (Integer val : data) {
  sum += val * val;
}
```

Re-write rules

Spark

```
int sum = 0;
sum = data.map(val -> val*val)
          .reduce((v1,v2) -> v1+v2);
```

A Synthesis Based Approach

$$\lambda_{m(v)} \to v \ * v$$

$$\lambda_{r(v_1, v_2)} \to v_1 + v_2$$

$$sum \ = \ reduce(map(data, \lambda_m), \lambda_r);$$

Program Synthesis

Codegen

Java

```
int sum = 0;
for (Integer val : data) {
  sum += val * val;
}
```

Re-write rules

Spark

```
int sum = 0;
sum = data.map(val -> val*val)
             .reduce((v1,v2) -> v1+v2);
```

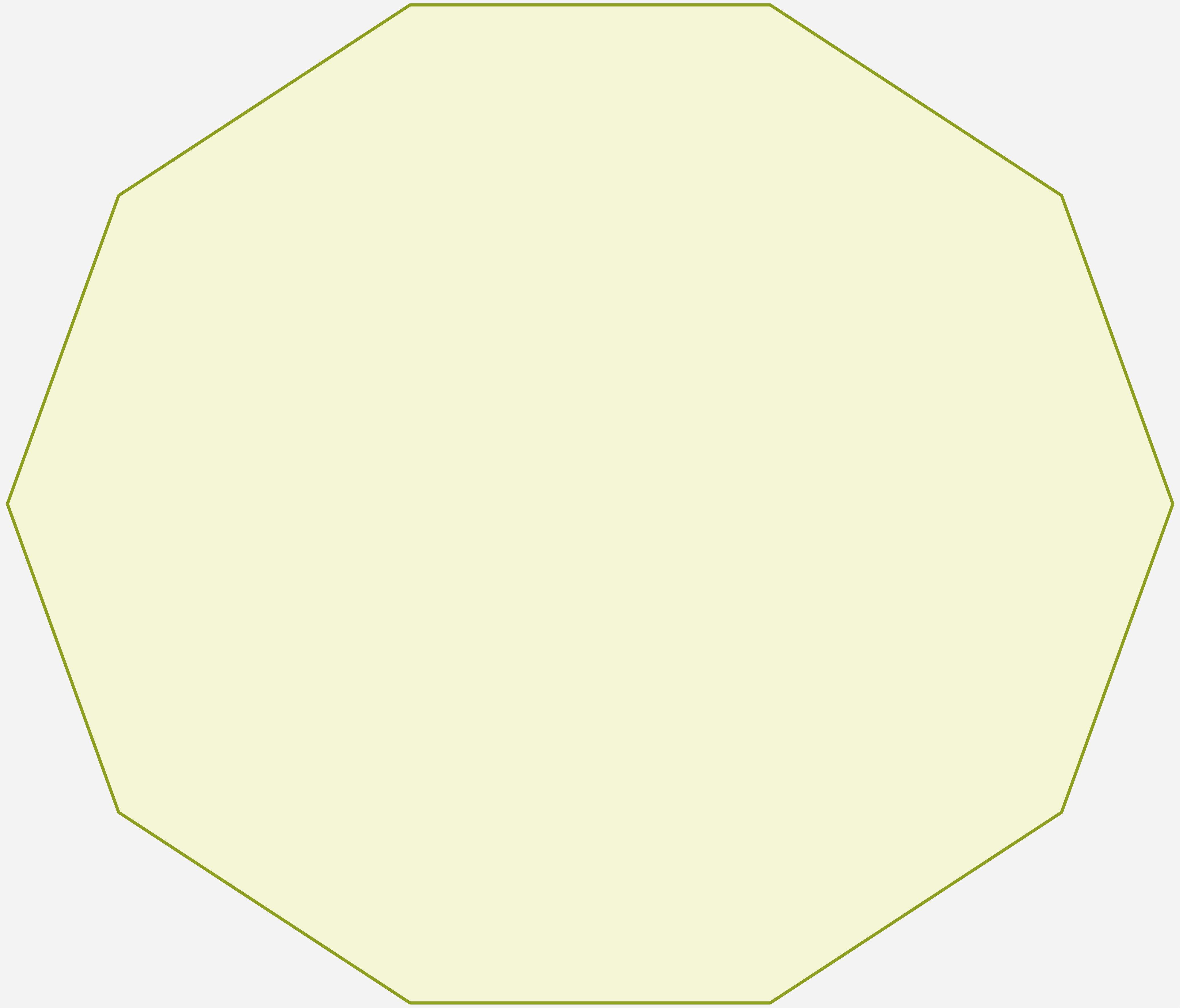
Program Synthesis

```
for (int i=0; i<N; i++) {
  HashMap<String,Double> contrib = new HashMap<>();
  for (Map.Entry<String,Double> r : ranks.entrySet()) {
    List<String> urls = grouped_links.get(r.getKey());
