Actionable Performance Analyses

Marija Selakovic

marija.selakovic@huawei.com

London, 21.01.2020

TECHNISCHE UNIVERSITÄT DARMSTADT

HUAWEI

Software Performance

One of the most important aspects of software quality

- Efficiency
- Responsiveness
- Scaling
- Throughput
- User satisfaction

Software Performance

One of the most important aspects of software quality

- Efficiency
- Responsiveness
- Scaling
- Throughput
- User satisfaction

> “**1 second** *of load lag time would cost Amazon* **$1.6 billion in sales** *per year”* [1]
>
> **- Amazon**

1 *http://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-1.6-billion-sales*

Software Performance

One of the most important aspects of software quality

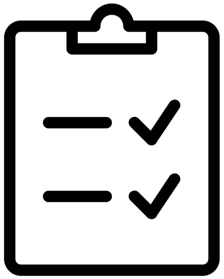
- Efficiency
- Responsiveness
- Scaling
- Throughput
- User satisfaction

> "**1 second** of load lag time would cost Amazon **$1.6 billion in sales** per year" [1]
>
> **- Amazon**

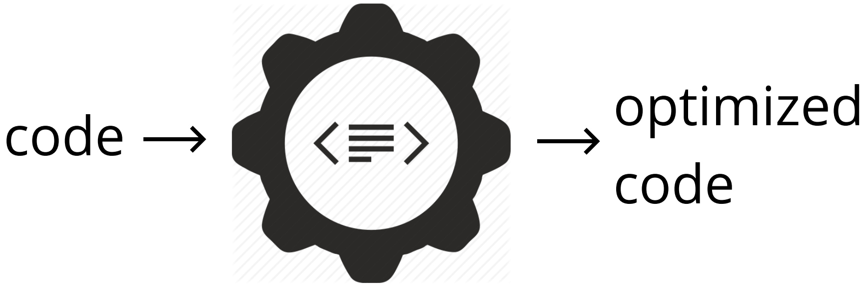
> "A **lag time of 400ms** results in **a decrease of 0.44% traffic** - In real terms this amounts to **440 million abandoned sessions/month** and a massive loss in advertising revenue for Google" [2]
>
> **- Google**

1 *http://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-1.6-billion-sales*
2 *http://www.cedexis.com/blog/for-google-400ms-of-increased-page-load-time-results-in-044-lost-search-sessions/*

Approaches for Improving Software Performance



CPU profiling



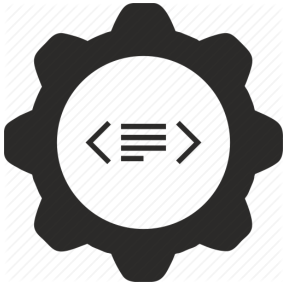
Compiler optimizations

Manual tuning

Approaches for Improving Software Performance

CPU profiling

code →



→ optimized code

Compiler optimizations

Manual tuning

code →

→ actionable suggestions

Actionable Performance Analyses

Actionable Performance Analyses

- Suggest concrete code changes
- Demonstrate the impact of applying optimizations
- Optimizations that are:
  - *exploitable* - easy to understand and apply
  - *effective* - lead to significant performance improvements
  - *recurring* - applicable across multiple projects

This Talk

- Reordering opportunities [1]
- Method inlining in Big Data system [2]

This Talk

- Reordering opportunities [1]
- Method inlining in Big Data system [2]

[1] Selakovic et al. *An Actionable Performance Profiler for Optimizing the Order of Evaluations* (ISSTA'17)

[2] Selakovic et al. *Cross-Language Optimizations in Big Data Systems: A Case Study of SCOPE* (ICSE-SEIP'18)

Inefficient Order of Evaluations

expensiveAndUnlikely() && cheapAndLikely()

Inefficient Order of Evaluations

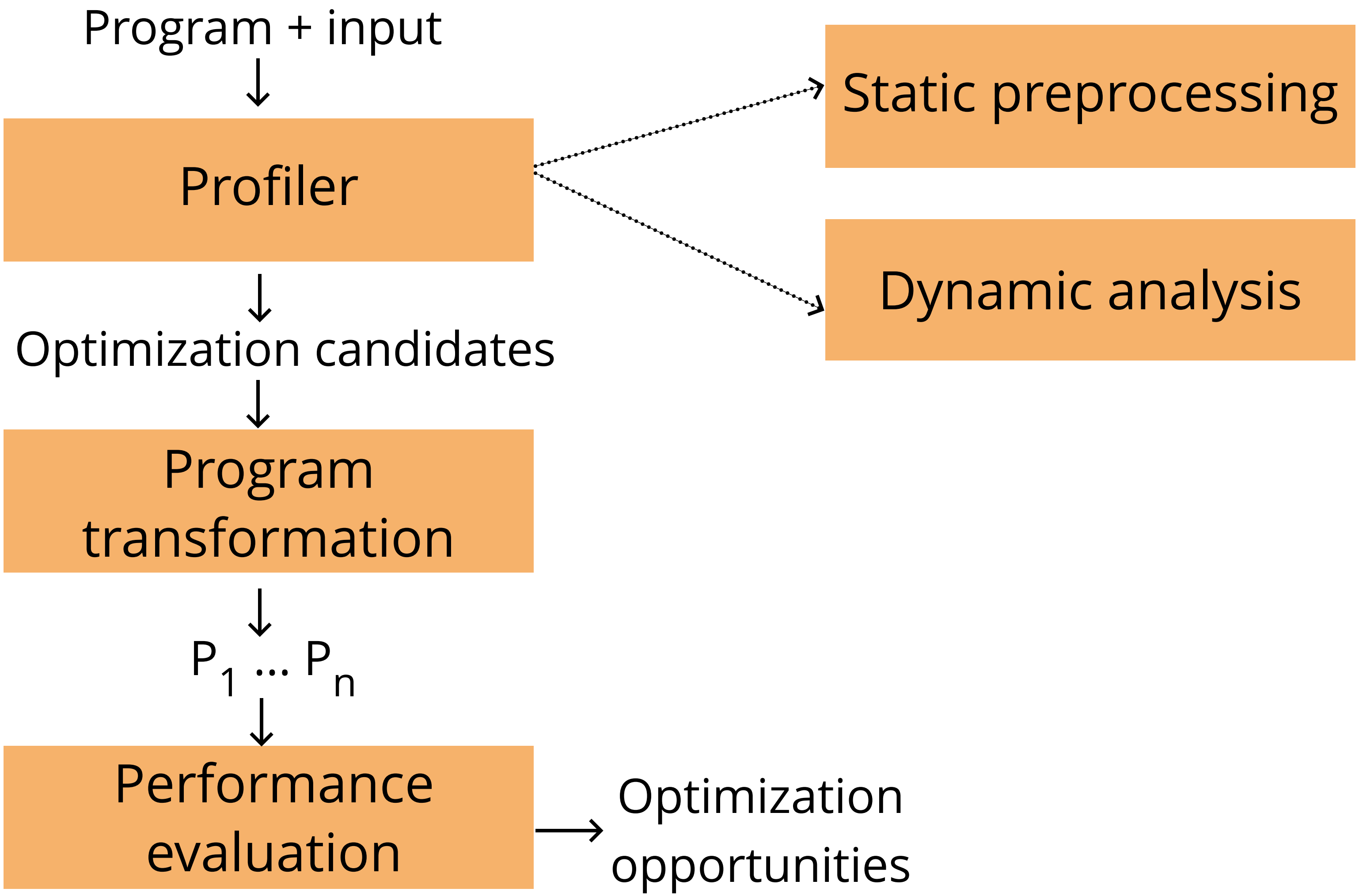
expensiveAndUnlikely() && cheapAndLikely()

