IMPROVING COMPILER OPTIMIZATIONS BY EMPLOYING MACHINE LEARNING

Raphael Mosaner – Johannes Kepler University Linz

JOHANNES KEPLER UNIVERSITY LINZ

Compiler Heuristics

Metrics to decide which transformation / optimization to apply in what way

COMPILER HEURISTIC EXAMPLE

Compiler Heuristics

Metrics to decide which transformation / optimization to apply in what way

if(ΔcodeSize*weight$_{size}$ < Δperformance*weight$_{perf}$) → doTransformation()

COMPILER HEURISTIC EXAMPLE

Compiler Heuristics

Metrics to decide which transformation / optimization to apply in what way

$$\text{if}(\Delta\text{codeSize}*\overbrace{\mathbf{weight}_{\mathbf{size}}}^{\textbf{heuristic}} < \Delta\text{performance}*\overbrace{\mathbf{weight}_{\mathbf{perf}}}^{\textbf{heuristic}}) \rightarrow \text{doTransformation}()$$

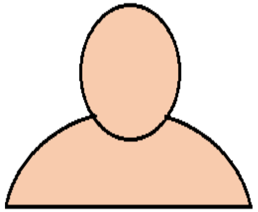
COMPILER HEURISTIC EXAMPLE

Compiler Heuristics

Metrics to decide which transformation / optimization to apply in what way

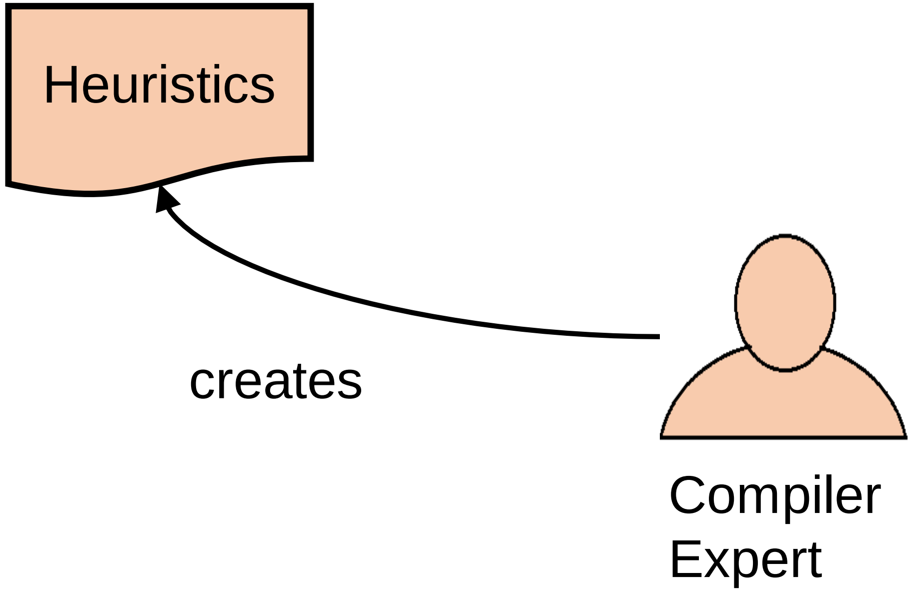
$$\text{if}(\underbrace{\Delta\textbf{codeSize*weight}_{\textbf{size}} \mathbf{<} \Delta\textbf{performance*weight}_{\textbf{perf}}}_{\textbf{heuristic}}) \rightarrow \text{doTransformation()}$$