    if(urls != null) {
      int size = urls.size();
      urls.forEach(dst -> {
        if (!contrib.containsKey(dst))
          contrib.put(dst, 0.0);
        contrib.put(dst, contrib.get(dst) +
                         (r.getValue() / size));
      });}}
  for (String dst : contrib.keySet())
    ranks.put(dst, contrib.get(dst) * 0.85 + 0.15);
}
```

Program Synthesis

```
for (int i=0; i<N; i++) {
  HashMap<String,Double> contrib = new HashMap<>();
  for (Map.Entry<String,Double> r : ranks.entrySet()) {
    List<String> urls = grouped_links.get(r.getKey());
    if(urls != null) {
      int size = urls.size();
      urls.forEach(dst -> {
        if (!contrib.containsKey(dst))
          contrib.put(dst, 0.0);
        contrib.put(dst, contrib.get(dst) +
                         (r.getValue() / size));
      });}}
  for (String dst : contrib.keySet())
    ranks.put(dst, contrib.get(dst) * 0.85 + 0.15);
}
```

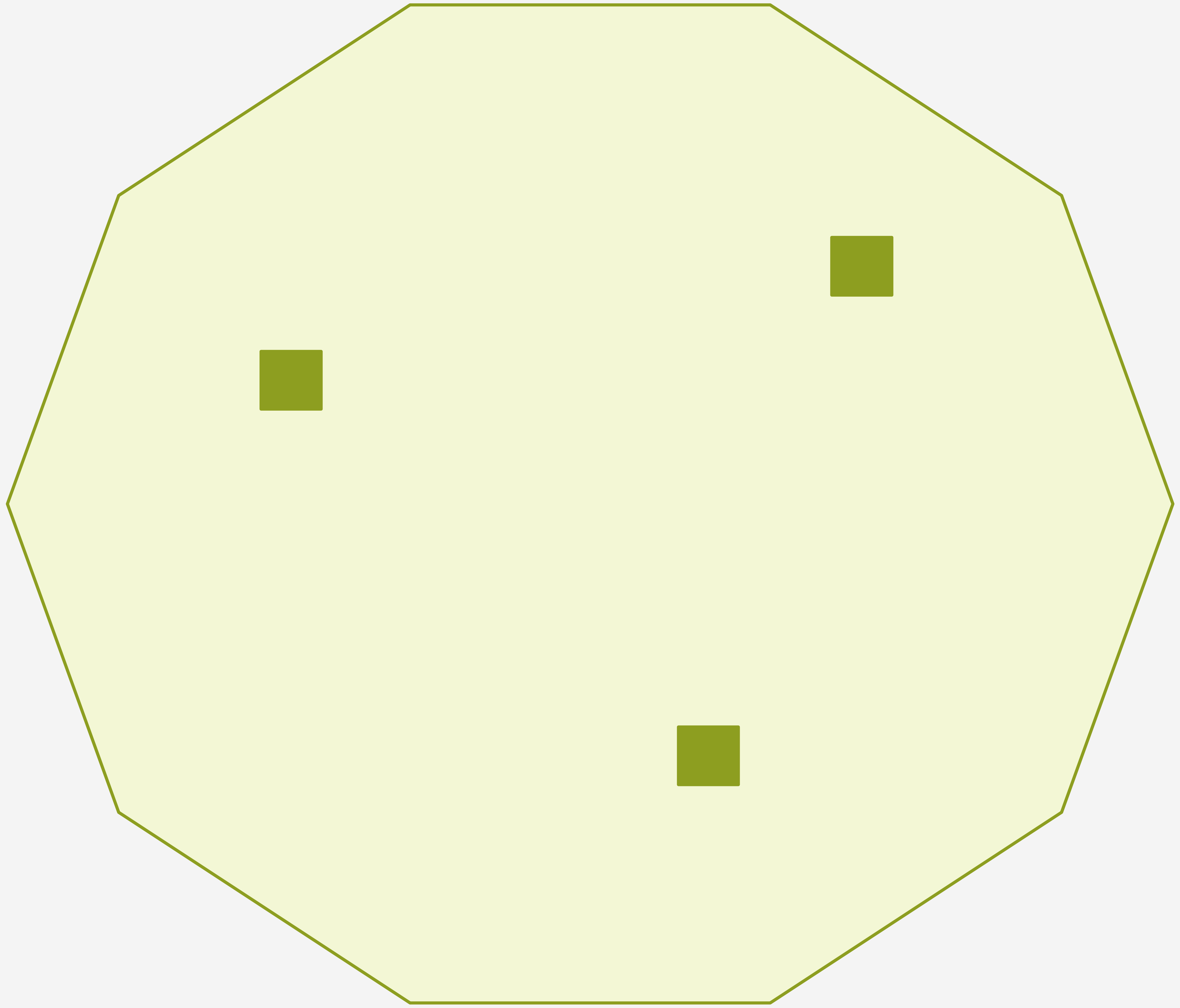
Space of programs constructed using ***map*** and ***reduce*** operators



Program Synthesis

Space of programs constructed using *map* and *reduce* operators

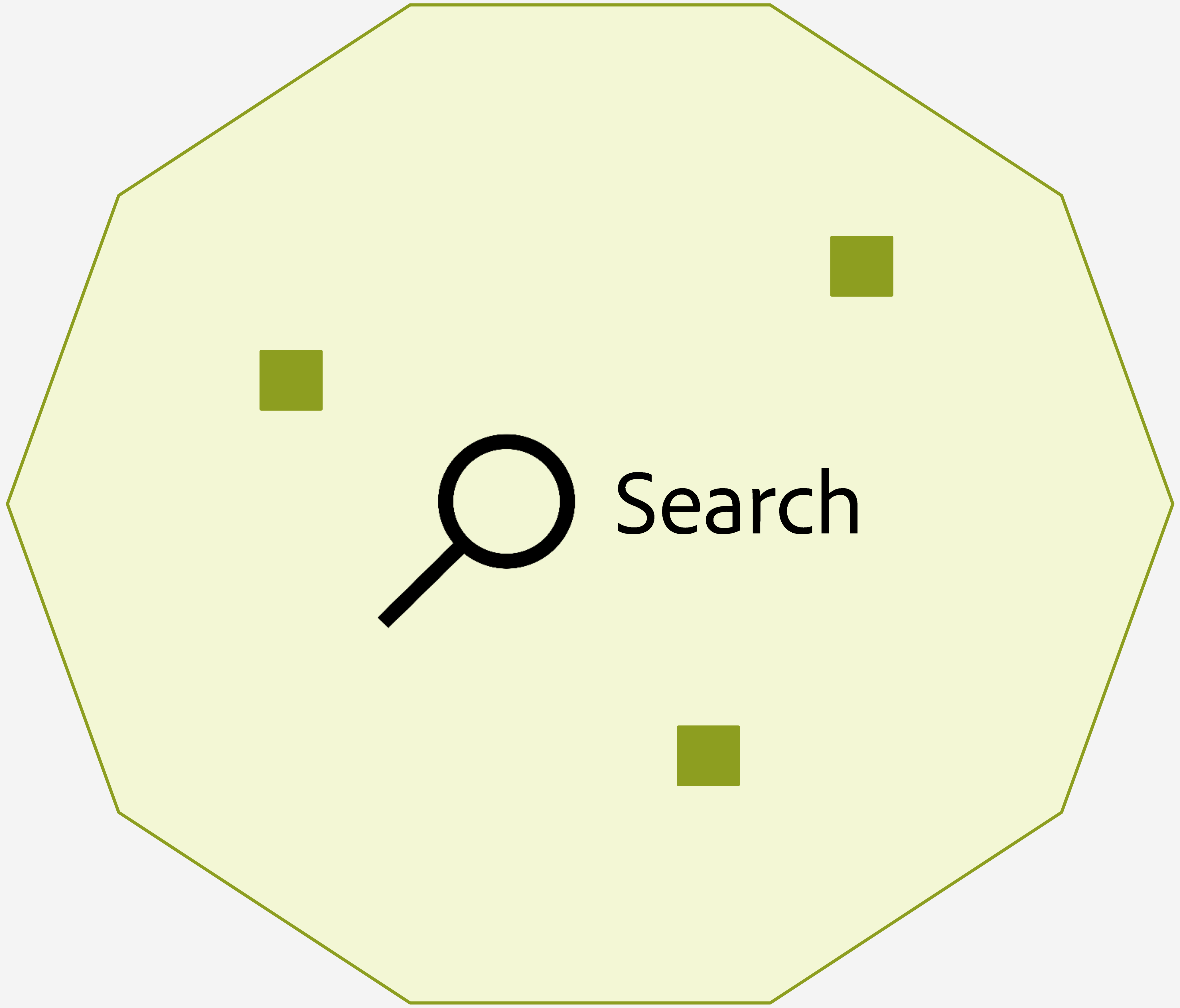
```
for (int i=0; i<N; i++) {
  HashMap<String,Double> contrib = new HashMap<>();
  for (Map.Entry<String,Double> r : ranks.entrySet()) {
    List<String> urls = grouped_links.get(r.getKey());
    if(urls != null) {
      int size = urls.size();
      urls.forEach(dst -> {
        if (!contrib.containsKey(dst))
          contrib.put(dst, 0.0);
        contrib.put(dst, contrib.get(dst) +