Inefficient Order of Evaluations

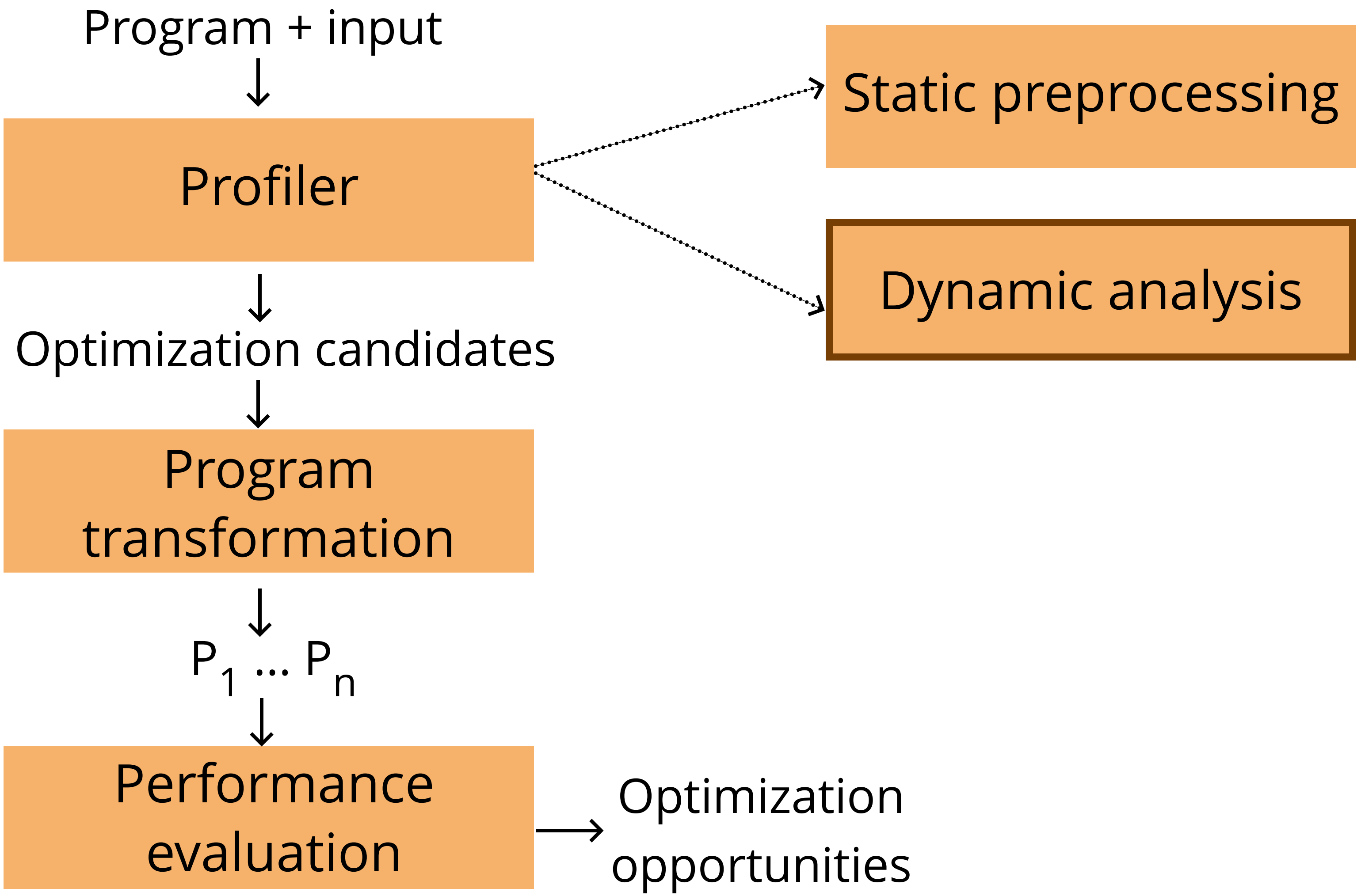
expensiveAndUnlikely() && cheapAndLikely()

- Analysis of all *conditions* in in logical expressions or switch statements
- Assessment of the computational cost
- Safe to apply and beneficial optimizations

DecisionProf: An Analysis for Optimizing Orders of Evaluations



DecisionProf: An Analysis for Optimizing Orders of Evaluations

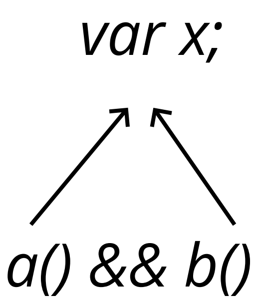


Dynamic Analysis

- Collecting runtime data:
  - Cost - number of branching point
  - Value - true/false
- Assessing the optimal order

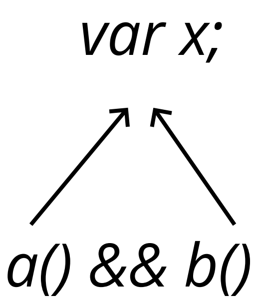
Dynamic Analysis

- Collecting runtime data:
  - Cost - number of branching point
  - Value - true/false
- Assessing the optimal order