COMPILER HEURISTICS STATE-OF-THE-ART

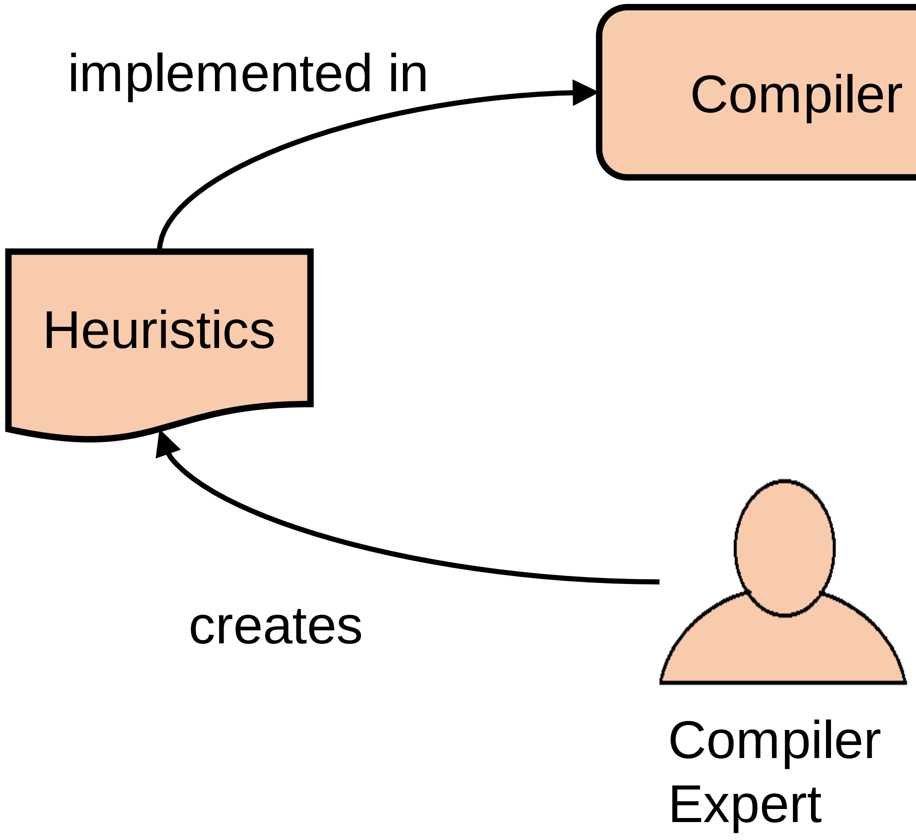


Compiler Expert

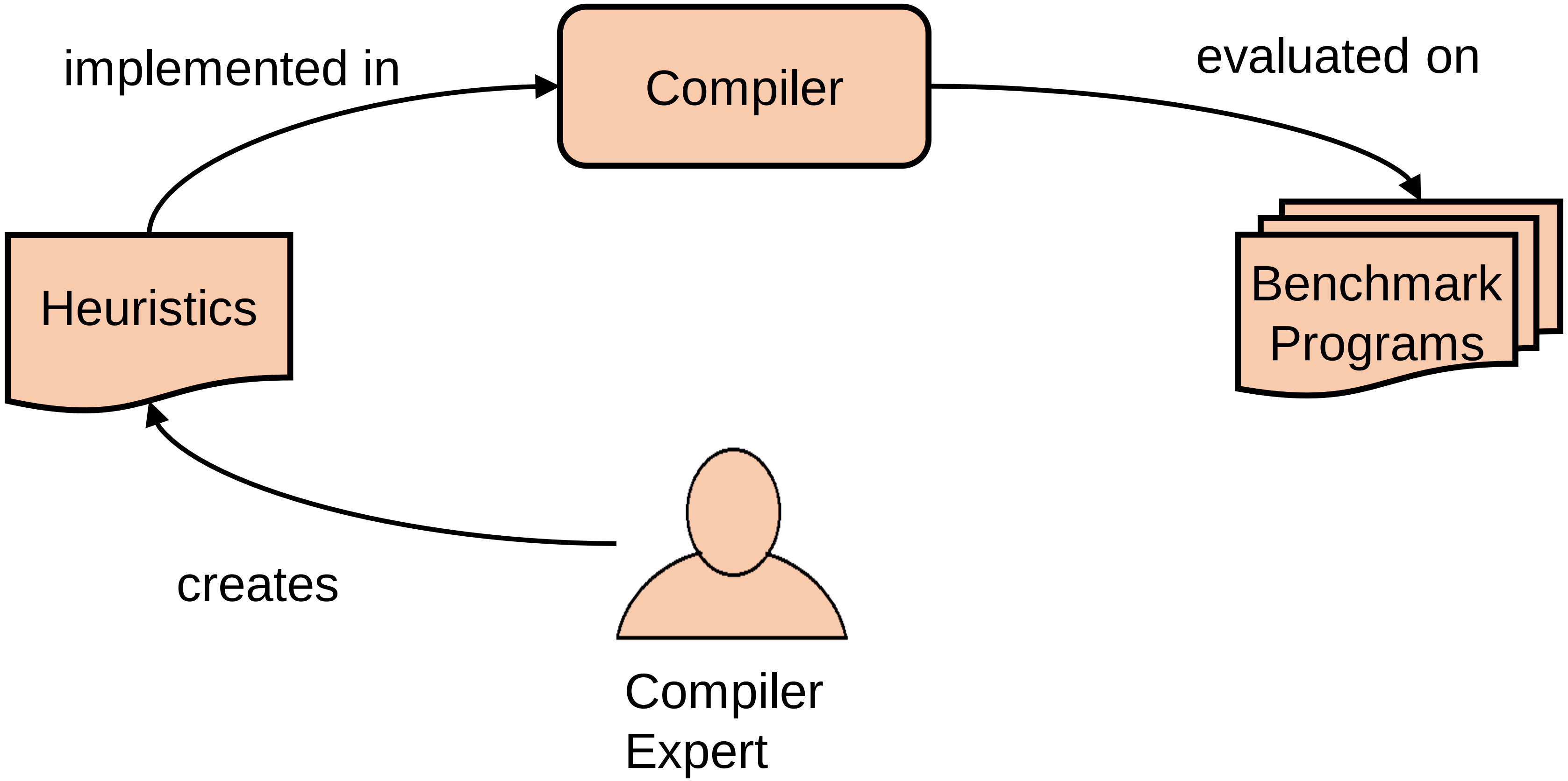
COMPILER HEURISTICS STATE-OF-THE-ART



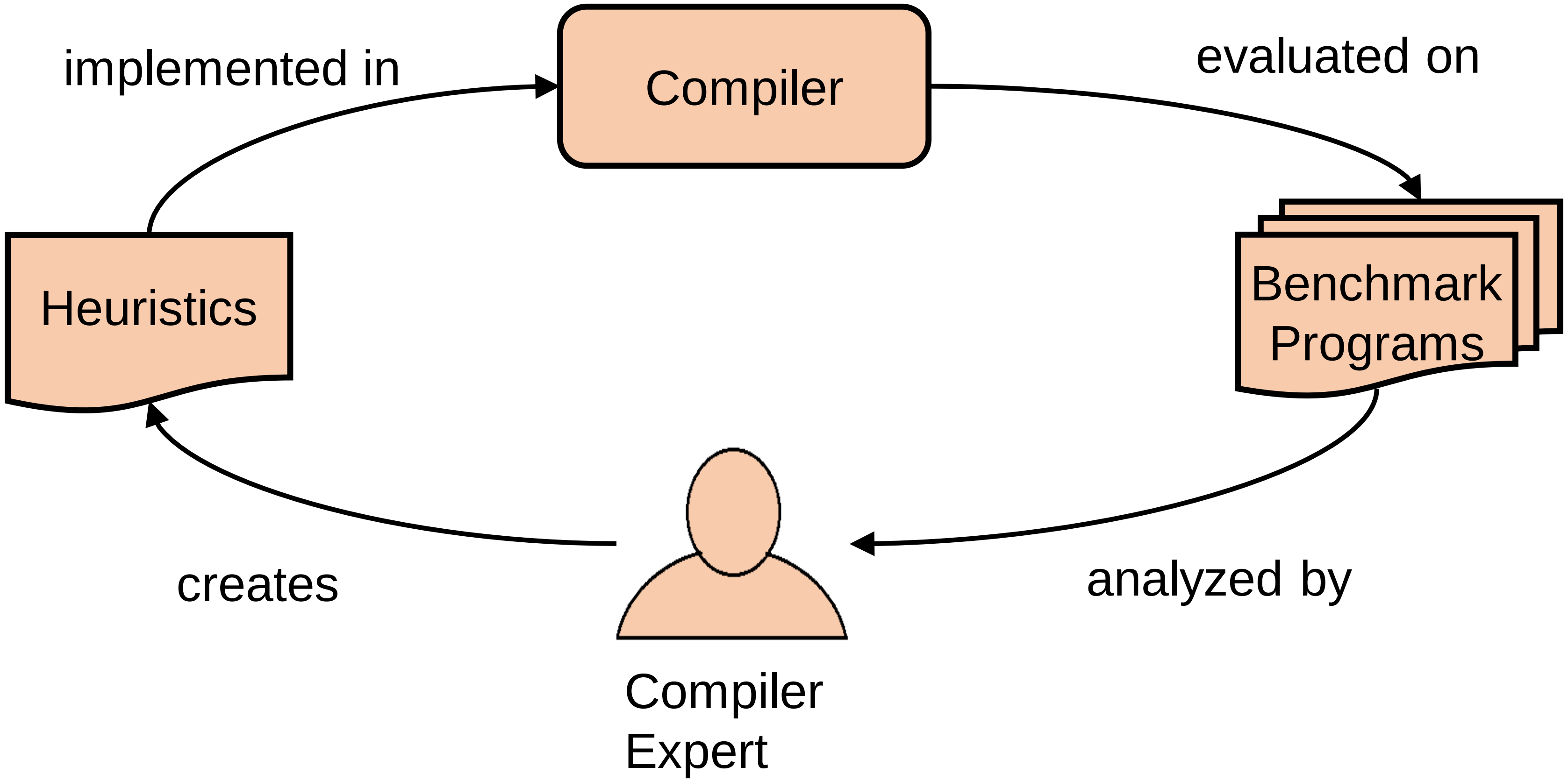
COMPILER HEURISTICS STATE-OF-THE-ART



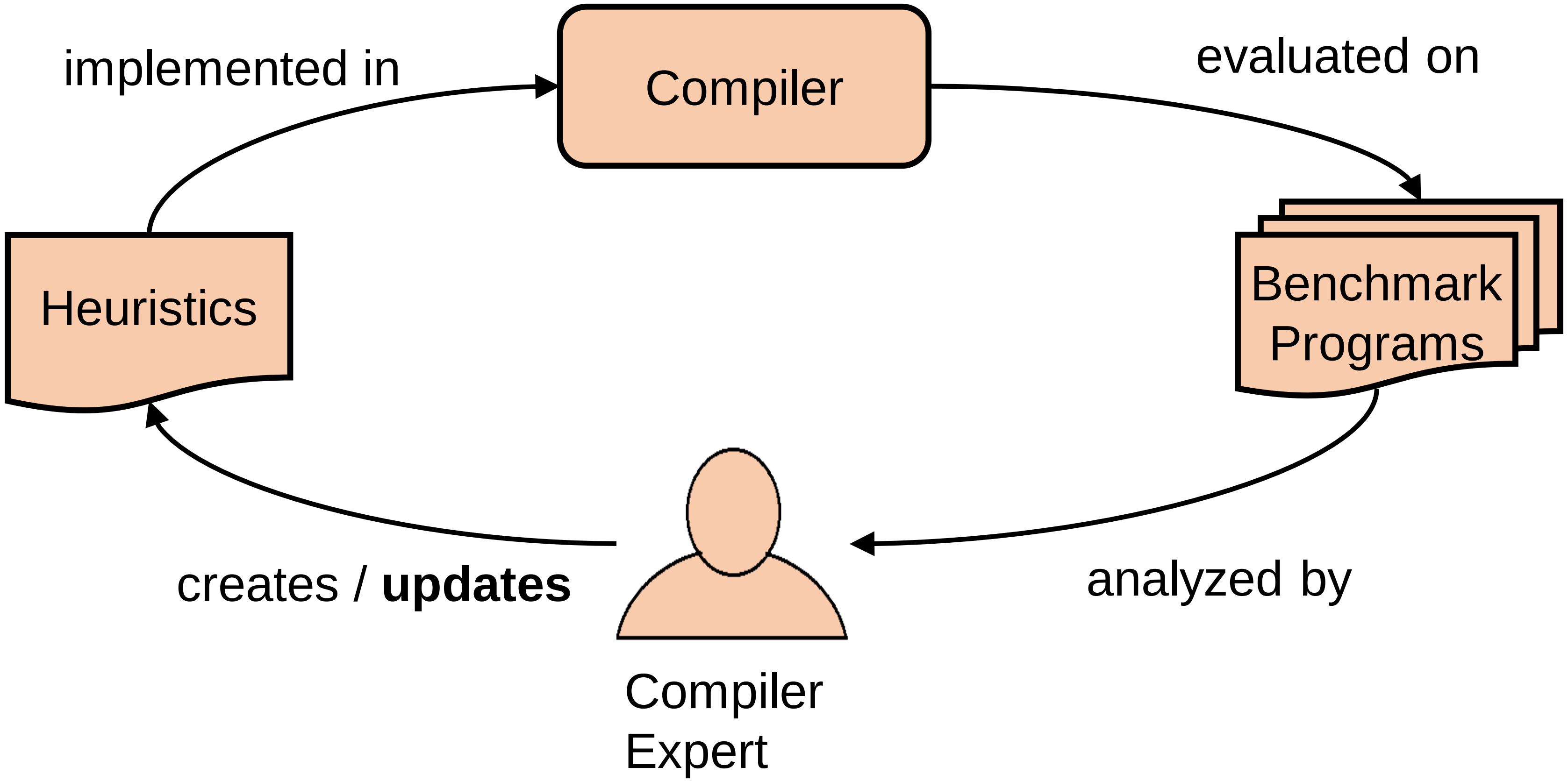
COMPILER HEURISTICS STATE-OF-THE-ART



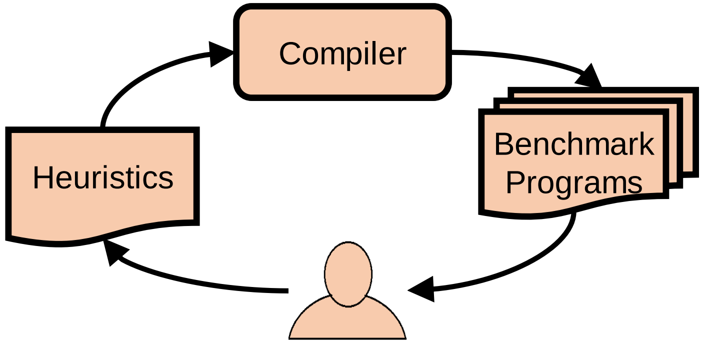
COMPILER HEURISTICS STATE-OF-THE-ART



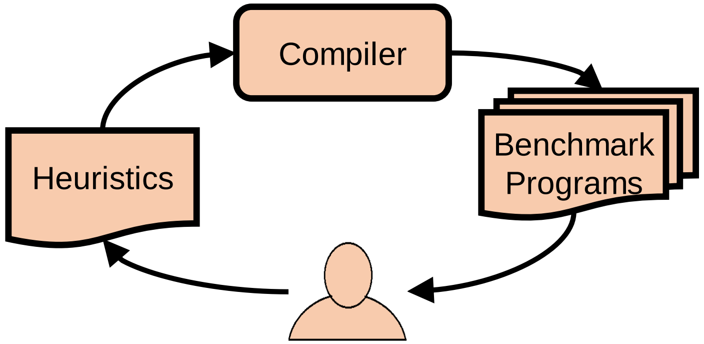
COMPILER HEURISTICS STATE-OF-THE-ART



HUMAN-CRAFTED HEURISTICS

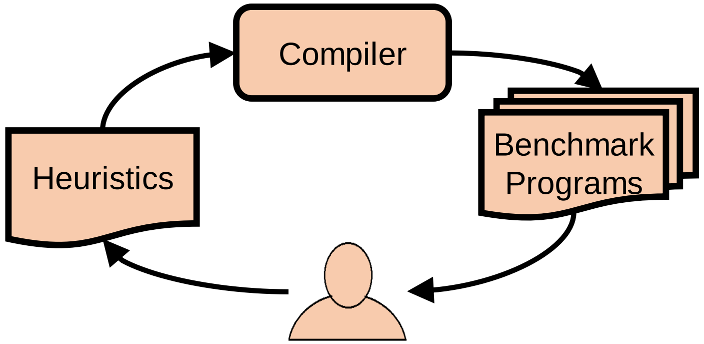


HUMAN-CRAFTED HEURISTICS



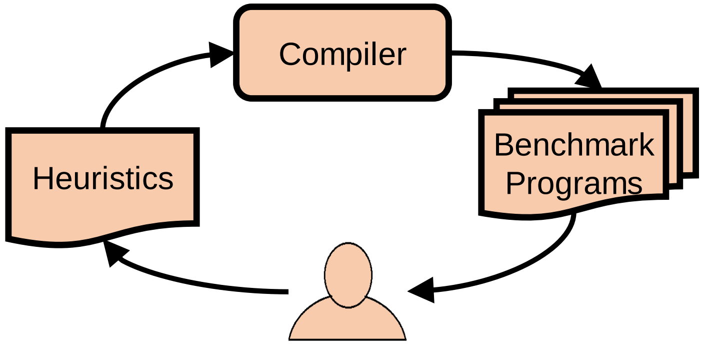
- Require domain expertise

HUMAN-CRAFTED HEURISTICS



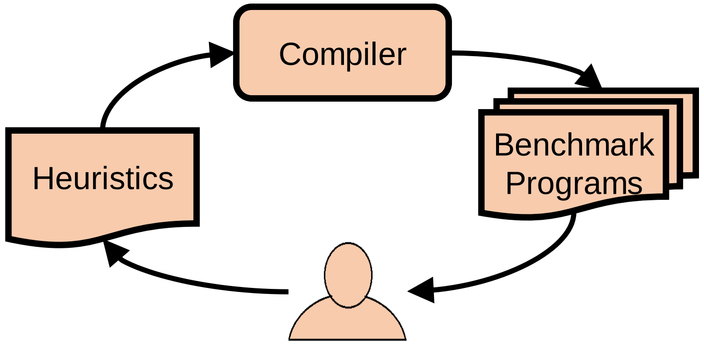
- Require domain expertise
- Abstracts „real“ world to few benchmark programs

HUMAN-CRAFTED HEURISTICS



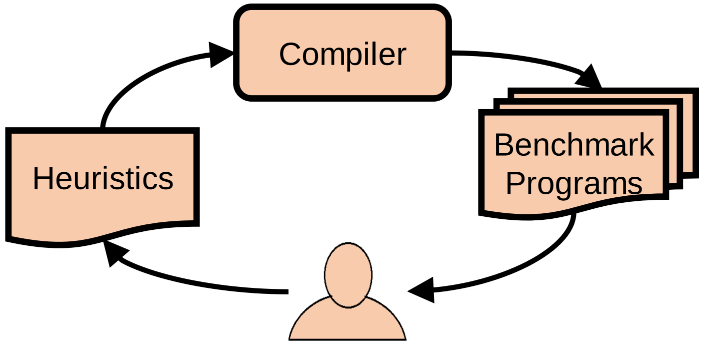
- Require domain expertise
- Abstracts „real“ world to few benchmark programs
- Often static / one-size-fits-all

HUMAN-CRAFTED HEURISTICS



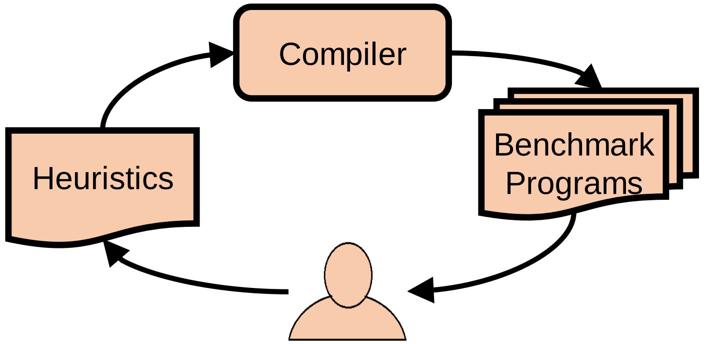
- Require domain expertise
- Abstracts „real“ world to few benchmark programs
- Often static / one-size-fits-all
- Manual maintainance and updates

HUMAN-CRAFTED HEURISTICS



- Require domain expertise
- Abstracts „real“ world to few benchmark programs
- Often static / one-size-fits-all
- Manual maintainance and updates
- Error prone

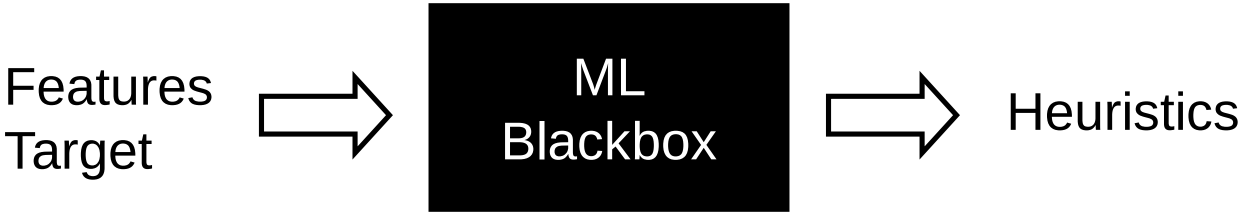
HUMAN-CRAFTED HEURISTICS



- Require domain expertise
- Abstracts „real“ world to few benchmark programs
- Often static / one-size-fits-all
- Manual maintainance and updates
- Error prone

experience driven

MACHINE LEARNED HEURISTICS STATE-OF-THE-ART



Feature

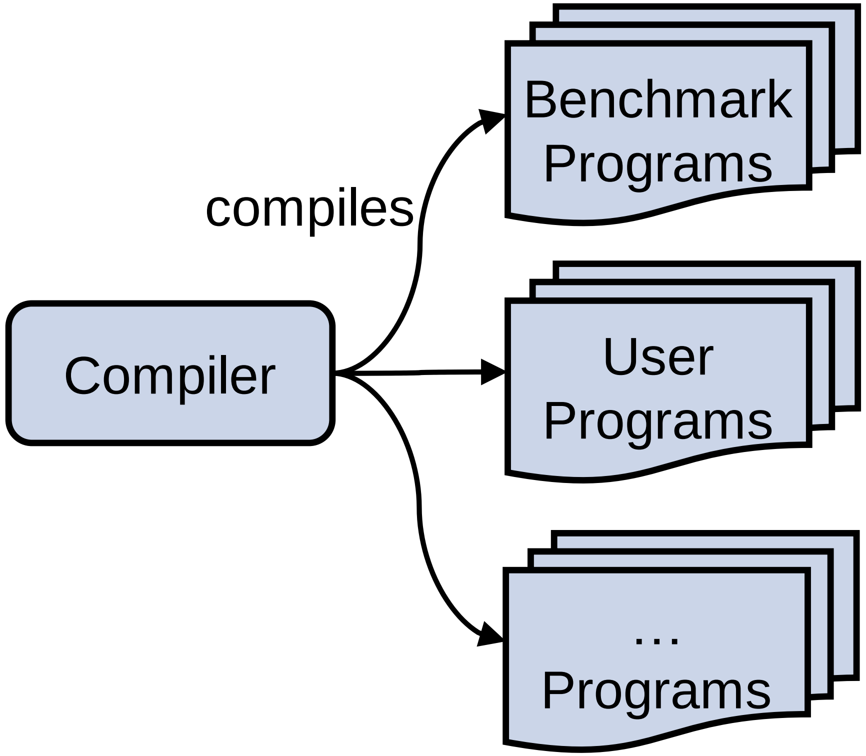
A feature is a measurable property of an object of interest.
(e.g., #branches, #memoryOperations)

Target

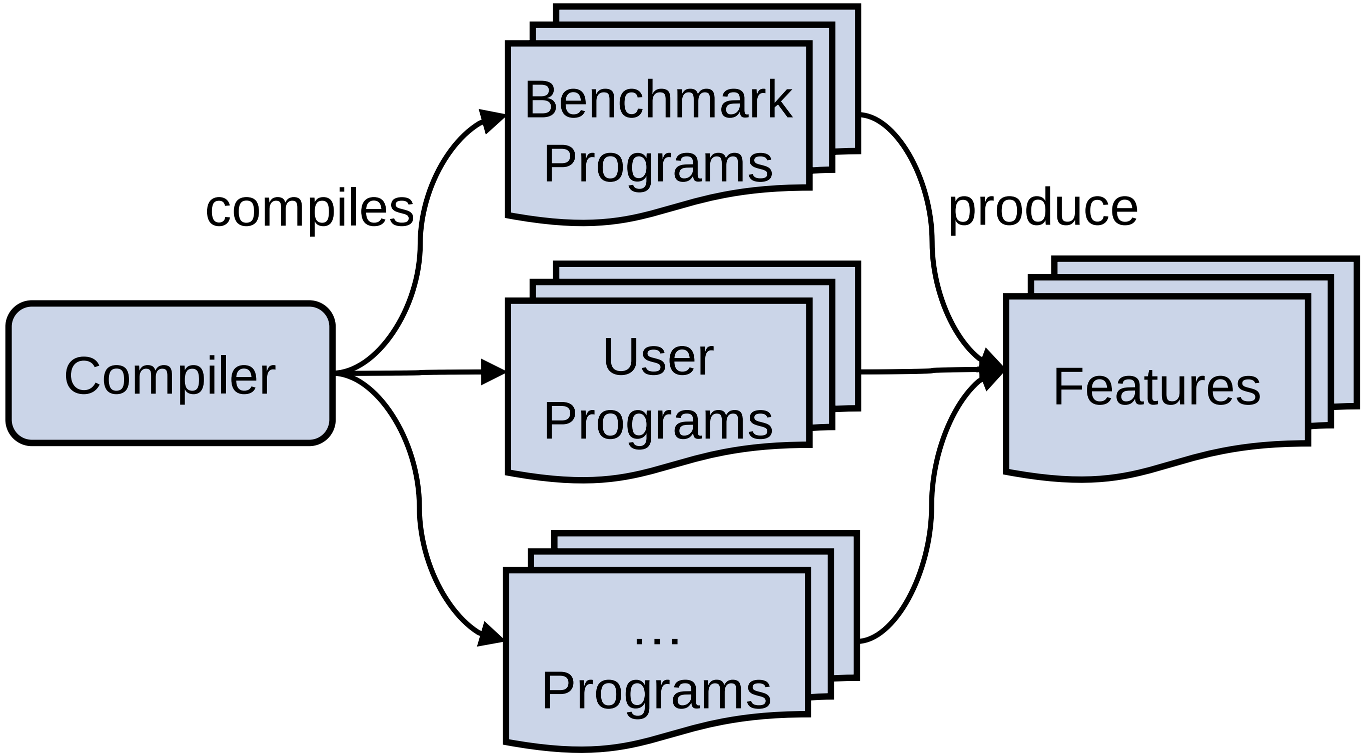
The target is the feature to be predicted.
(e.g., best optimization decision)