                         (r.getValue() / size));
      });}}
  for (String dst : contrib.keySet())
    ranks.put(dst, contrib.get(dst) * 0.85 + 0.15);
}
```



Program Synthesis

```
for (int i=0; i<N; i++) {
  HashMap<String,Double> contrib = new HashMap<>();
  for (Map.Entry<String,Double> r : ranks.entrySet()) {
    List<String> urls = grouped_links.get(r.getKey());
    if(urls != null) {
      int size = urls.size();
      urls.forEach(dst -> {
        if (!contrib.containsKey(dst))
          contrib.put(dst, 0.0);
        contrib.put(dst, contrib.get(dst) +
                         (r.getValue() / size));
      });}}
  for (String dst : contrib.keySet())
    ranks.put(dst, contrib.get(dst) * 0.85 + 0.15);
}
```

Space of programs constructed using
map and ***reduce*** operators

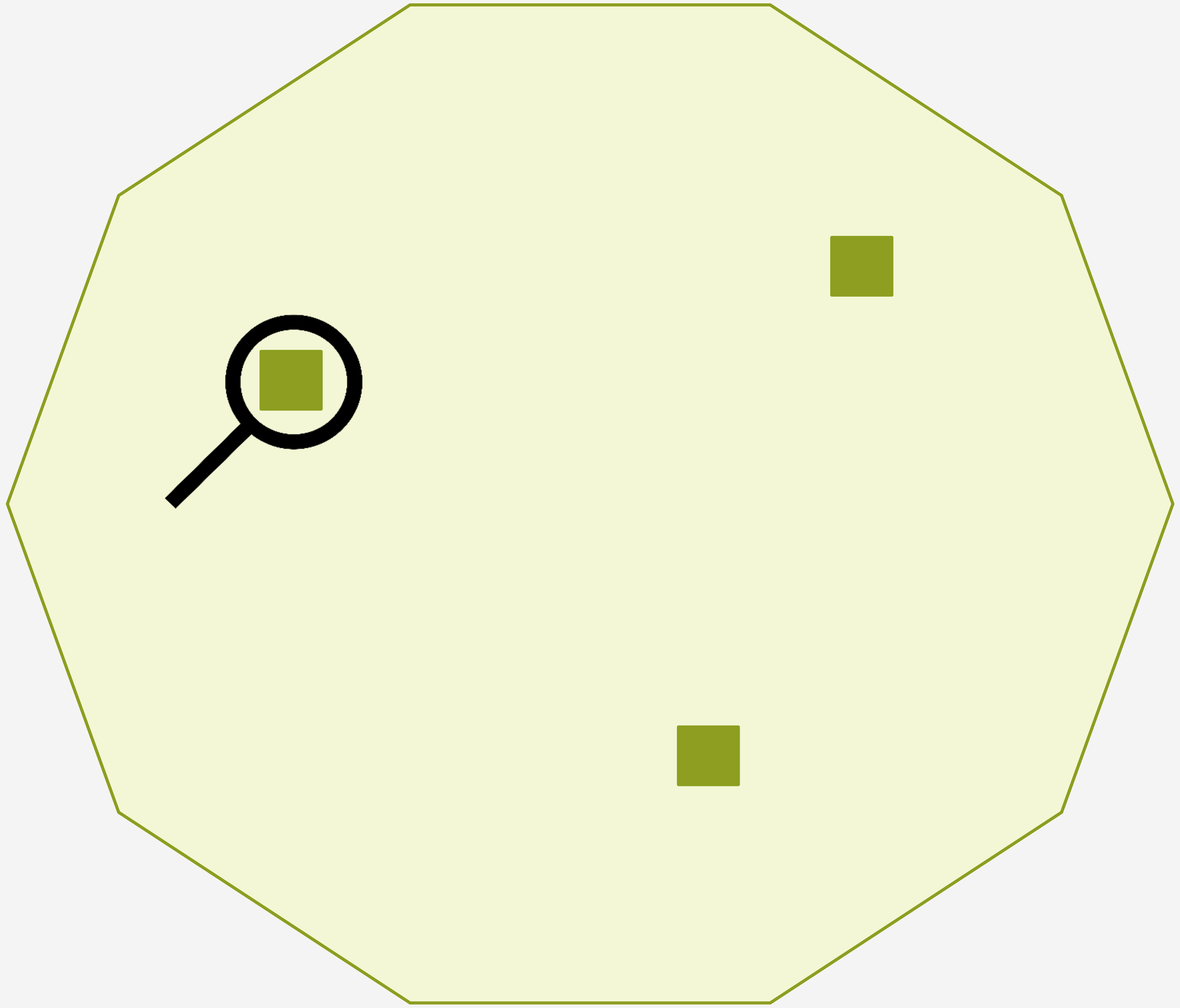


Program Synthesis

Space of programs constructed using *map* and *reduce* operators

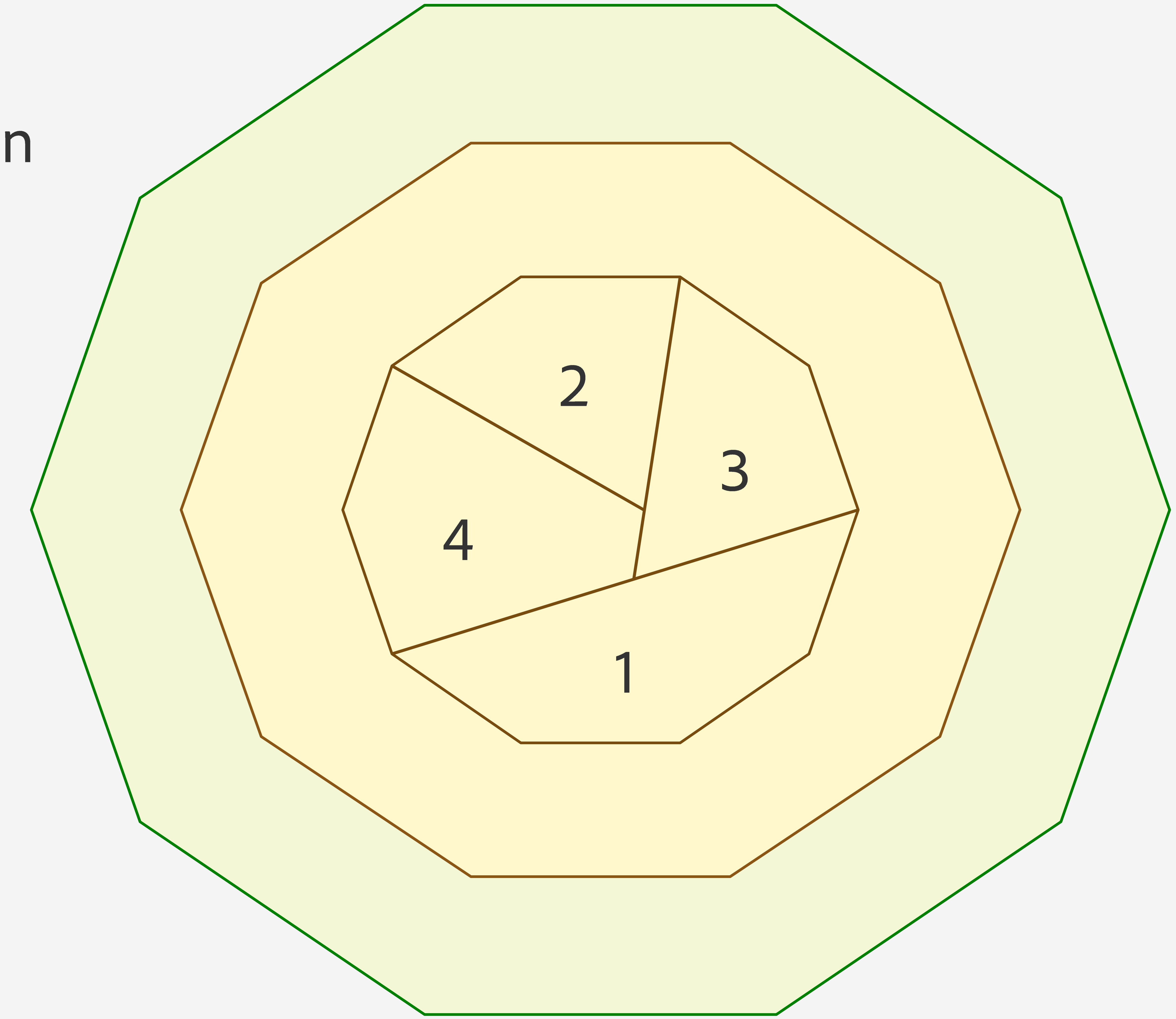
```
for (int i=0; i<N; i++) {
  HashMap<String,Double> contrib = new HashMap<>();
  for (Map.Entry<String,Double> r : ranks.entrySet()) {
    List<String> urls = grouped_links.get(r.getKey());
    if(urls != null) {
      int size = urls.size();
      urls.forEach(dst -> {
        if (!contrib.containsKey(dst))
          contrib.put(dst, 0.0);
        contrib.put(dst, contrib.get(dst) +
                                 (r.getValue() / size));
      });}}
  for (String dst : contrib.keySet())
    ranks.put(dst, contrib.get(dst) * 0.85 + 0.15);
}
```

$\equiv$



Making Search Manageable

- Design a concise API to express specification