Dynamic Analysis

- Collecting runtime data:
  - Cost - number of branching point
  - Value - true/false
- Assessing the optimal order



a() && b()

cost	value
c_{a1}	v_{a1}
c_{a2}	v_{a2}
c_{a3}	v_{a3}

cost	value
c_{b1}	v_{b1}
c_{b2}	v_{b2}
c_{b3}	v_{b3}

Dynamic Analysis: Example

_.isNumber(input) && isNaN(input)

	cost	value
Execution 1	3	true
Execution 2	2	true
Execution 3	4	true

cost	value
1	false
1	false
1	true

Dynamic Analysis: Example

_.isNumber(input) && isNaN(input)

	cost	value
	cost	**value**
Execution 1	3	true
Execution 2	2	true
Execution 3	4	true

cost	value
1	false
1	false
1	true

Overall cost = 12

Dynamic Analysis: Example

_.isNumber(input) && isNaN(input)

	cost	value	cost	value
Execution 1	3	true	1	false
Execution 2	2	true	1	false
Execution 3	4	true	1	true

Overall cost = 12

Dynamic Analysis: Example

isNaN(input) && _.isNumber(input)[1]

	Cost	Value
Execution 1	1	false
Execution 2	1	false
Execution 3	1	true

	Cost	Value
Execution 1	3	true
Execution 2	2	true
Execution 3	4	true

Dynamic Analysis: Example

isNaN(input) && _.isNumber(input)[1]

	isNaN(input): Cost	isNaN(input): Value	_.isNumber(input): Cost	_.isNumber(input): Value
	Cost	**Value**	**Cost**	**Value**
Execution 1	1	false	3	true
Execution 2	1	false	2	true
Execution 3	1	true	4	true

Overall cost = 7

1 See pull request #2496 of Underscore.js

UNDERSCORE.JS

Pruning Non-Commutative Conditions

Non-commutative conditions: change program semantics

Two approaches:

Static: known patterns

```
if (a && a.x) {...}

y = x || "abc"
```

Dynamic: write to the same memory location

```
var x = 0;
function a(){
  x++;
}

if (a() && x) {...}
```

Performance Evaluation

- Program transformation for each optimization candidate
- Methodology by Georges et al. [1]

Original program + input

Optimized program + input

VM instances

Warm up

Measure

t-test

[1] Georges et al. *Statistically rigorous Java performance evaluation* (OOPSLA 07)

DecisonProf: Evaluation

- Subject programs:
  - 9 JavaScript libraries and test suites
  - 34 benchmarks from JetStream suite
- Results:
  - 23 opportunities across libraries
  - 29 opportunities across benchmarks
  - Performance improvements: **2.5% - 59%** (function level) and **2.5% - 6.5%** (application level)

Examples of Reordering Opportunities

Cheerio library:

```
//code before
isTag (elem) && elems.indexOf(elem) === -1

//code after
elems.indexOf(elem) === -1 && isTag (elem)
```

tests: 26%, 34%

Gbemu benchmark:

```
//code before
numberType != "float32" && GameBoyWindow.opera
           && this.checkForOperaMathBug ()

//code after
GameBoyWindow.opera && numberType != "float32"
                    && this.checkForOperaMathBug ()
```

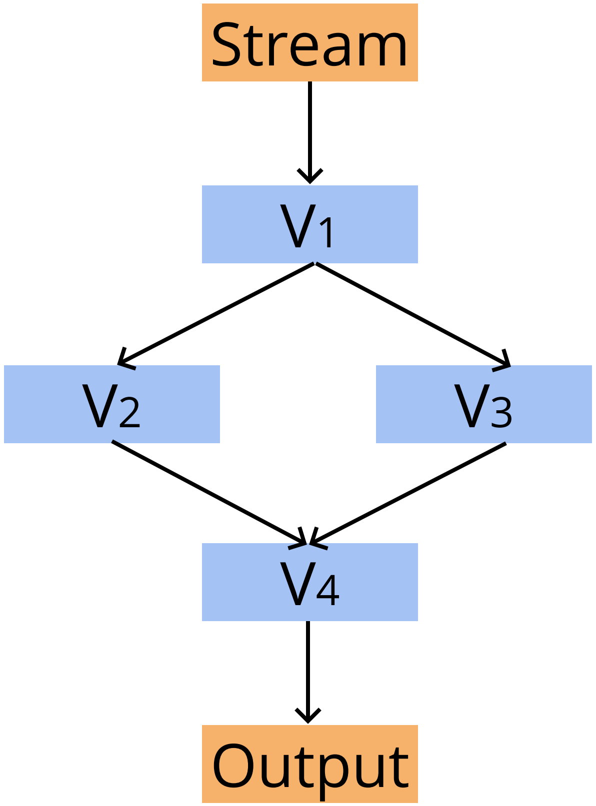
application: 5.8%

DecisionProf: Summary

- The first profiler to detect inefficient orders of evaluations
- Simple and easy to exploit optimizations
- Suggests program refactorings
- Guaranteed improvements for given inputs

Cross-language Optimizations in Big Data Systems

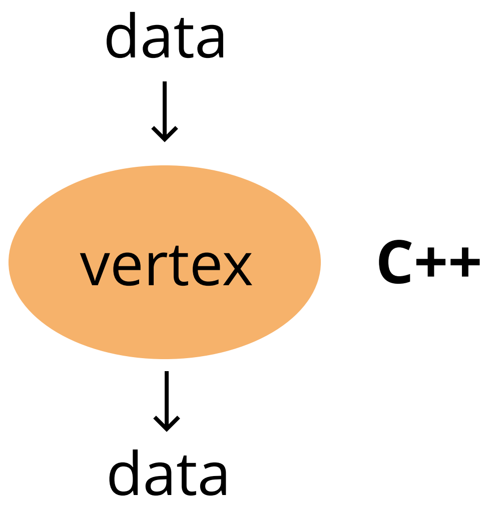
- **SCOPE** - **S**tructured **C**omputations **O**ptimized for **P**arallel **E**xecutions
- Relational (SQL) + non-relational (C#)
- SCOPE job is DAG where:
  - Vertices - processes
  - Edges - data flows



Performance Problem in SCOPE

- Cross runtime interaction
- Intrinsics vs non-intrinsics

```
data =
    SELECT *
    FROM inputStream;
```