ML MODELS IN COMPILERS STATE-OF-THE-ART

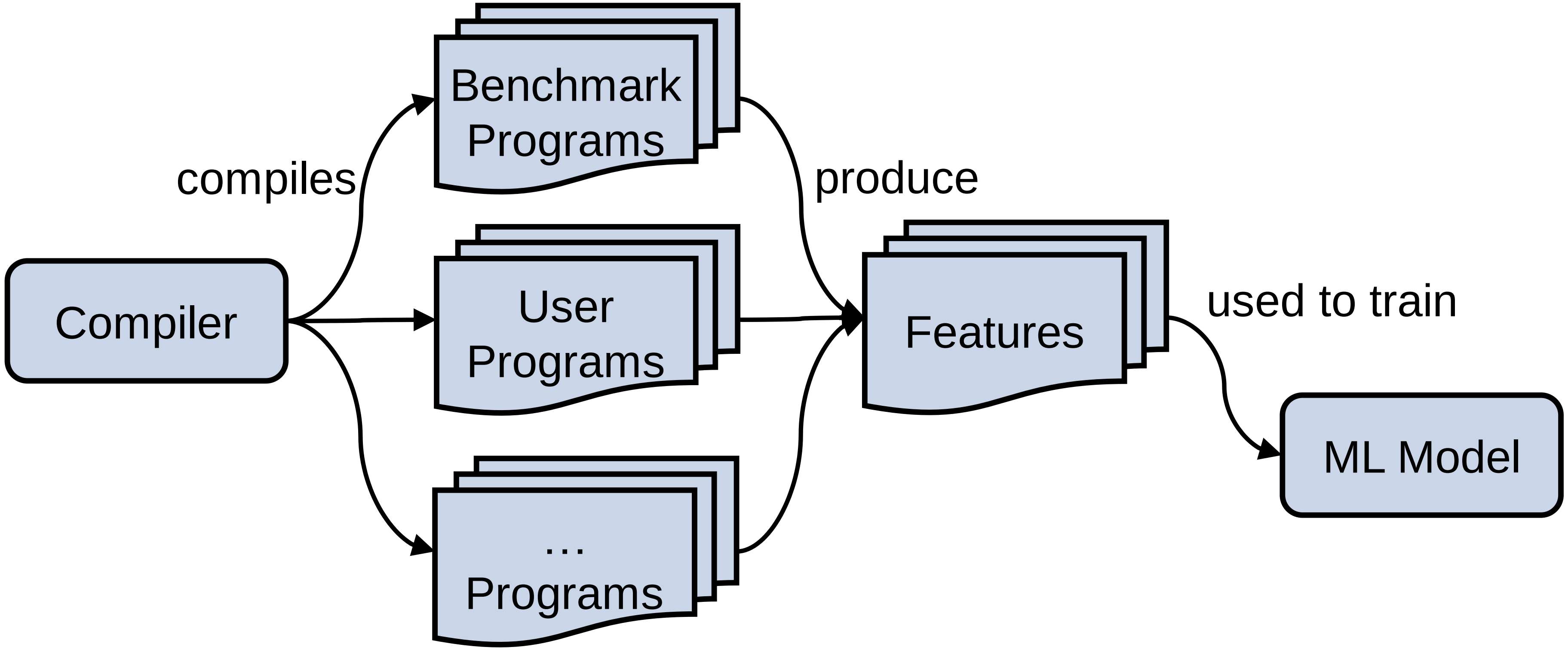
ML MODELS IN COMPILERS STATE-OF-THE-ART



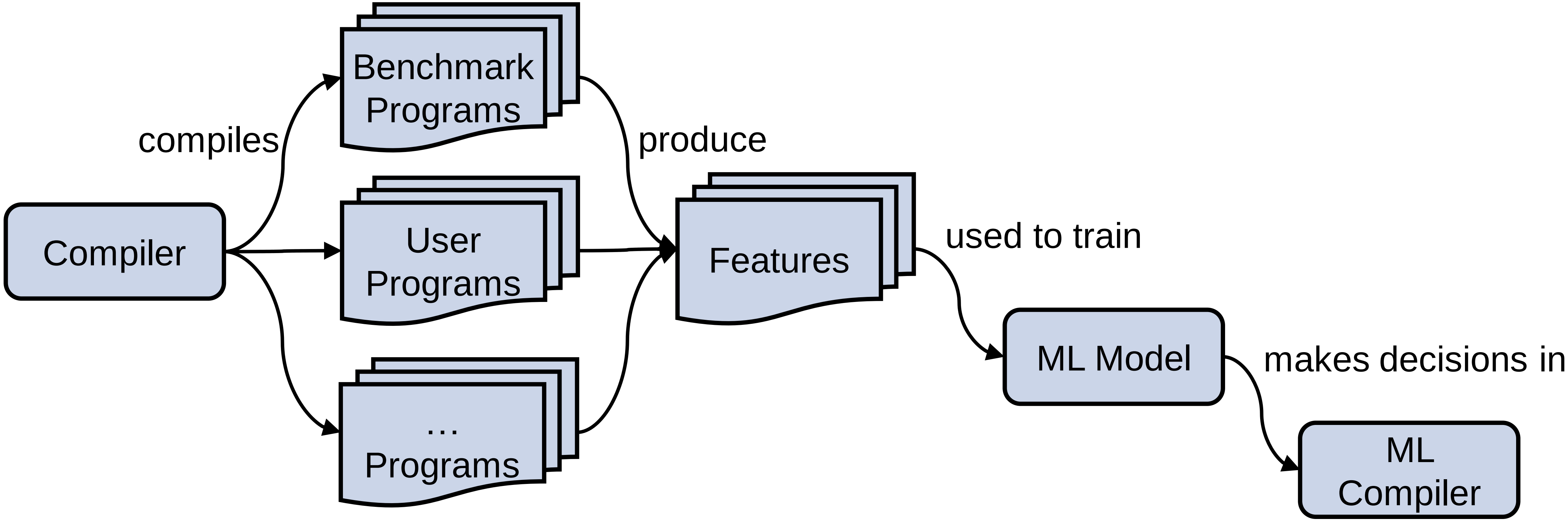
ML MODELS IN COMPILERS STATE-OF-THE-ART



ML MODELS IN COMPILERS STATE-OF-THE-ART



ML MODELS IN COMPILERS STATE-OF-THE-ART



Compiler

compiles

Benchmark Programs

User Programs

… Programs

produce

Features

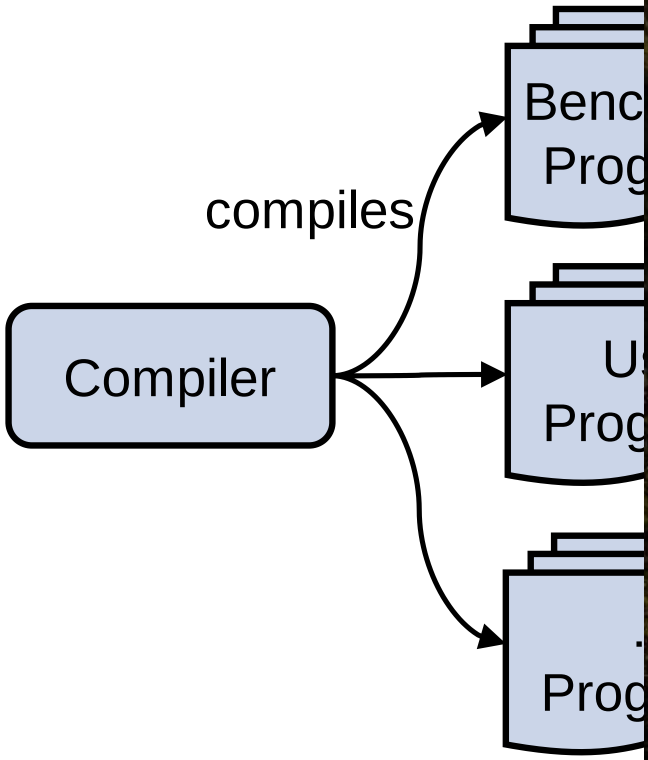
used to train

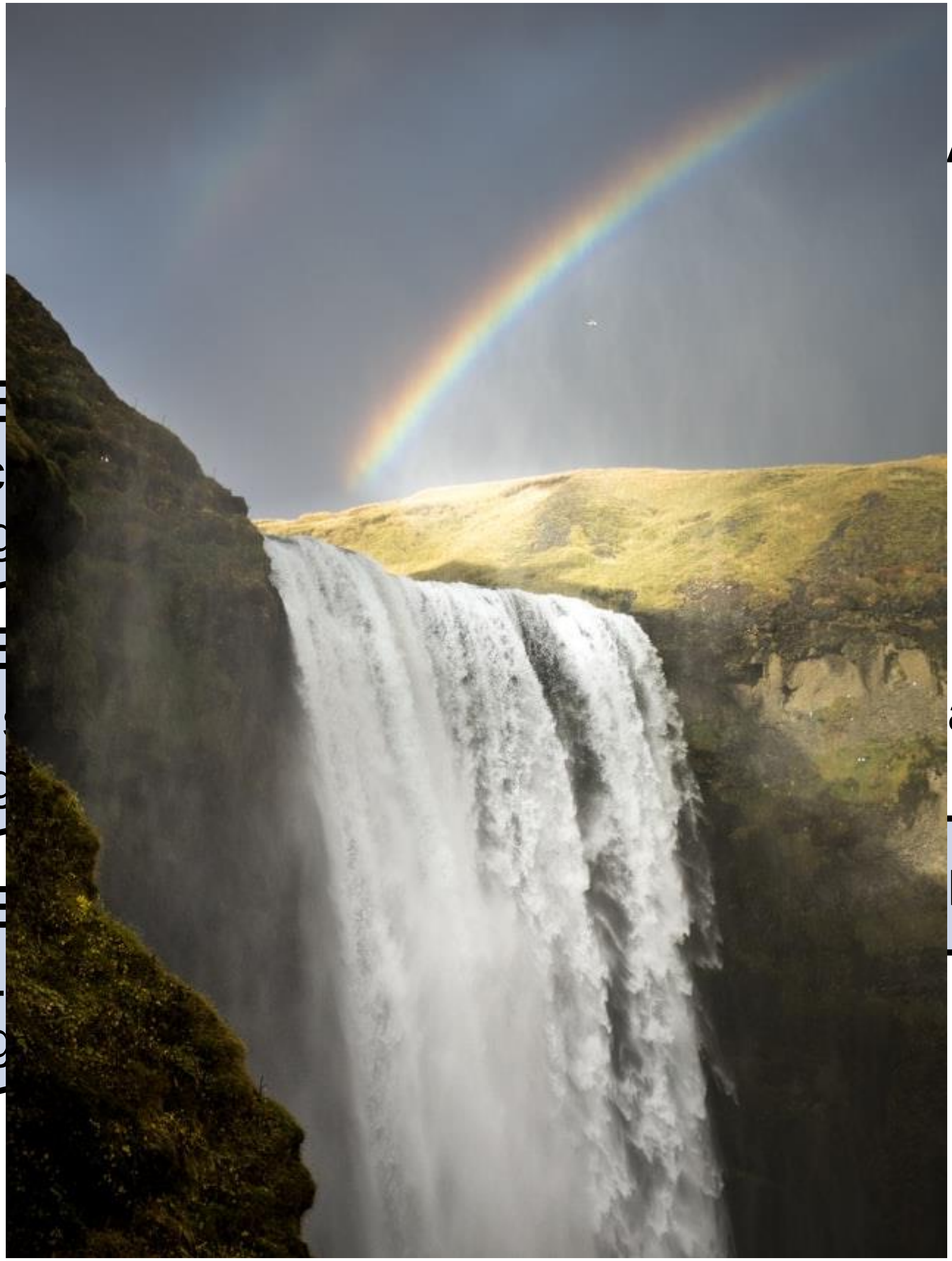
ML Model

makes decisions in

ML Compiler

ML MODELS I ART





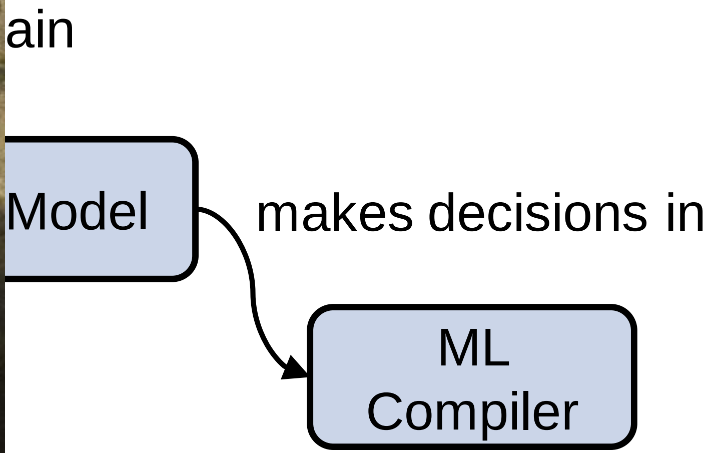
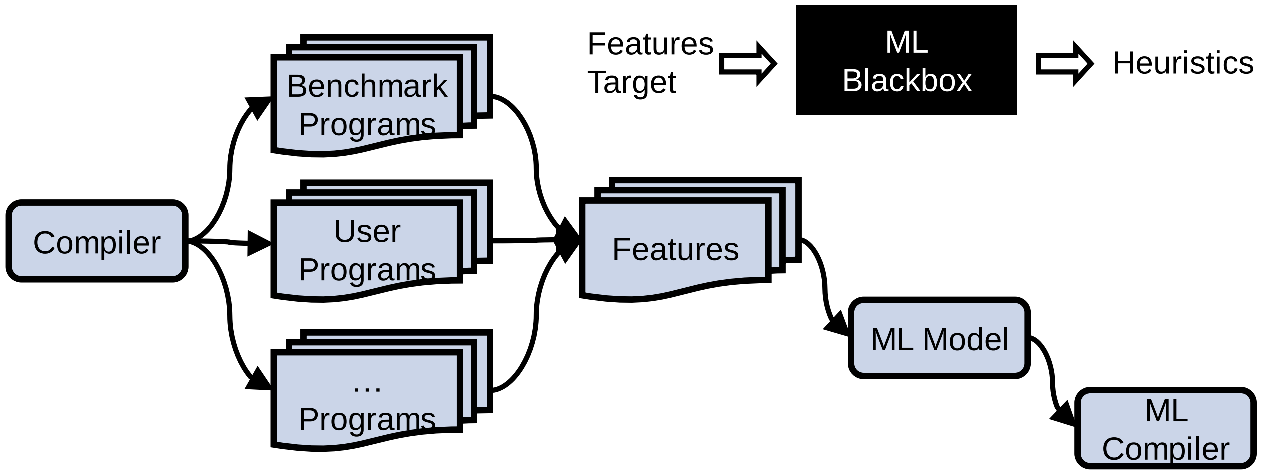
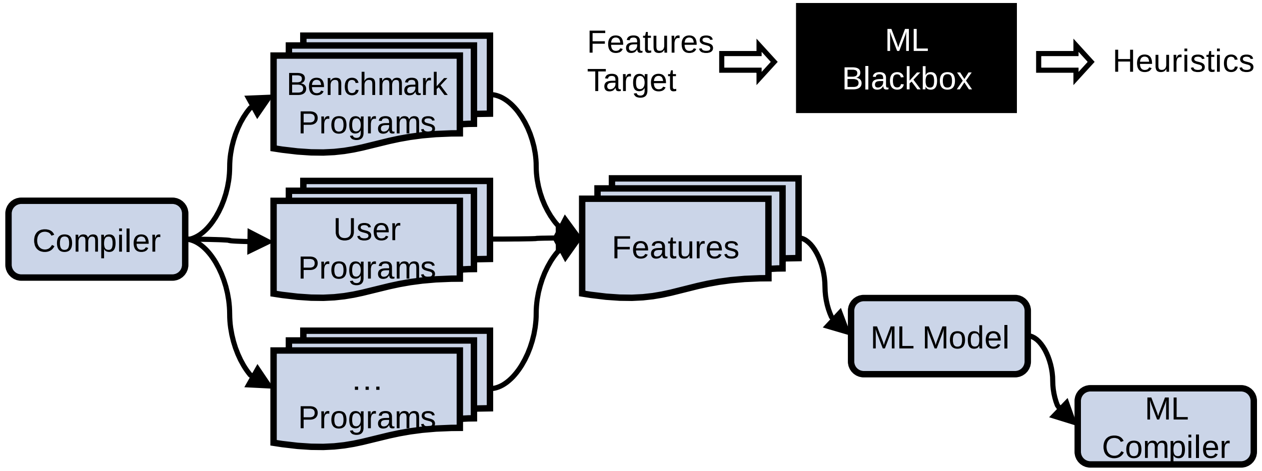