- Use program analysis to specialize search
  - Ex: Only use specific operators
- Use incremental search
- Cost-based pruning



Introducing Casper

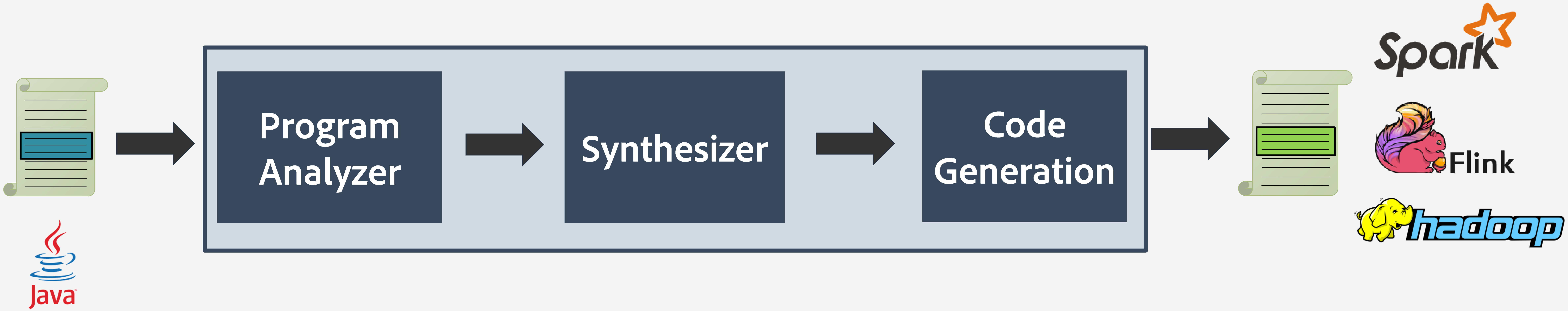
A compiler that automatically re-targets sequential Java applications to MapReduce frameworks.

Input

Un-annotated sequential Java application source code.

Output

An optimized version of the application that uses either Spark, Flink or Hadoop.



System Evaluation: Benchmarks

We used to optimize **55 benchmarks** collected from various sources.

Category	Description
Phoenix	Classical MapReduce problems
Fiji	Four open-sourced plugins implementing image processing algorithms
Bigλ	Big-Data analytics kernels
Ariths	Mathematical functions such as *sum*, *count*, *delta etc.*