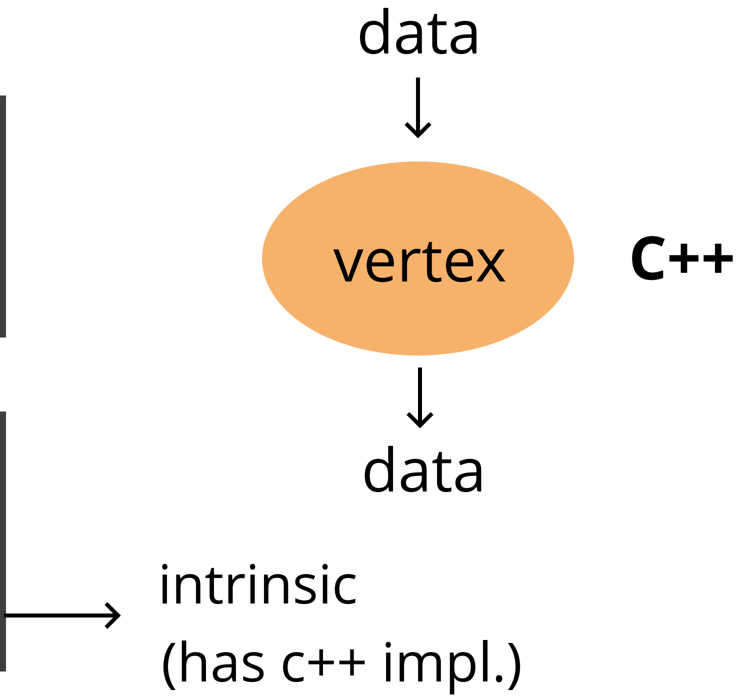


Performance Problem in SCOPE

- Cross runtime interaction
- **Intrinsics** vs non-intrinsics

```
data =
    SELECT *
    FROM inputStream;
```

```
data =
    SELECT *
    FROM inputStream
    WHERE !String.IsNullOrEmpty(A);
```

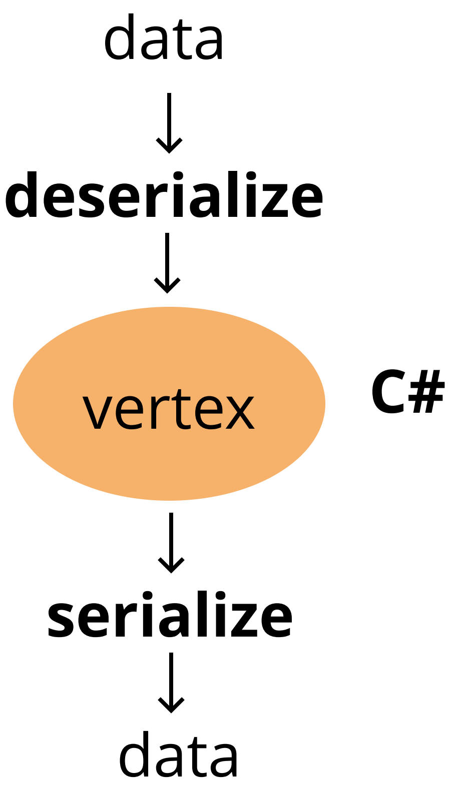


Performance Problem in SCOPE

- Cross runtime interaction
- Intrinsics vs **non-intrinsics**

```
data =
    SELECT *
    FROM inputStream
    WHERE A.Equals('ColumnA');
```

→ non-intrinsic



Method Inlining: Example

- Replacing function call with the body of a function

```
data =
    SELECT *
    FROM inputStream
    WHERE filter(JobID);

#CS

bool filter (string s) {
     return (!string.IsNullOrEmpty(s)
            && s.StartsWith("07"));
}

#ENDCS
```

C#

Method Inlining: Example

- Replacing function call with the body of a function

```
data =
    SELECT *
    FROM inputStream
    WHERE !string.IsNullOrEmpty(JobID)
           && JobID.StartsWith("07");
```

C++

Method Inlining: Example

- Replacing function call with the body of a function

```
data =
    SELECT *
    FROM inputStream
    WHERE !string.IsNullOrEmpty(JobID)
           && JobID.StartsWith("07");
```

C++

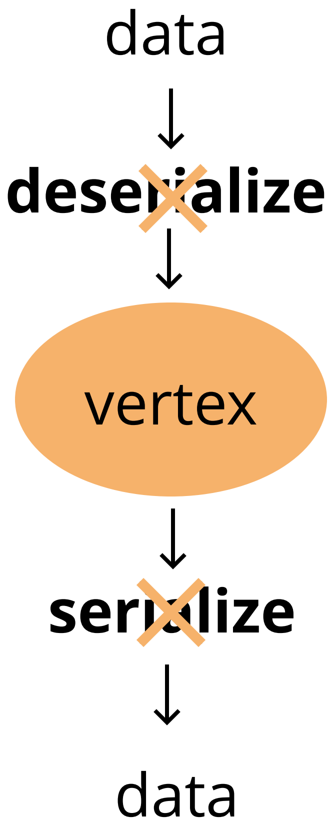
~ 4X faster

Static Analysis

Goal: C# to C++ translation for a vertex

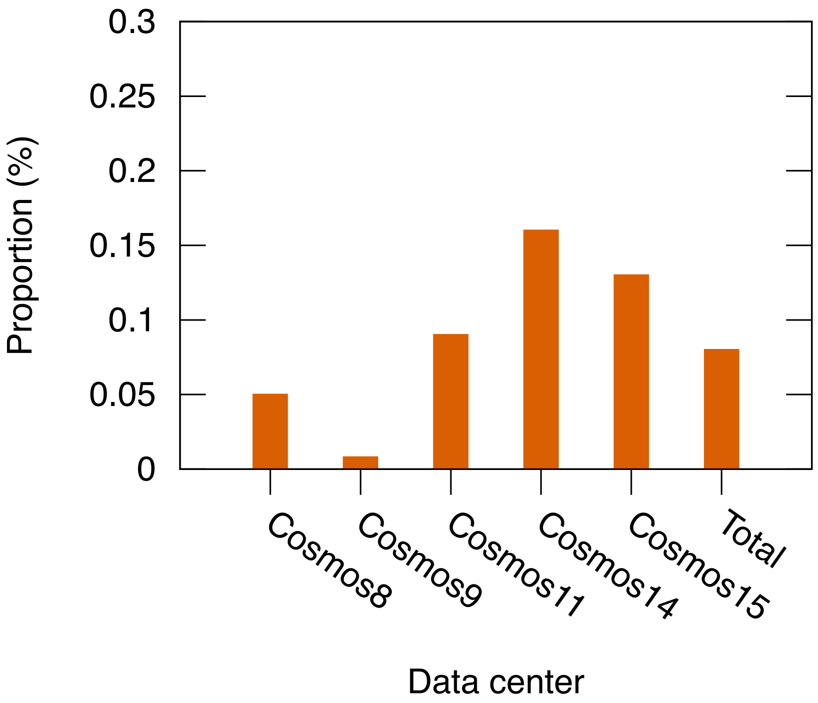
Inlineable methods:

- Only calls to intrinsics
- No loops
- No try-catch blocks
- No new and cast operations
- No arguments passed by reference