Photo by Dylan Gialanella on Unsplash

ML IN COMPILERS

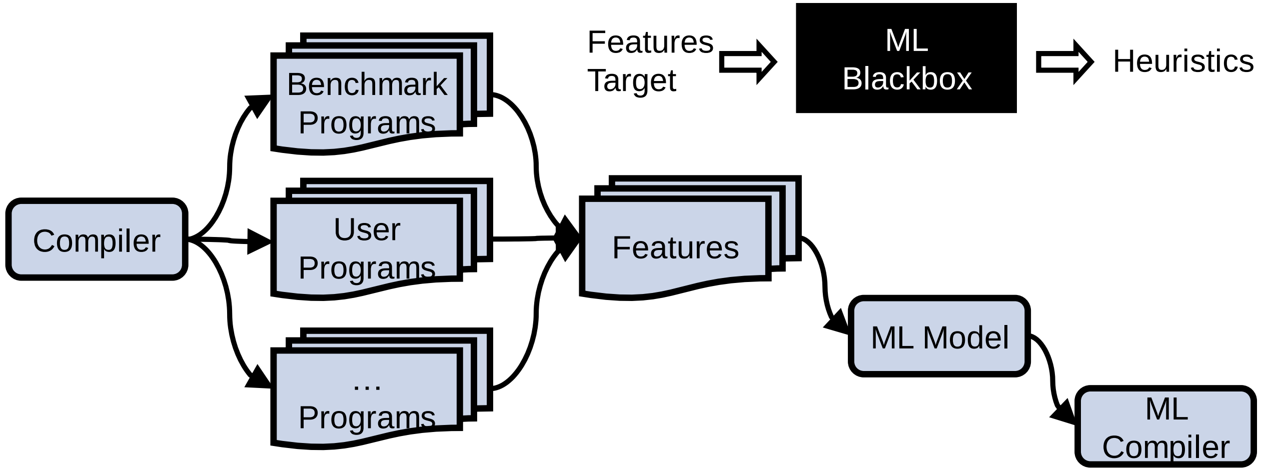


ML IN COMPILERS



- Captures large amount of „real“ world

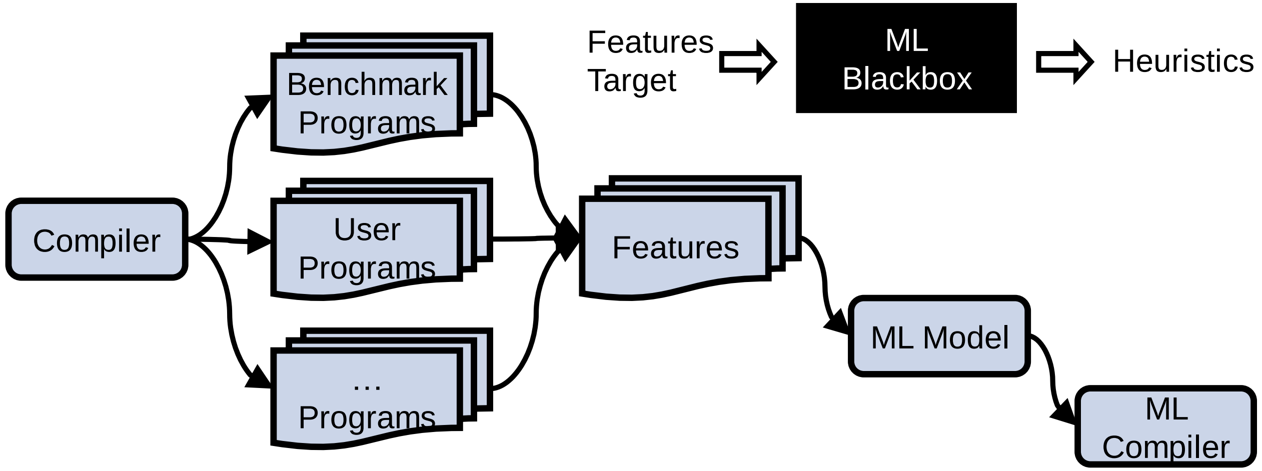
ML IN COMPILERS



- Captures large amount of „real“ world

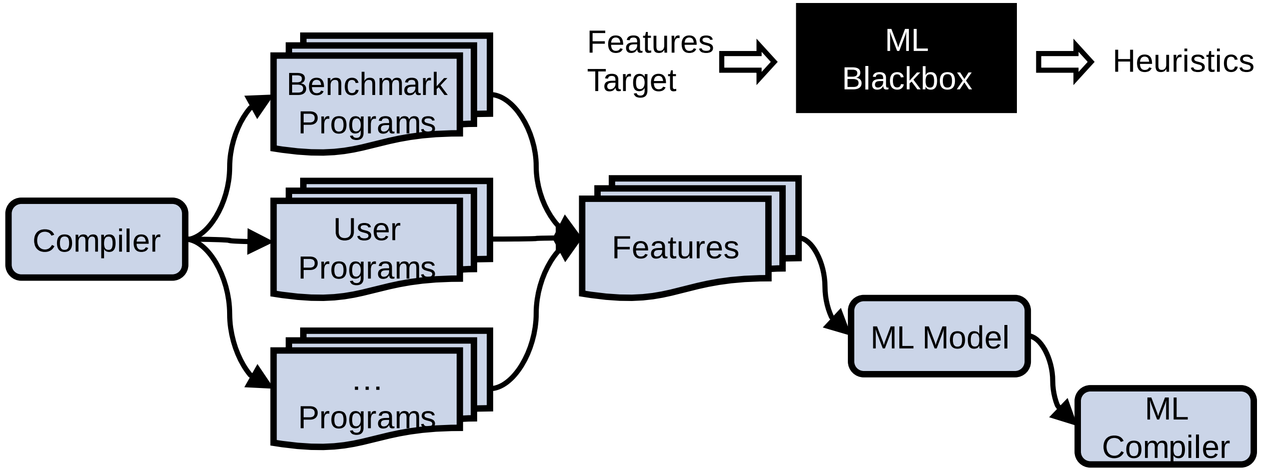
- (Semi-)automated
  - □ Waterfall

ML IN COMPILERS



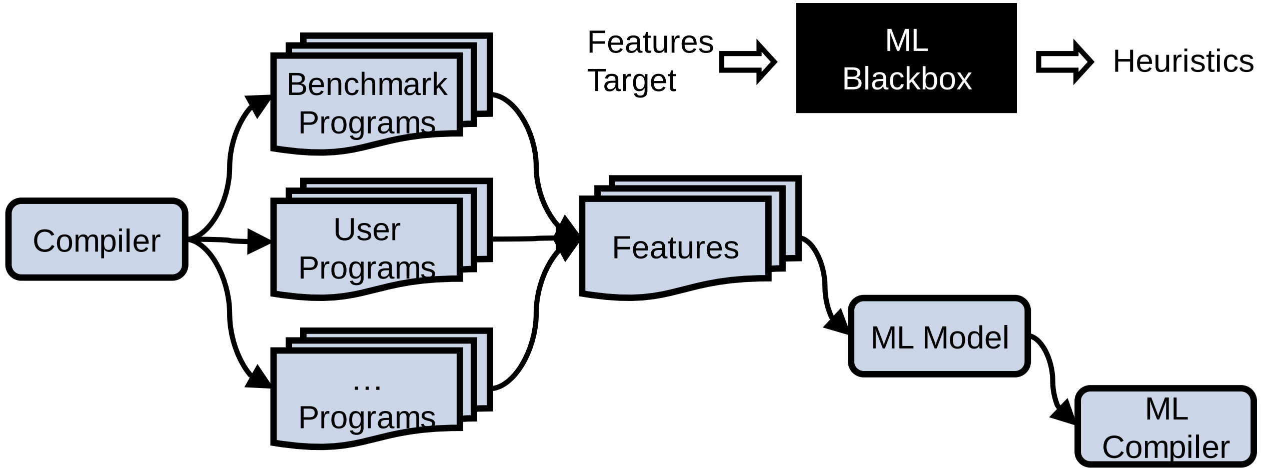
- Captures large amount of „real“ world
- (Semi-)automated
  - Waterfall
- Black boxes
  - Lack maintainability & understandability
  - Hard to infer compiler knowledge

ML IN COMPILERS



- Captures large amount of „real“ world
- (Semi-)automated
  - Waterfall
- Black boxes
  - Lack maintainability & understandability
  - Hard to infer compiler knowledge
- Online prediction has overhead
  - Crucial for dynamic compilation

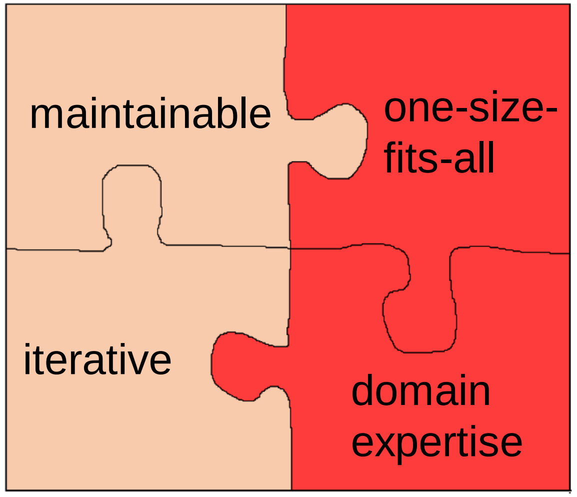
ML IN COMPILERS



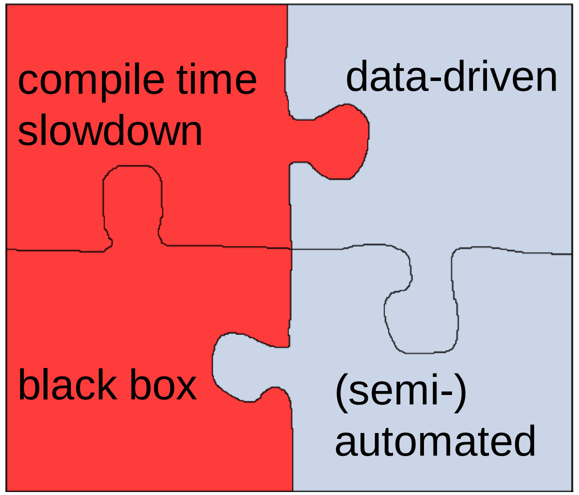
- Captures large amount of „real“ world
- (Semi-)automated
  - □ Waterfall
- Black boxes
  - □ Lack maintainability & understandability
  - □ Hard to infer compiler knowledge
- Online prediction has overhead
  - □ Crucial for dynamic compilation