Stats	Statistical functions such as *mean*, *variance*, *standard error* etc.
TPC-H	Java implementations for q1, q6, q15 and q17
Iterative	Page-rank and Logistic Regression based classification

Feasibility Analysis

Casper successfully translated **82** of the **101** identified code fragments across all benchmarks.

Causes of failures

- 3 caused by references to external library calls which were not currently supported
- 7 benchmarks could not be expressed in our intermediate language
- 9 benchmarks timed out (required more than 90 minutes)

Performance Analysis (Spark)

75GB data on a **10 node cluster** (8 cores, 30gb ram)

Performance Analysis (Spark)

75GB data on a **10 node cluster** (8 cores, 30gb ram)

How long does Casper take?

Mean compilation time for one benchmark was **11.4 minutes.**

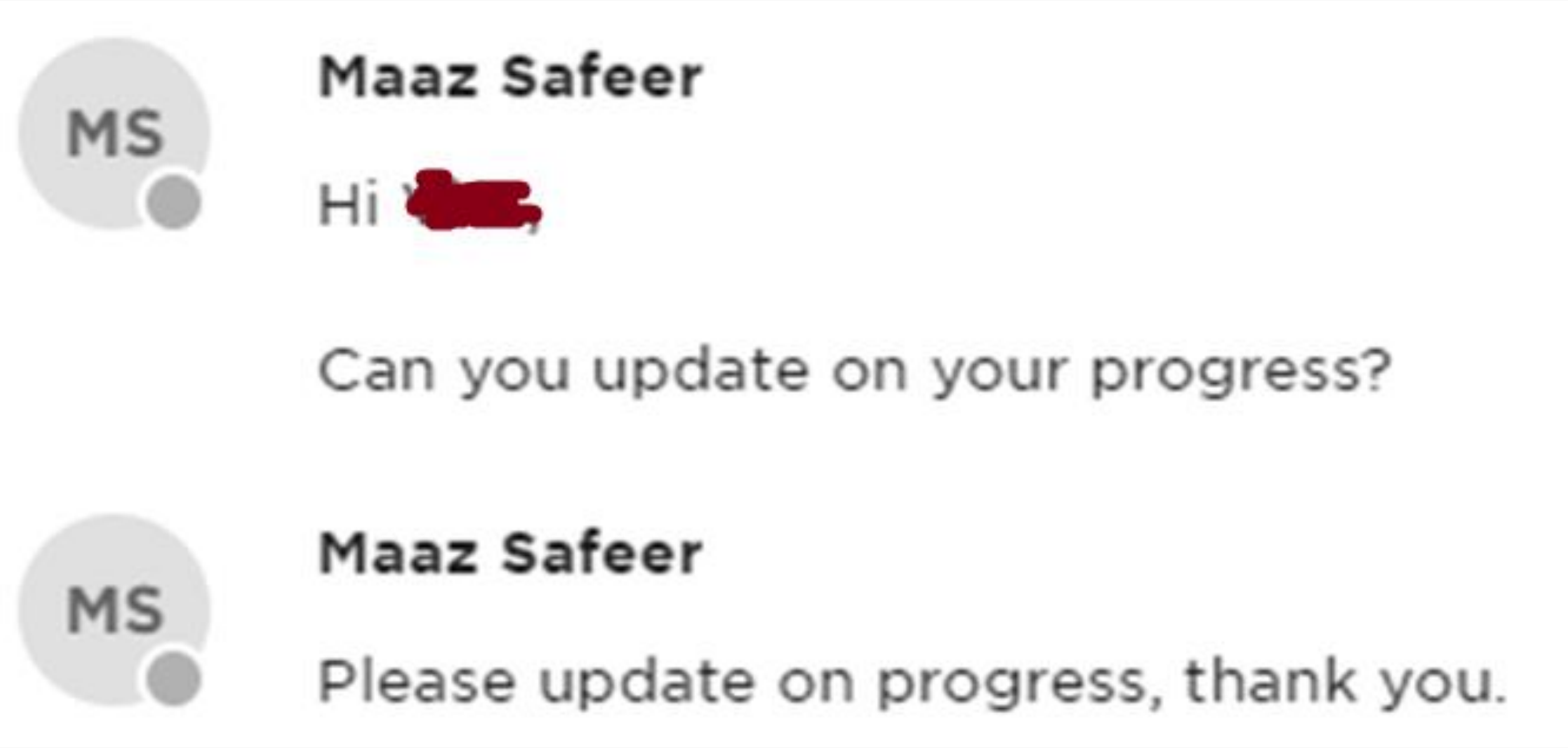
Median compilation time for one benchmark was **2.1 minutes.**

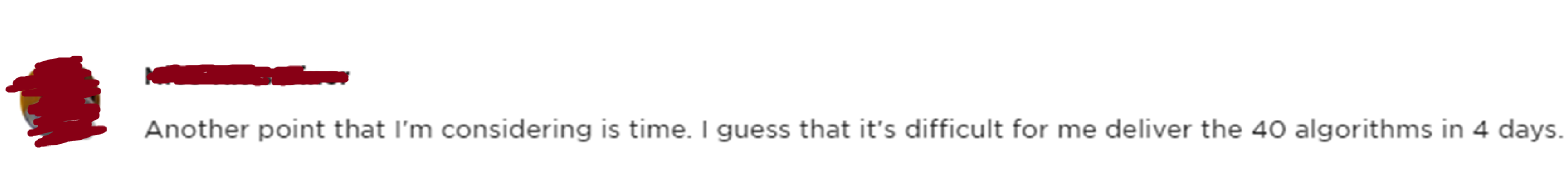
How long does Casper take?

Mean compilation time for one benchmark was **11.4 minutes.**

Median compilation time for one benchmark was **2.1 minutes.**

No managerial overhead!





Take-aways

- Casper can automatically translate a wide array of sequential applications to MapReduce.

- With average speedups of 15.6x, Casper is competitive with hand written translations.

Take-aways

- Casper can automatically translate a wide array of sequential applications to MapReduce.

- With average speedups of 15.6x, Casper is competitive with hand written translations.

casper.uwplse.org

Paper | Online Demo | Source Code