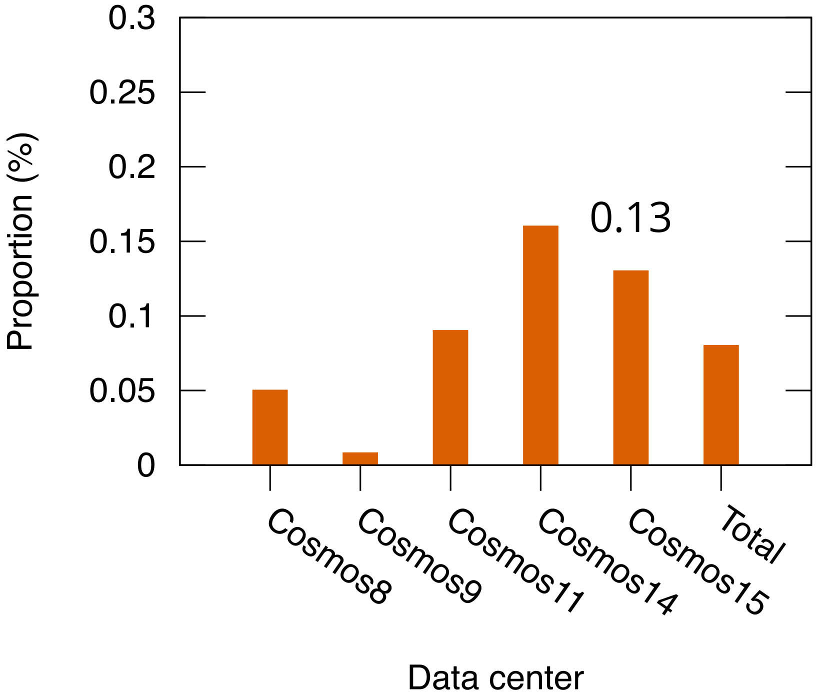
Static Analysis: Evaluation

Optimizable vertices:



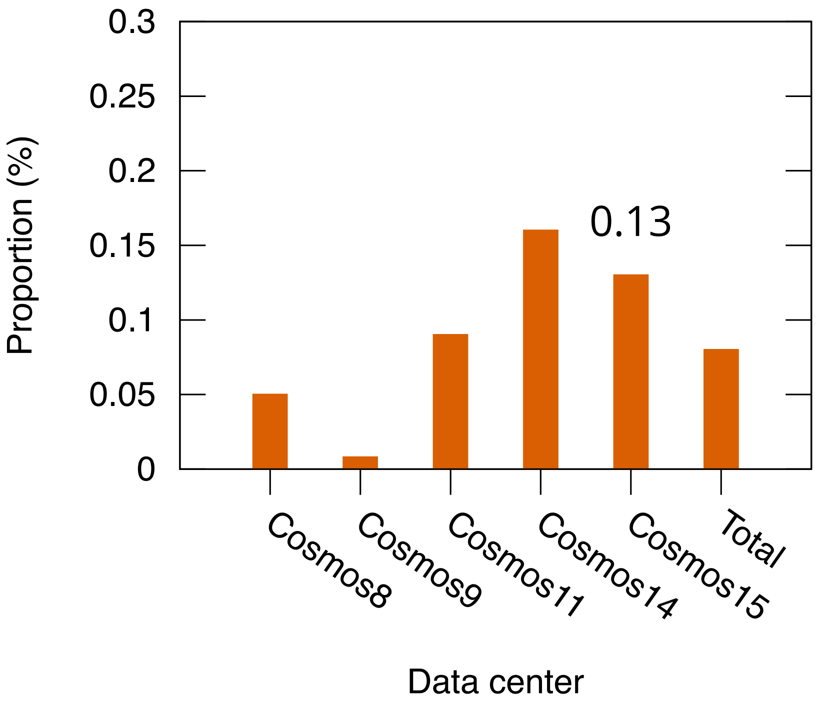
Static Analysis: Evaluation

Optimizable vertices:



Static Analysis: Evaluation

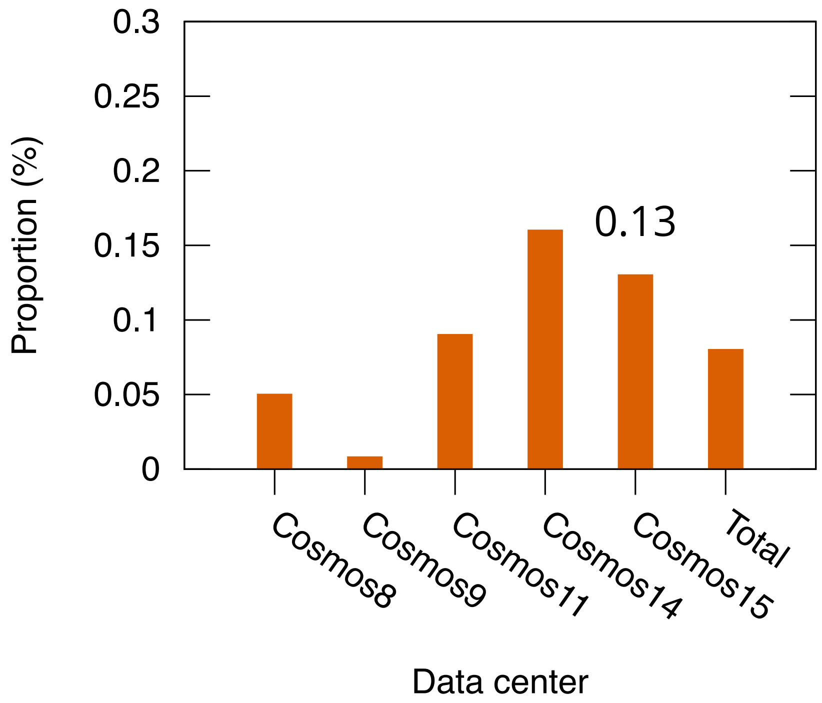
Optimizable vertices:



0.13% ~ 40,000 hours

Static Analysis: Evaluation

Optimizable vertices:



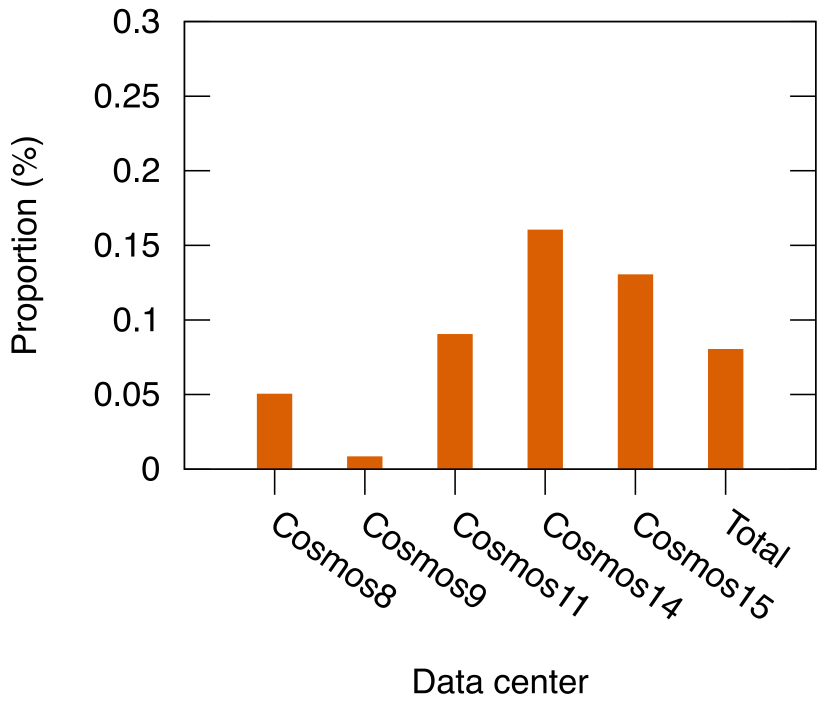
Case studies:

Job	CPU Time	Throughput
A	23%	30%
B	no change	no change
C	25%	38%
E	4.7%	5%
F	no change	115%

0.13% ~ 40,000 hours

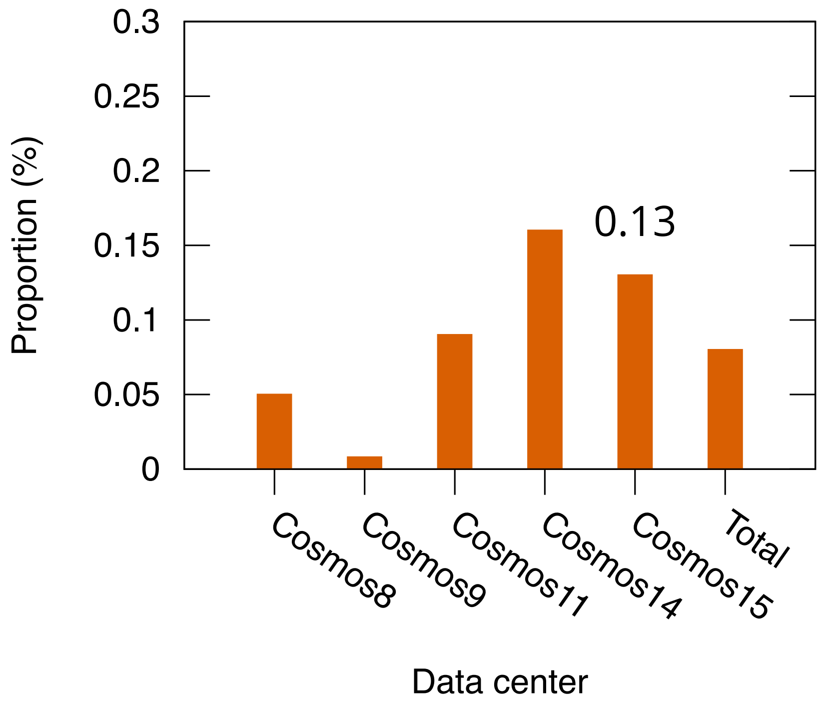
Static Analysis: Evaluation

Optimizable vertices:



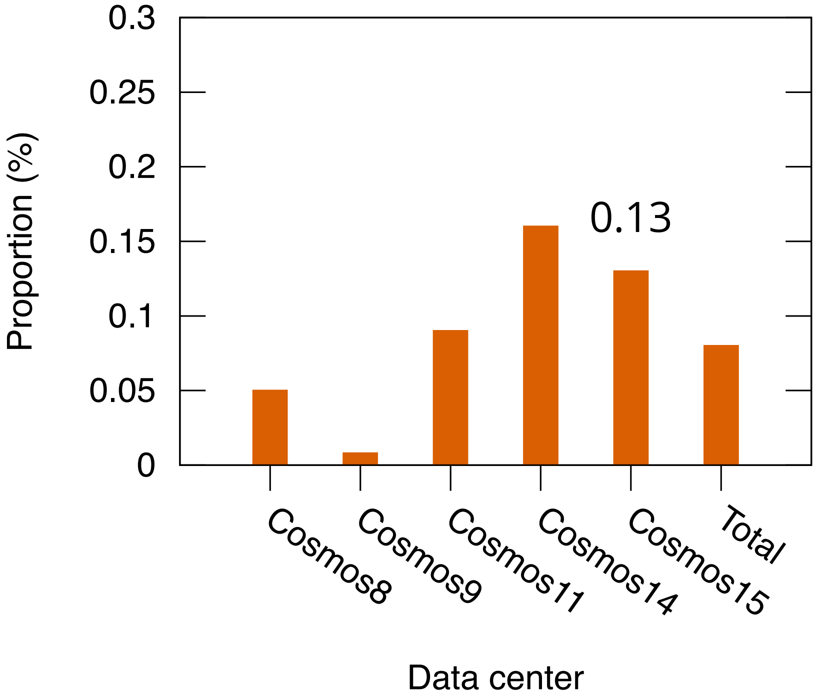
Static Analysis: Evaluation

Optimizable vertices:



Static Analysis: Evaluation

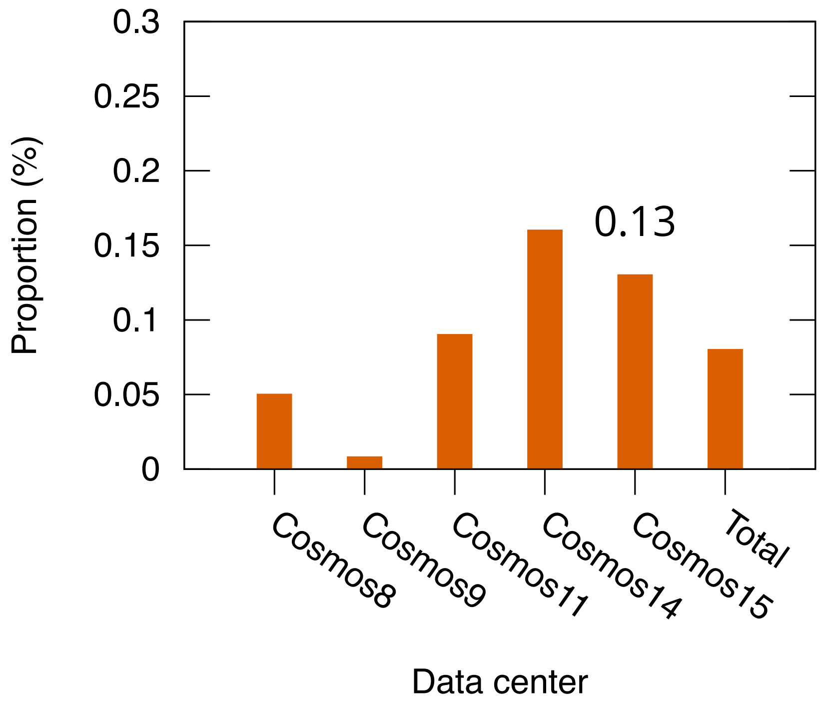
Optimizable vertices:



0.13% ~ 40,000 hours

Static Analysis: Evaluation

Optimizable vertices:



Case studies:

Job	CPU Time	Throughput
A	23%	30%
B	no change	no change
C	**25%**	**38%**
E	4.7%	5%
F	no change	115%

0.13% ~ 40,000 hours

Case Study in SCOPE: Example

```
//optimization opportunity

SELECT url_id, url, t_url_id,
       CommonMethod.GetTargetUrl(url,t_url) AS t_url
FROM SSTREAM @VLPMapSS;

//optimized code

SELECT url_id, url, t_url_id,
       string.isNullOrEmpty(t_url)?url:t_url AS t_url
FROM SSTREAM @VLPMapSS;
```

Vertex level improvement: 42%

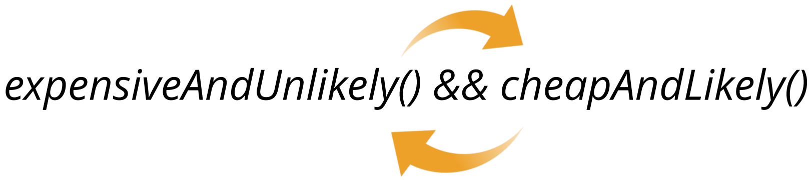
Job level improvement: 25%

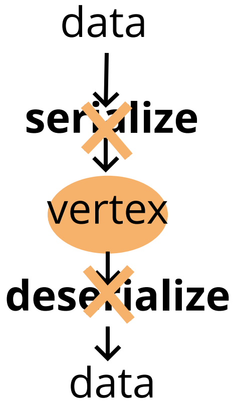
Cross-language Optimizations in Big Data Systems: Summary

- Static analysis for method inlining opportunities
- Only optimizations that reduce cross-runtime interactions
- Large scale evaluation

Conclusions

- Actionable performance analyses
- Easy to exploit classes of optimizations
- Future work: automatically inferring optimization patterns across and within *different domains*





Huawei Dresden Research Center

Research topics:

- Design and development of embedded systems
- Program analysis and formal verification
- Software testing: fuzzing and automated test case generation

HUAWEI

Huawei Dresden Research Center

Research topics:

- Design and development of embedded systems
- Program analysis and formal verification
- Software testing: fuzzing and automated test case generation

We are hiring!

HUAWEI

Questions?

DecisionProf: Static Preprocessing

```
startDecision;

startCheck: a(); endCheck();
startCheck: b(); endCheck();
startCheck: c(); endCheck();

endDecision;


if (a() && b() && c()) {
    ...
```

- Hoists leaf expressions
- Beginning and end of each decision
- Beginning and end of each checks

DecisionProf: Safe Check Evaluation

Collect and undo all writes to variables and object properties that may affect code after check evaluation

```
var x = 0;

function a () {
  x++;
  var y=1;
  ......
 }

startCheck: a();
startCheck: b();

//reset all side effects
if (a () && b()) ...
```

DecisionProf: Safe Check Evaluation

Collect and undo all writes to variables and object properties that may affect code after check evaluation

```
var x = 0;

function a () {
  x++;
  var y=1;
  .......
 }

startCheck: a();
startCheck: b();

//reset all side effects
if (a () && b()) ...
```

← *write to x affects program state*

DecisionProf: Safe Check Evaluation

Collect and undo all writes to variables and object properties that may affect code after check evaluation

```
var x = 0;

function a () {
  x++;
  var y=1;
  ......
 }

startCheck: a();
startCheck: b();

//reset all side effects
if (a () && b()) ...
```

write to x affects program state

program state is changed outside normal execution

DecisionProf: Safe Check Evaluation

Collect and undo all writes to variables and object properties that may affect code after check evaluation

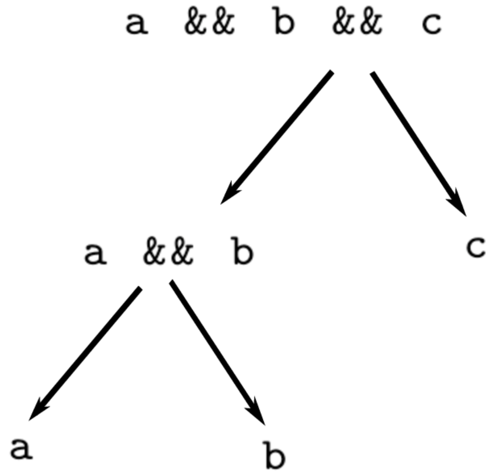
```
var x = 0;

function a () {
  x++;
  var y=1;
  .......
 }

startCheck: a();
startCheck: b();

//reset all side effects
if (a () && b()) ...
```

← *write to x affects program state*

← *program state is changed outside normal execution*

← *dynamically execute x = 0;*

DecisionProf: Finding Optimal Order



③ (a && b && c)

Cost	Value
5	true
1	false
1	false

② (a && b)

Cost	Value
4	true
1	false
1	false

① (c)