data driven

UTILIZE MACHINE LEARNING ASSISTIVELY

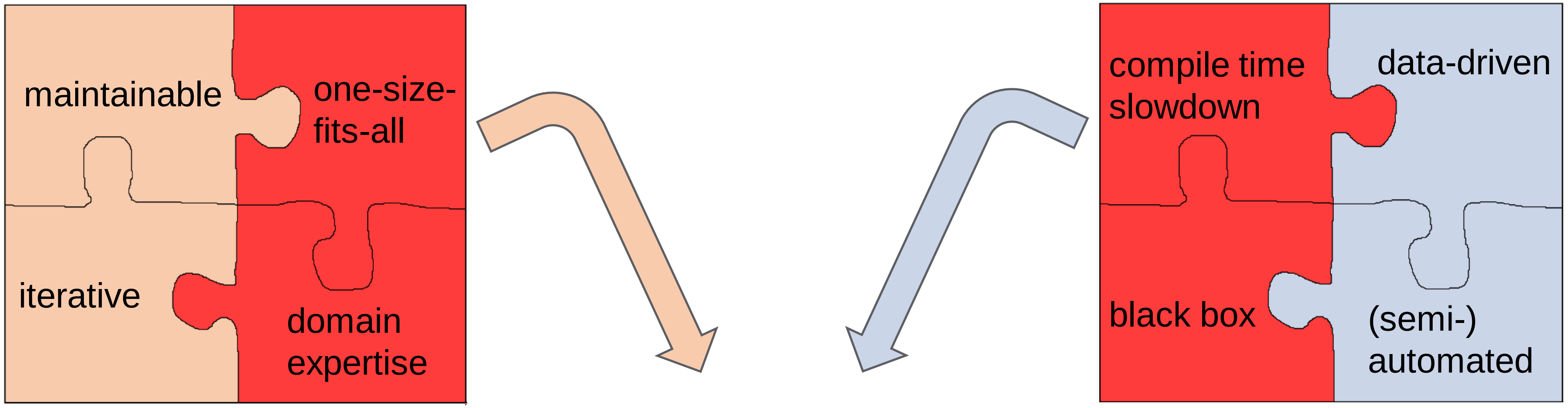


human-crafted heuristics



machine learned heuristics

UTILIZE MACHINE LEARNING ASSISTIVELY



maintainable

one-size-fits-all

iterative

domain expertise

compile time slowdown

data-driven

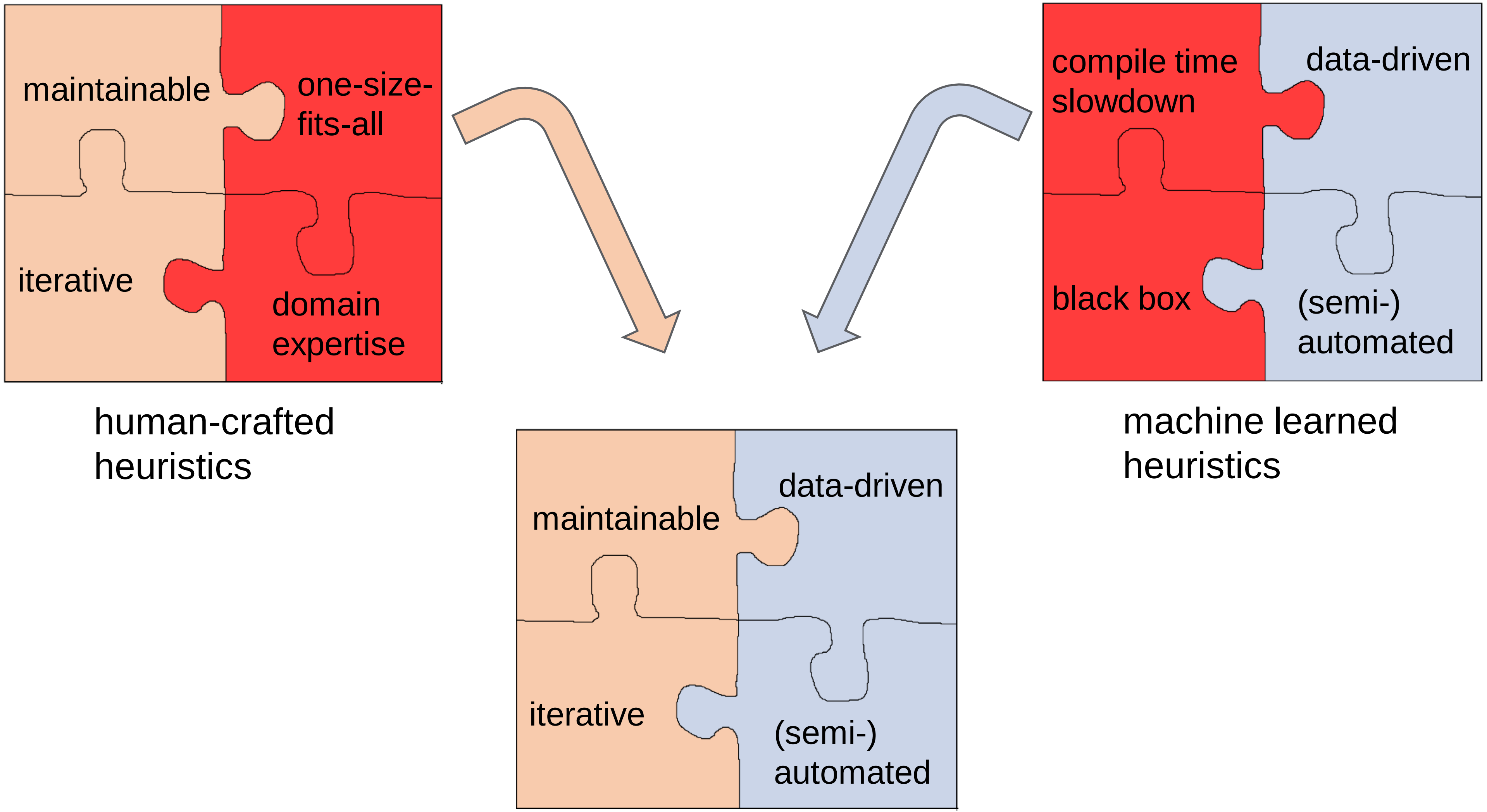
black box

(semi-) automated

human-crafted heuristics

machine learned heuristics

UTILIZE MACHINE LEARNING ASSISTIVELY



maintainable

one-size-fits-all

iterative

domain expertise

human-crafted heuristics

compile time slowdown

data-driven

black box

(semi-) automated

machine learned heuristics

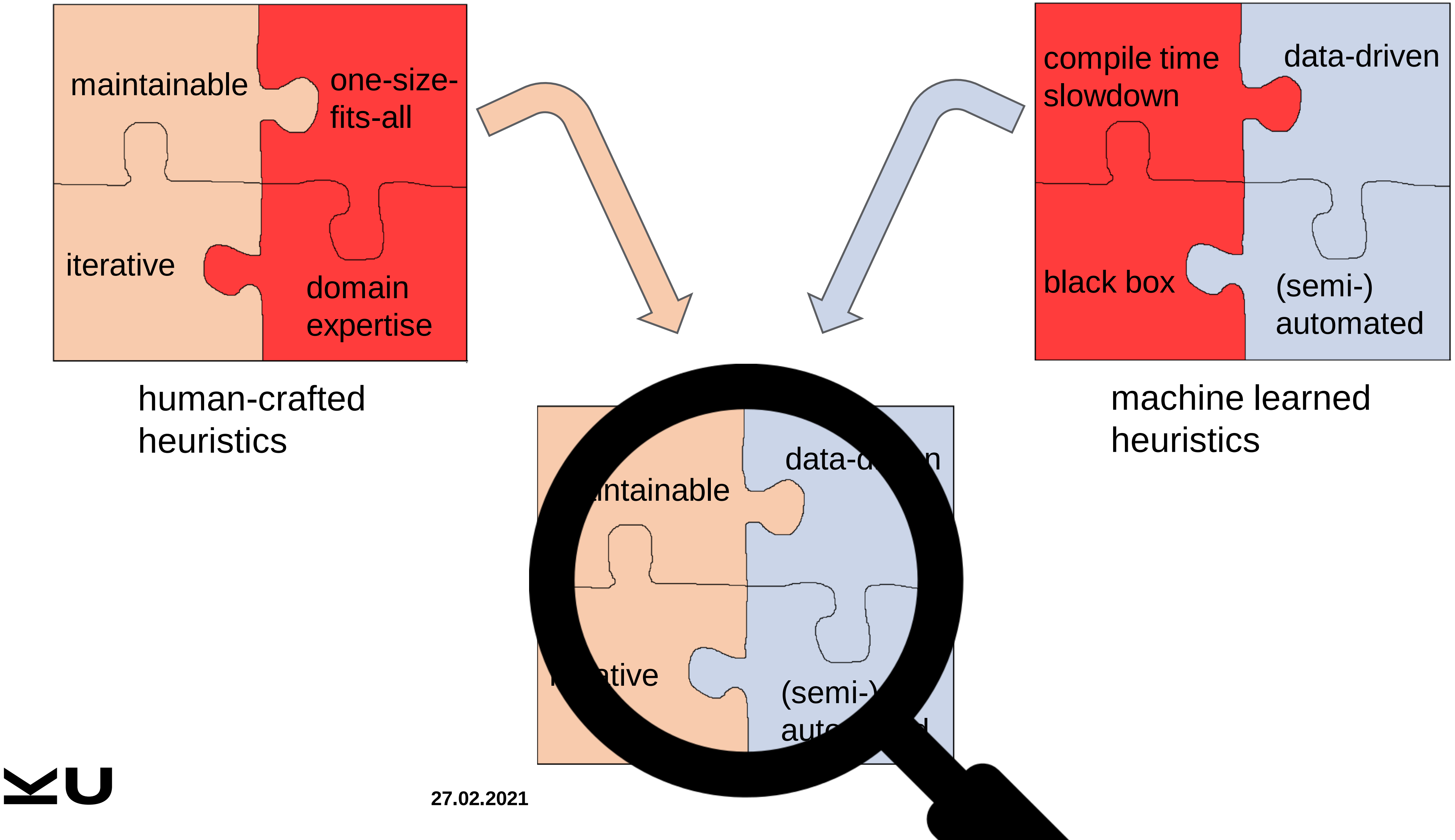
maintainable

data-driven

iterative

(semi-) automated

UTILIZE MACHINE LEARNING ASSISTIVELY



maintainable

one-size-fits-all

iterative

domain expertise

human-crafted heuristics

compile time slowdown

data-driven

black box

(semi-) automated

machine learned heuristics

intainable

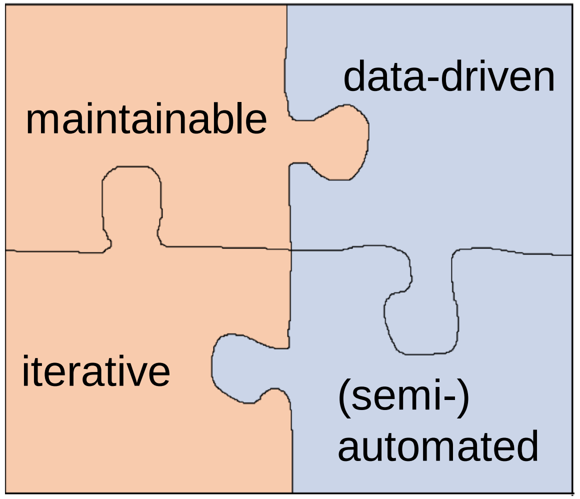
data-d n

tive

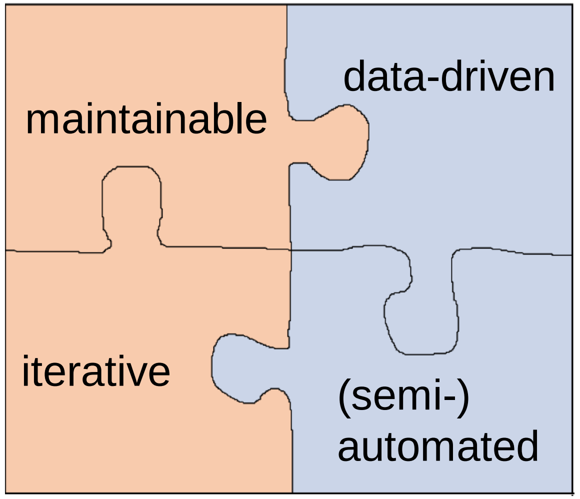
(semi-)

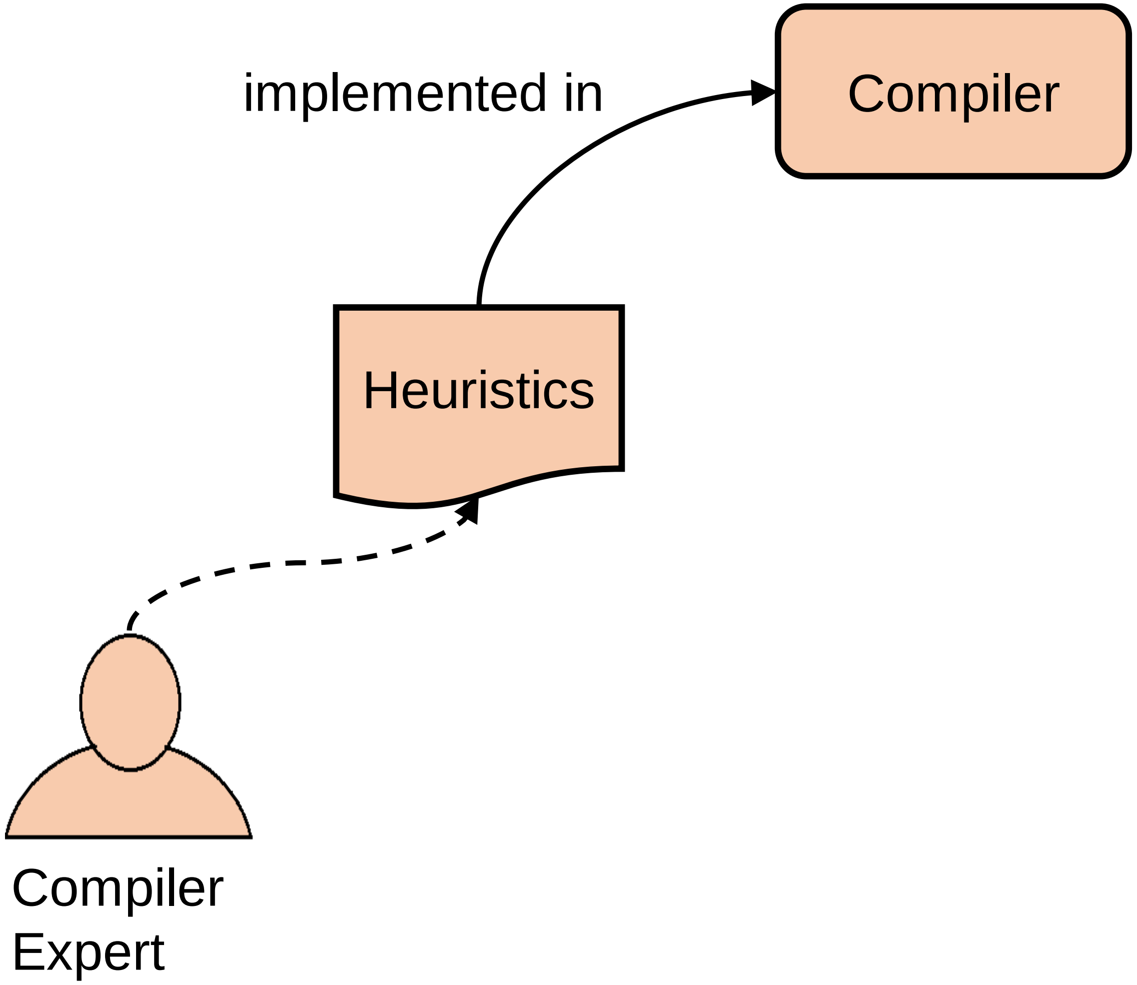
aut

COMBINED APPROACH

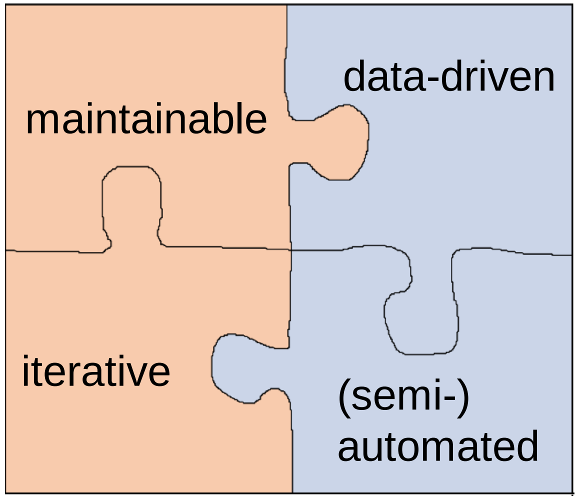


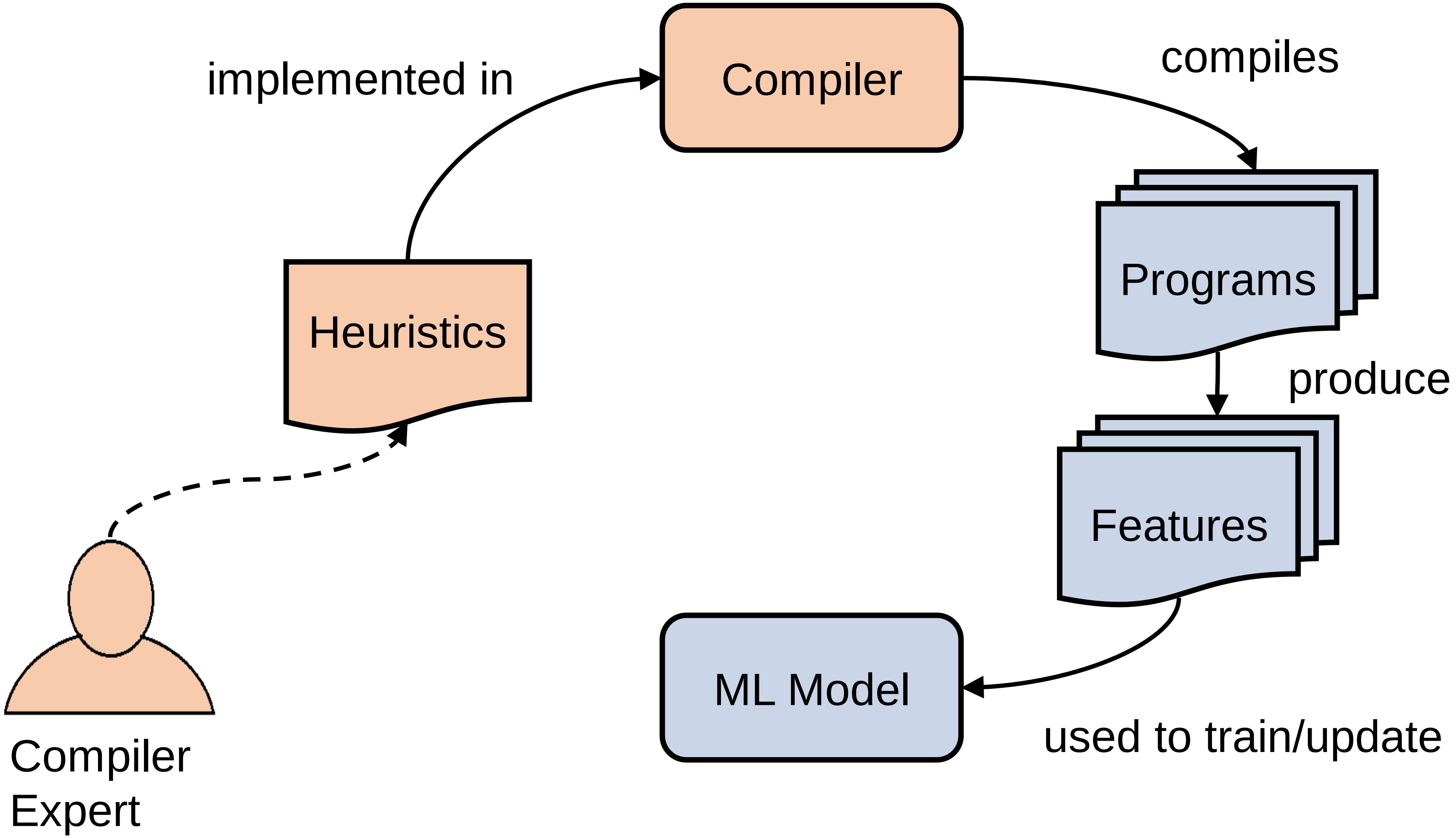
COMBINED APPROACH



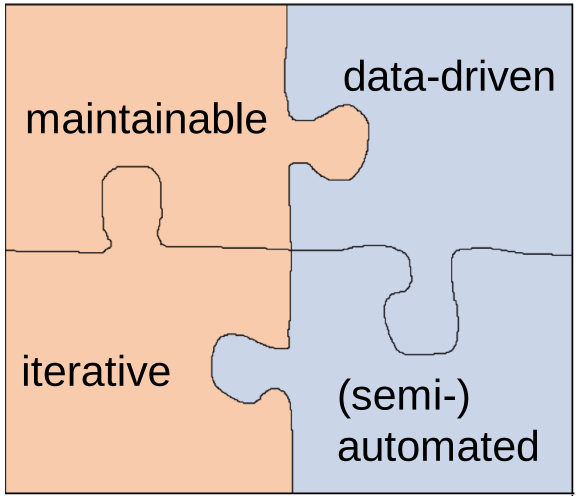


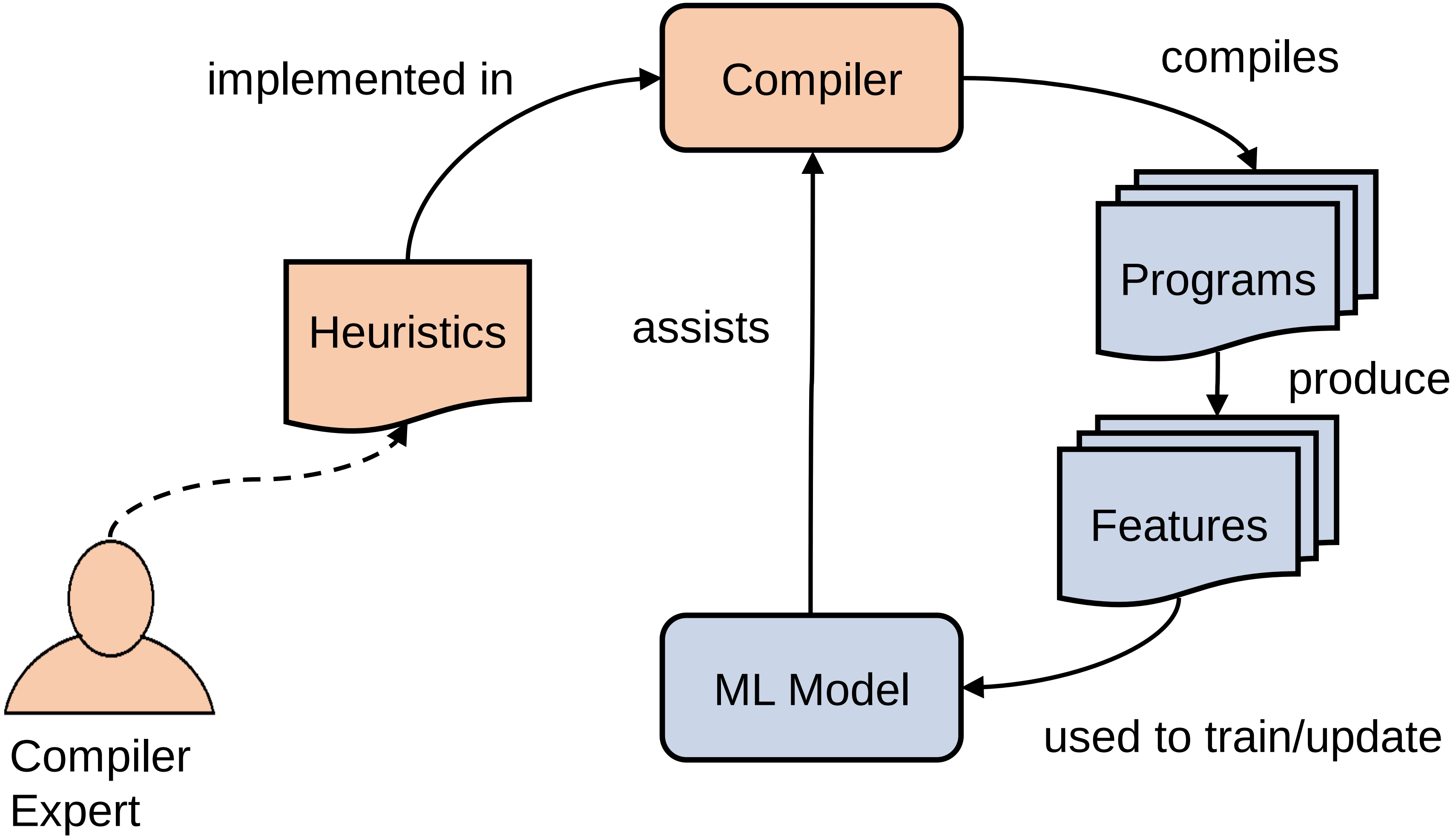
COMBINED APPROACH



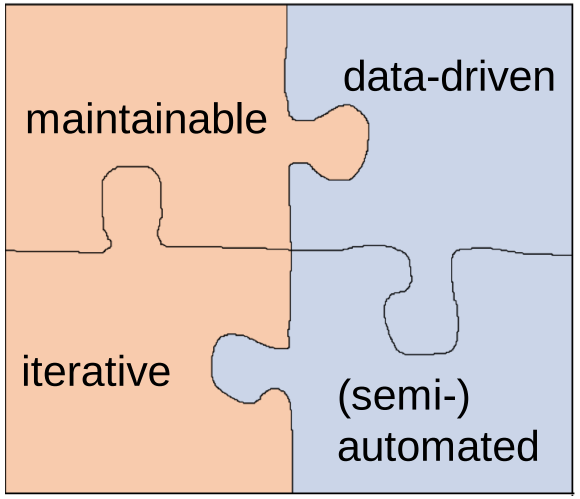


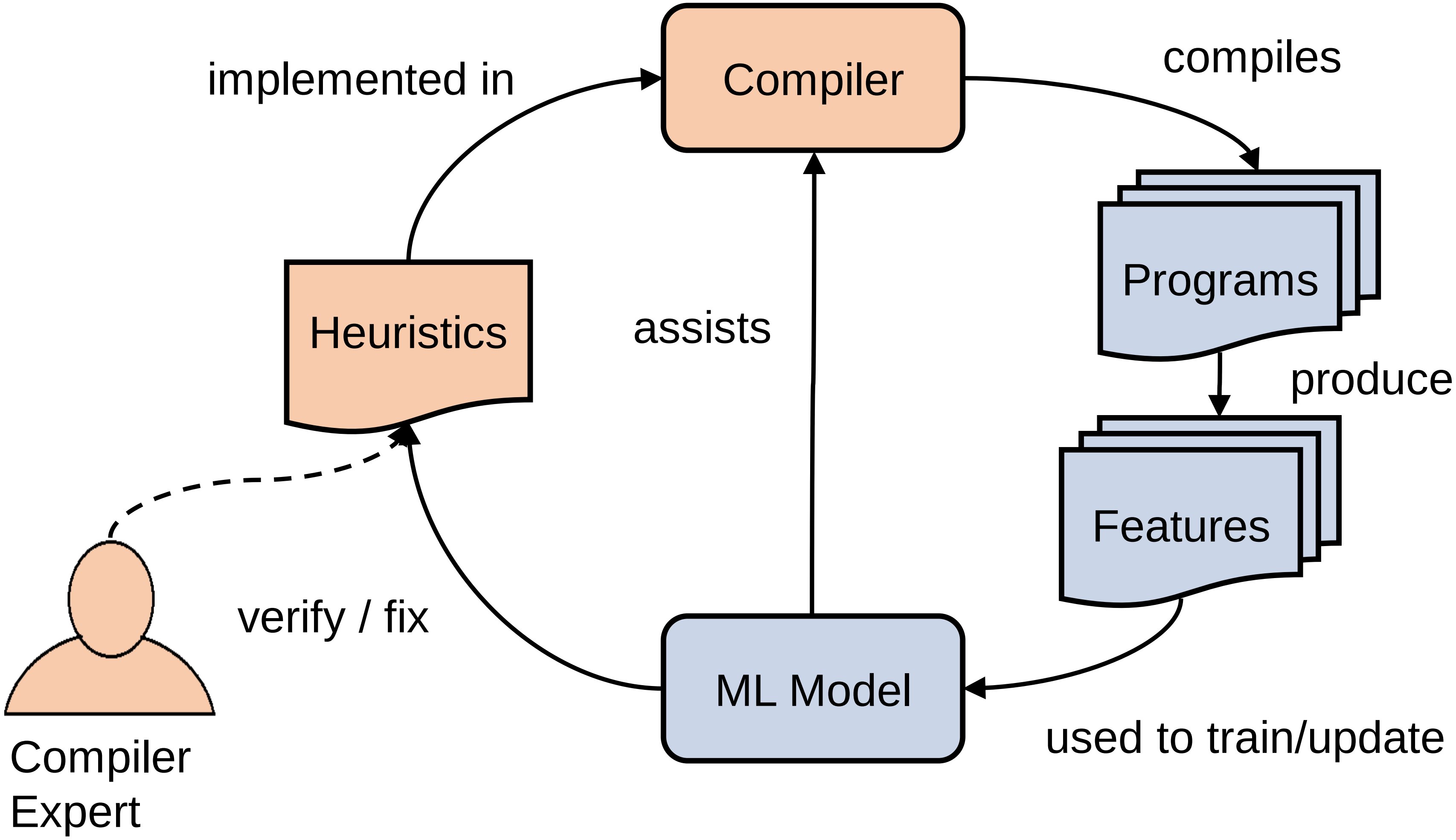
COMBINED APPROACH



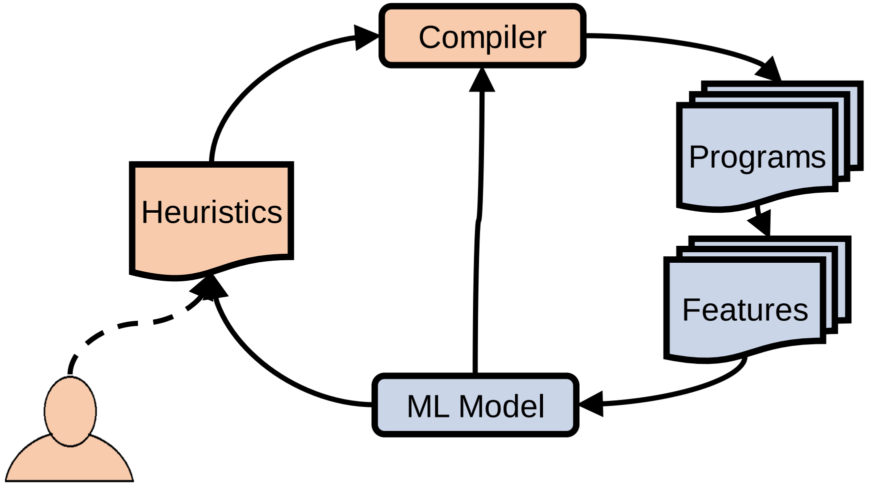


COMBINED APPROACH

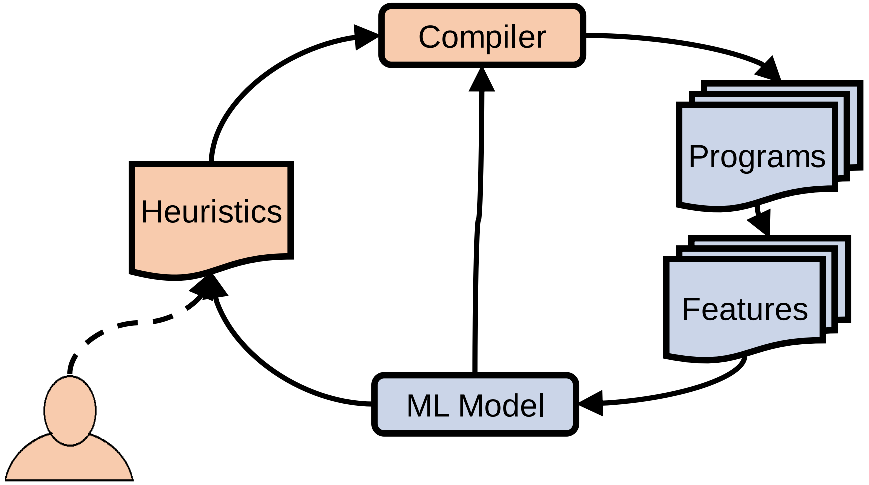




COMBINED APPROACH

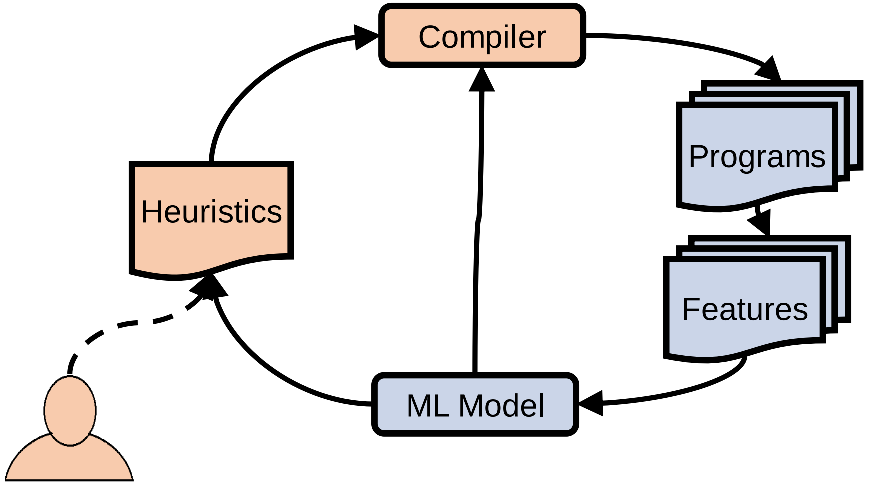


COMBINED APPROACH



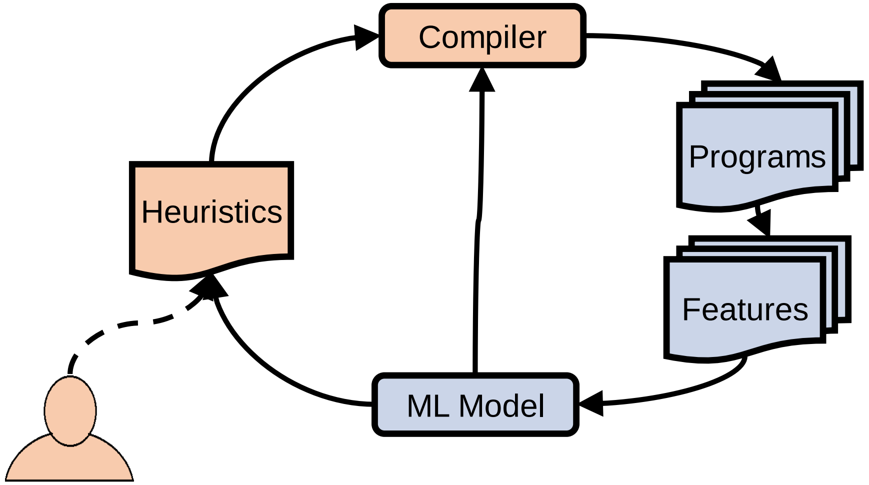
- Avoid black boxes in parts crucial for understandability

COMBINED APPROACH



- Avoid black boxes in parts crucial for understandability
- Automated feedback based on environmental changes
  - Do current heuristics fit data?