Cost	Value
1	true
1	true
1	true

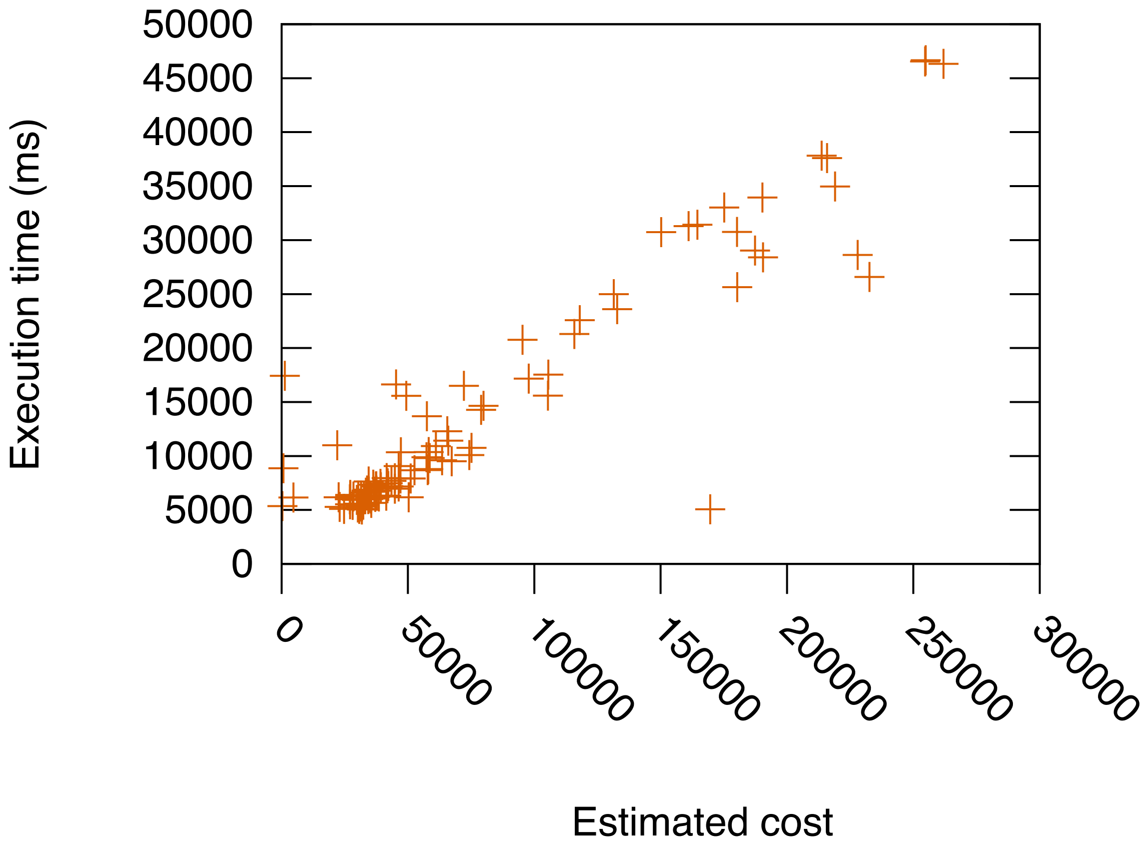
① (a)

Cost	Value
3	true
3	false
3	false

① (b)

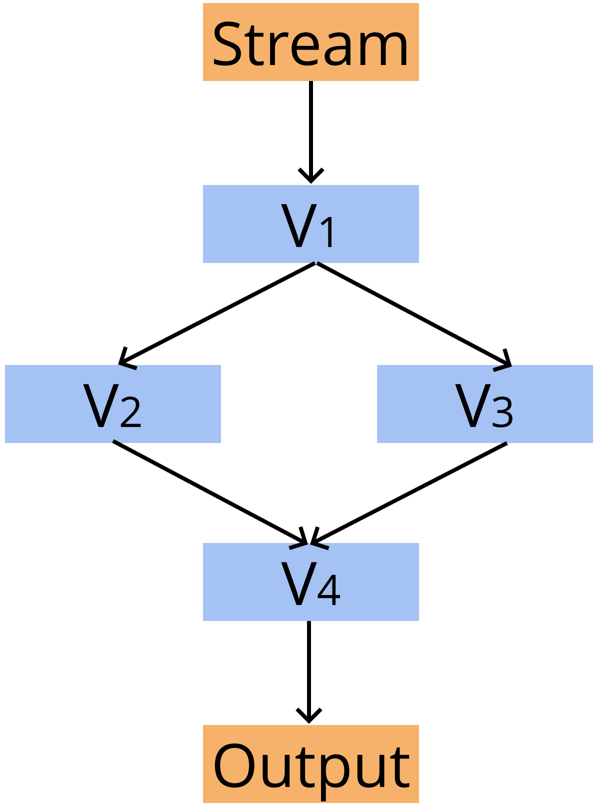
Cost	Value
1	true
1	false
1	false

DecisionProf: Actual vs Estimated Cost

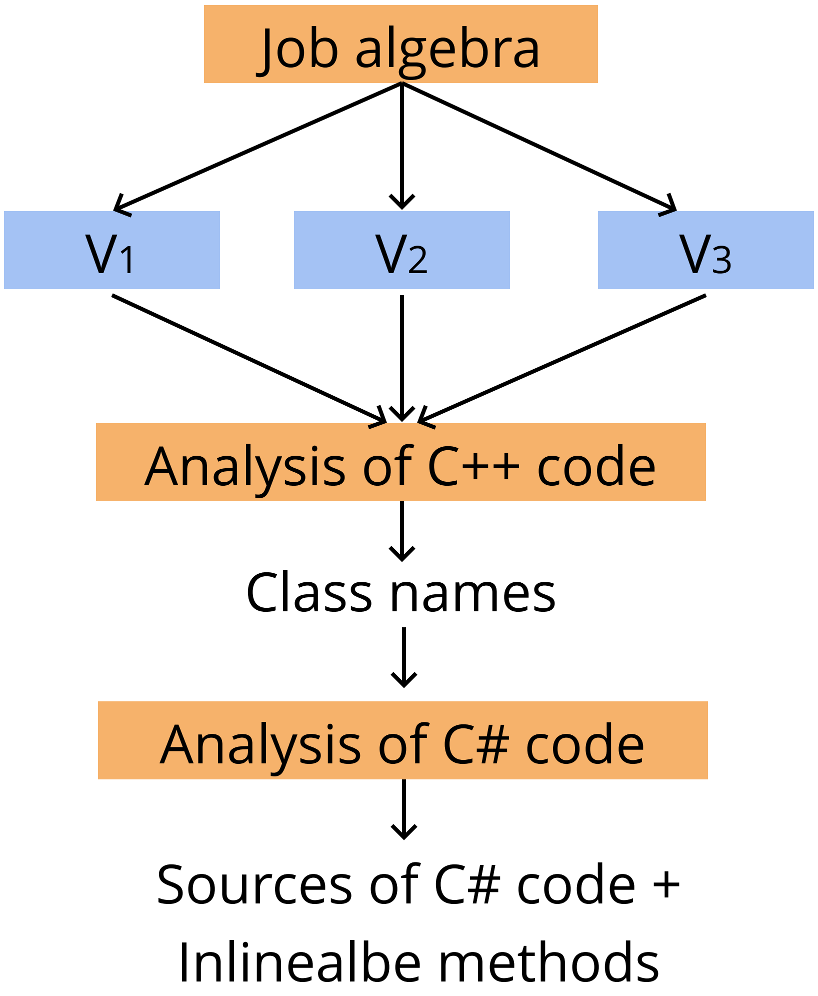


Job Artifacts

- Job Algebra
  - Vertices - processes
  - Edges - data flow
- Runtime statistics
- Script source code
- Generated C# and C++ code



Profiling Infrastructure



- .NET framework methods
- User-written methods
- Processors and reduces

Native vs Non-Native Time