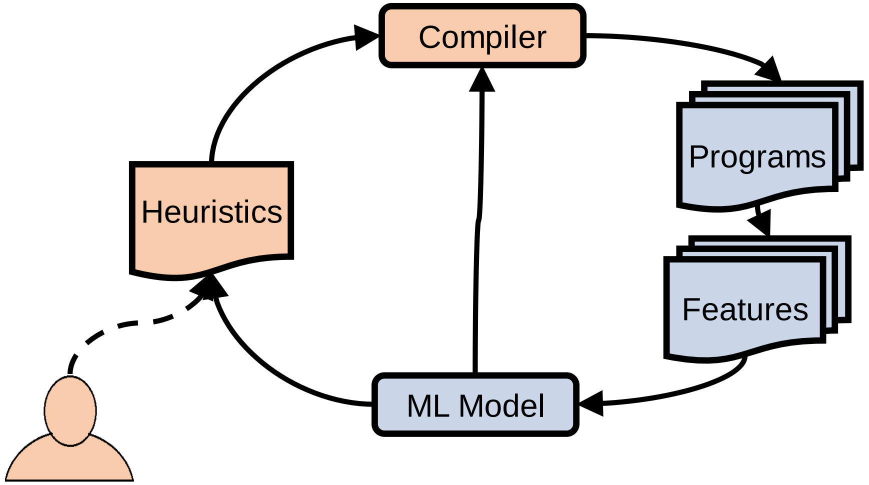
COMBINED APPROACH



- Avoid black boxes in parts crucial for understandability
- Automated feedback based on environmental changes
  - Do current heuristics fit data?
- Infer knowledge for compiler experts
  - Helpful for problem analysis
  - Compensate lack in experience

COMBINED APPROACH

- Avoid black boxes in parts crucial for understandability
- Automated feedback based on environmental changes
  - Do current heuristics fit data?
- Infer knowledge for compiler experts
  - Helpful for problem analysis
  - Compensate lack in experience



data guided

CASE STUDY:
DUPLICATION IN THE GRAAL COMPILER

DUPLICATION

```
if (x > 0) {
  phi = x;
} else {
  phi = 0;
}
return phi + 2;
```

DUPLICATION

```
if (x > 0) {
  phi = x;
} else {
  phi = 0;
}
return phi + 2;
```

Copy code after control flow merges…

DUPLICATION

```
if (x > 0) {
  phi = x;
} else {
  phi = 0;
}
return phi + 2;
```

```
if (x > 0) {
  phi = x;
  return phi + 2;
} else {
  phi = 0;
  return phi + 2;
}
```

Copy code after control flow merges… … into predecessor blocks …

DUPLICATION

```
if (x > 0) {
  phi = x;
} else {
  phi = 0;
}
return phi + 2;
```

Copy code after control flow merges…

```
if (x > 0) {
  phi = x;
  return phi + 2;
} else {
  phi = 0;
  return phi + 2;
}
```

… into predecessor blocks …

```
if (x > 0) {
  return x + 2;
} else {
  return 2;
}
```

… to enable further optimizations.

CASE STUDY:
DUPLICATION IN THE GRAAL COMPILER

CASE STUDY: DUPLICATION IN THE GRAAL COMPILER

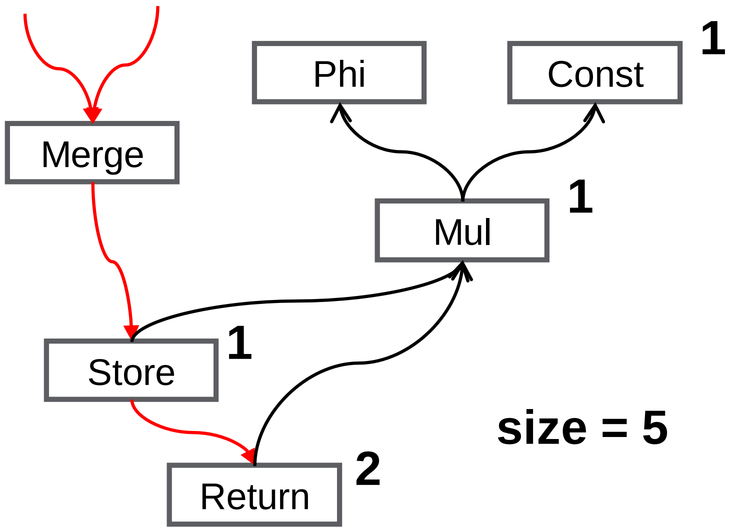
- **Heuristic** to trigger duplication: codeSize ↑ < performance↑ ? duplicate

CASE STUDY: DUPLICATION IN THE GRAAL COMPILER

- **Heuristic** to trigger duplication: codeSize ↑ < performance↑ ? duplicate
  - □ ΔcodeSize = $\sum_{node} size(node) * \#node$
  - □ Nodes are manually annotated with their size (= cost)

CASE STUDY: DUPLICATION IN THE GRAAL COMPILER

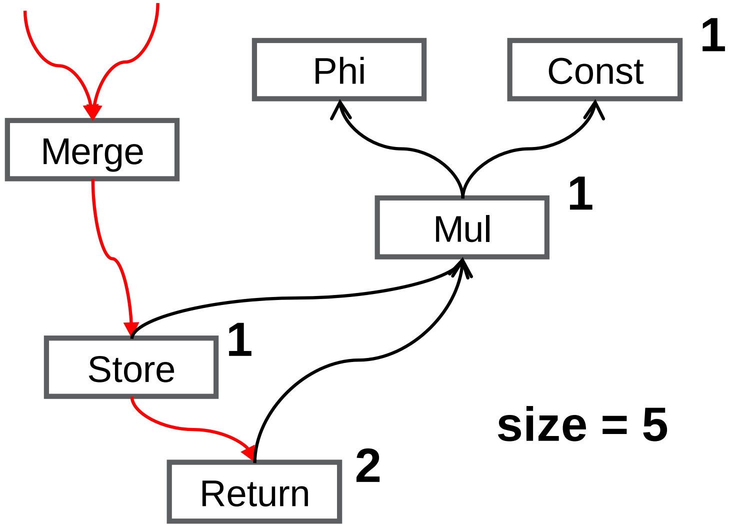
- **Heuristic** to trigger duplication: codeSize↑ < performance↑ ? duplicate
  - □ ΔcodeSize = $\sum_{node} size(node) * \#node$
  - □ Nodes are manually annotated with their size (= cost)



CASE STUDY:
DUPLICATION IN THE GRAAL COMPILER

- **Heuristic** to trigger duplication: codeSize↑ < performance↑ ? duplicate
  - □ ΔcodeSize = $\sum_{node} size(node) * \#node$
  - □ Nodes are manually annotated with their size (= cost)

- Hand crafted **cost model** for over 450 different IR nodes
  - □ Code size
  - □ Execution cycles

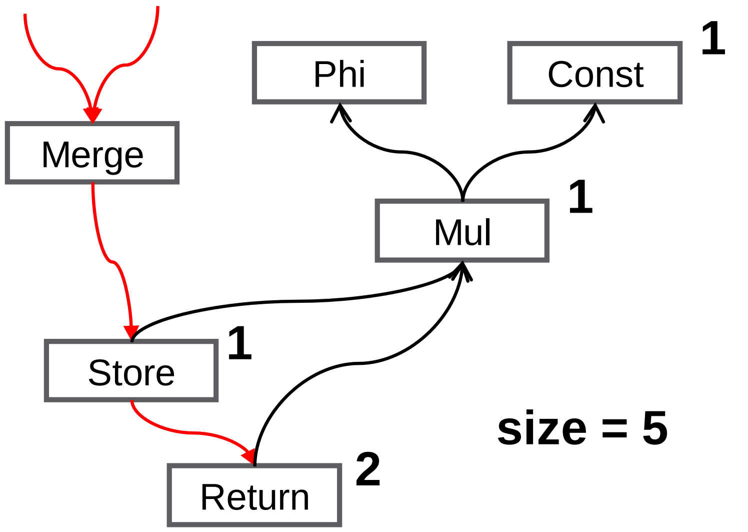


CASE STUDY: DUPLICATION IN THE GRAAL COMPILER

- **Heuristic** to trigger duplication: codeSize ↑ < performance ↑ ? duplicate
  - □ ΔcodeSize = $\sum_{node} size(node) * \#node$
  - □ Nodes are manually annotated with their size (= cost)

- Hand crafted **cost model** for over 450 different IR nodes
  - □ Code size
  - □ Execution cycles

- Node costs are only estimations made from experience



GOAL:
VERIFY OR FIX NODE COST MODEL

GOAL:
VERIFY OR FIX NODE COST MODEL

- Learn „actual“ code size impact per IR node based on data

GOAL:
VERIFY OR FIX NODE COST MODEL

- Learn „actual“ code size impact per IR node based on data

- Features: IR node counts
  - [#AddNode, #SubNode, #IfNode, …]

GOAL:
VERIFY OR FIX NODE COST MODEL

- Learn „actual“ code size impact per IR node based on data

- Features: IR node counts
  - [#AddNode, #SubNode, #IfNode, …]

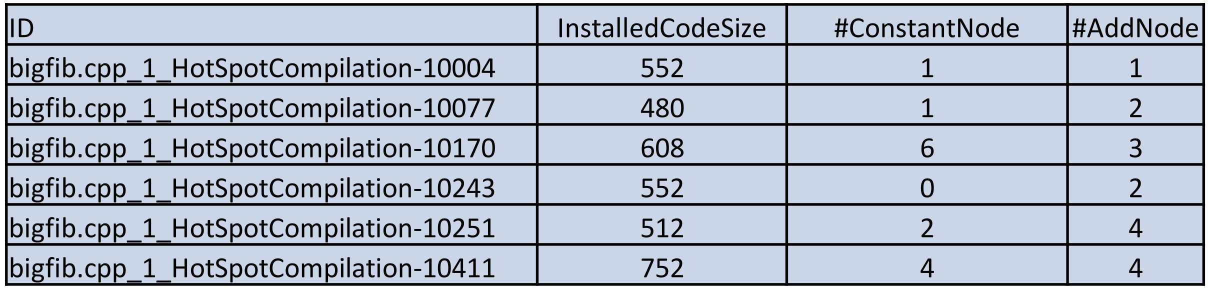
- Target: code size in bytes

GOAL:
VERIFY OR FIX NODE COST MODEL

- Learn „actual“ code size impact per IR node based on data

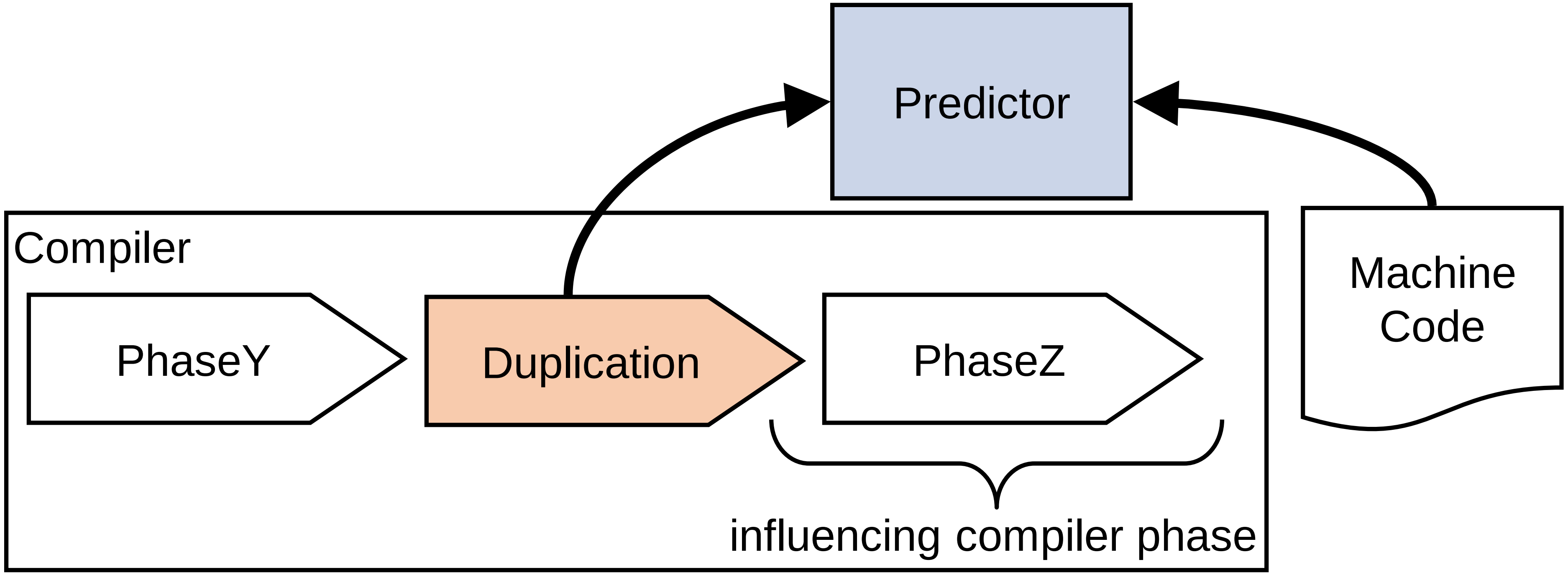
- Features: IR node counts
  - □ [#AddNode, #SubNode, #IfNode, …]

- Target: code size in bytes

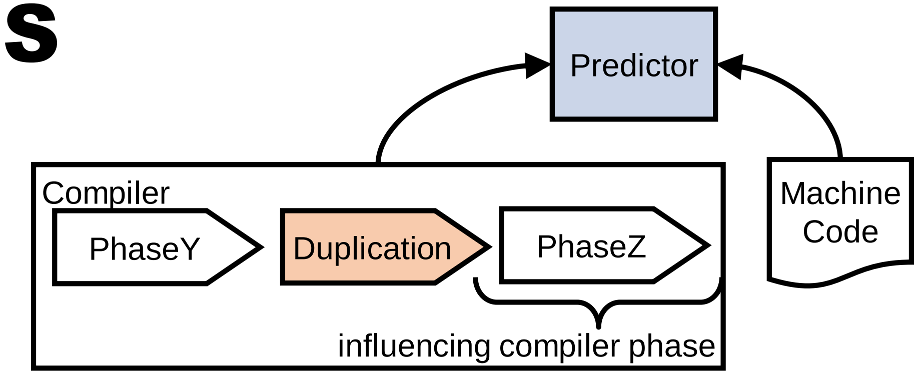


ID	InstalledCodeSize	#ConstantNode	#AddNode
bigfib.cpp_1_HotSpotCompilation-10004	552	1	1
bigfib.cpp_1_HotSpotCompilation-10077	480	1	2
bigfib.cpp_1_HotSpotCompilation-10170	608	6	3
bigfib.cpp_1_HotSpotCompilation-10243	552	0	2
bigfib.cpp_1_HotSpotCompilation-10251	512	2	4
bigfib.cpp_1_HotSpotCompilation-10411	752	4	4

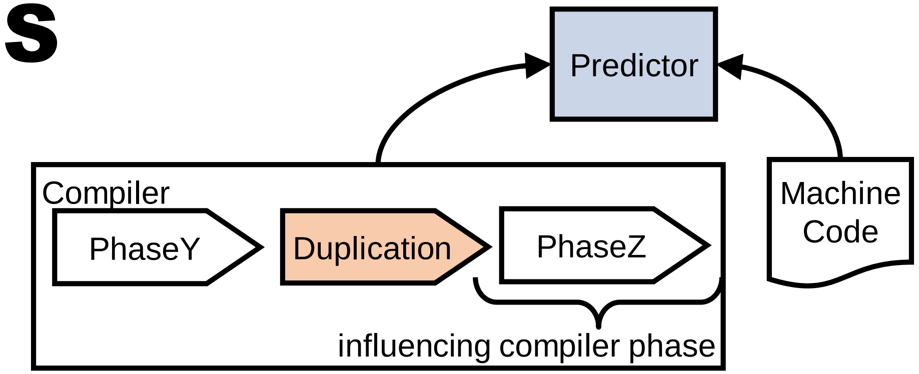
PROBLEM: SUBSEQUENT COMPILER PHASES



PROBLEM: SUBSEQUENT COMPILER PHASES

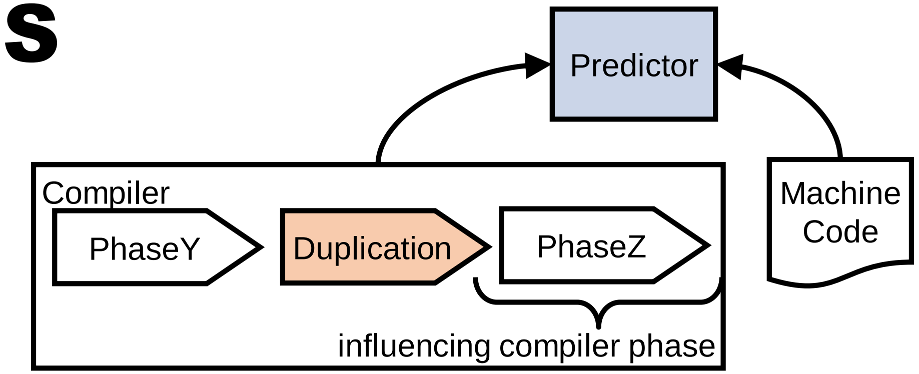


PROBLEM: SUBSEQUENT COMPILER PHASES



- Transformations on IR level wreck linear relation between nodes and final code size

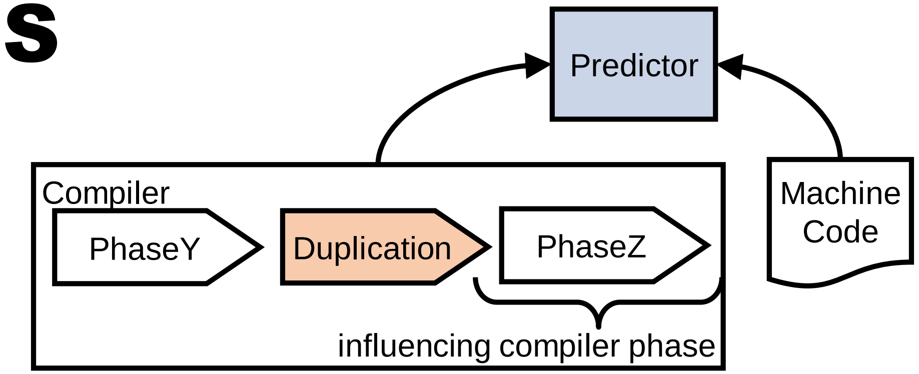
PROBLEM:
SUBSEQUENT COMPILER PHASES



- Transformations on IR level wreck linear relation between nodes and final code size

- Linear regression model mispredicts code size impact

PROBLEM: SUBSEQUENT COMPILER PHASES



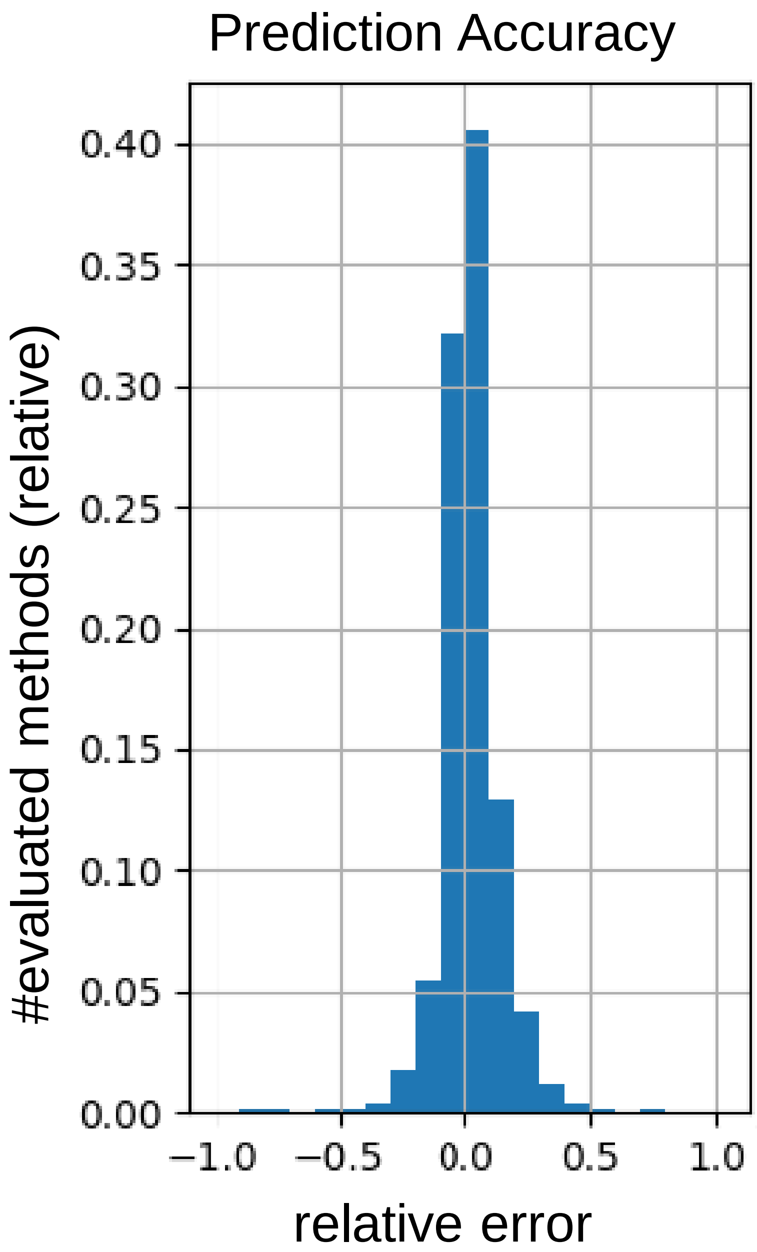
- Transformations on IR level wreck linear relation between nodes and final code size
- Linear regression model mispredicts code size impact
- Requires non-linear predictor to account for intermediate compiler phases

ANN PREDICTOR

- Trained a simple ANN on benchmarks
  - dacapo, scala-dacapo, octane, jetstream, renaissance

ANN PREDICTOR

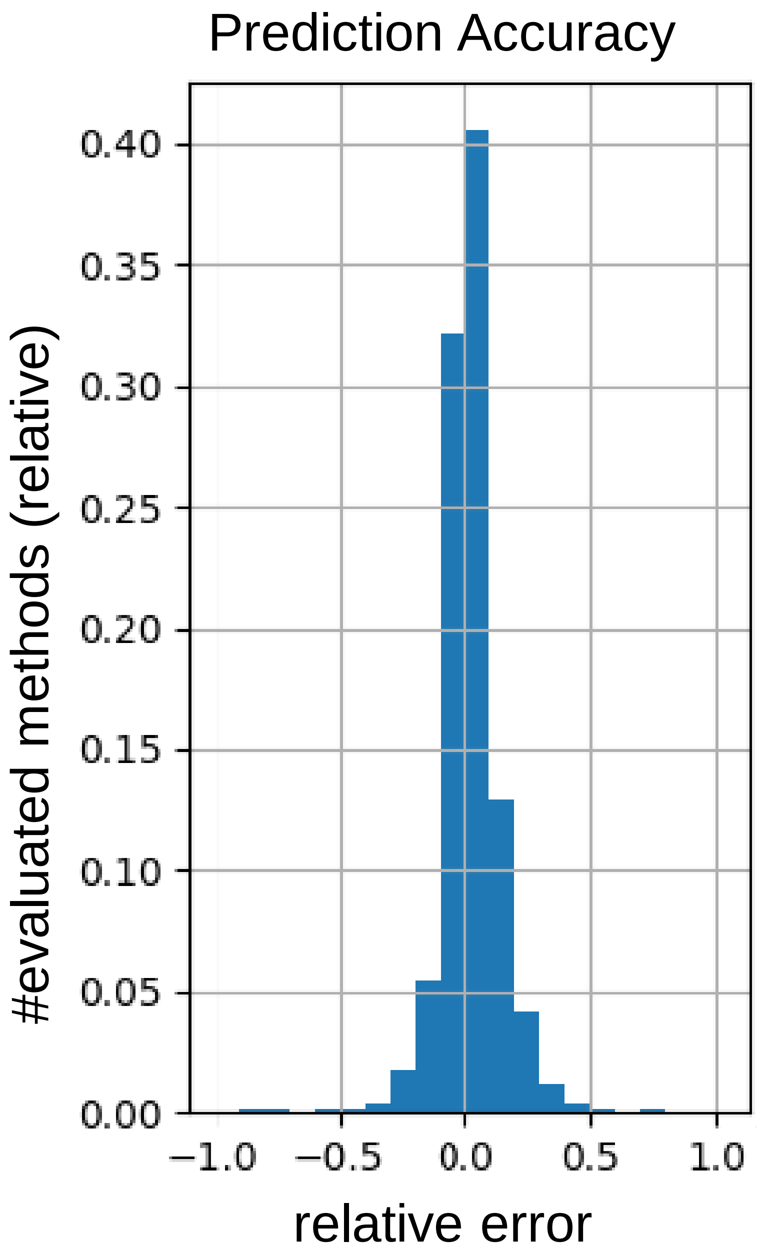
- Trained a simple ANN on benchmarks
  - dacapo, scala-dacapo, octane, jetstream, renaissance



ANN PREDICTOR

- Trained a simple ANN on benchmarks
  - dacapo, scala-dacapo, octane, jetstream, renaissance

- Accurately predicts code size impact
  - 3 out of 4 predictions have **errors <10%**

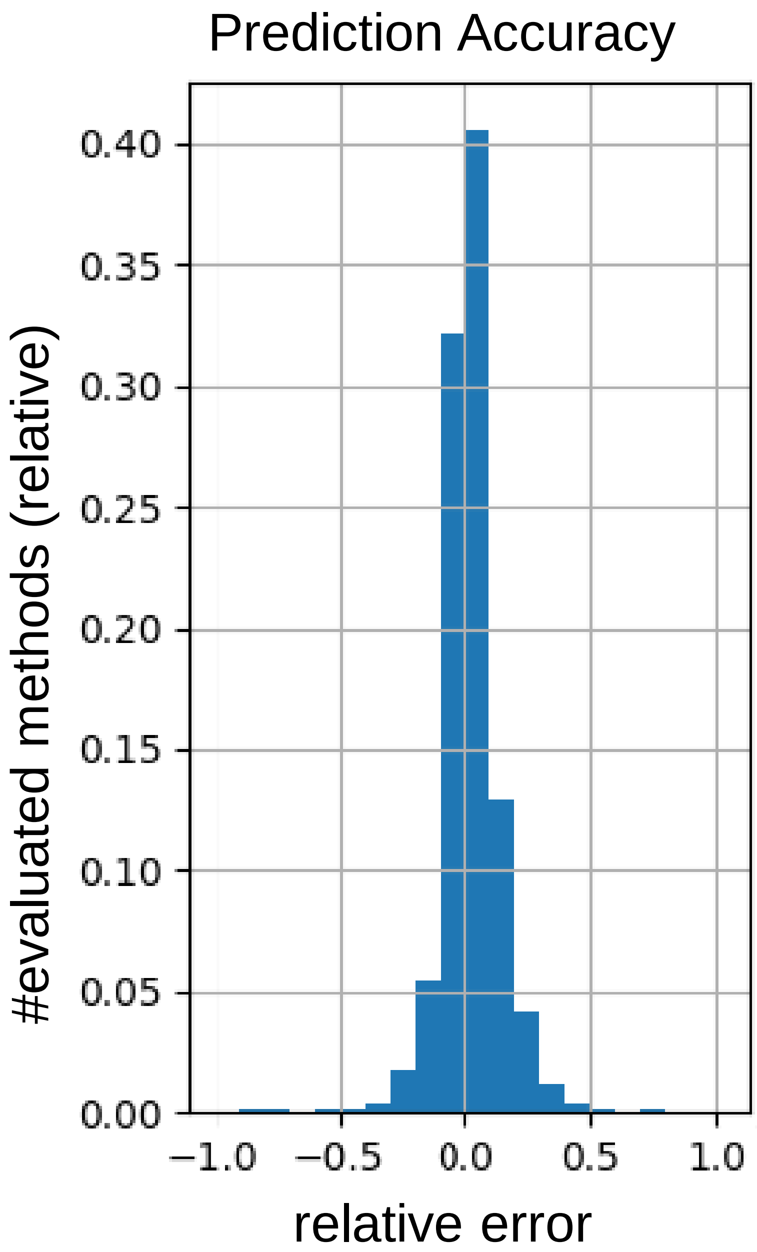


ANN PREDICTOR

- Trained a simple ANN on benchmarks
  - dacapo, scala-dacapo, octane, jetstream, renaissance

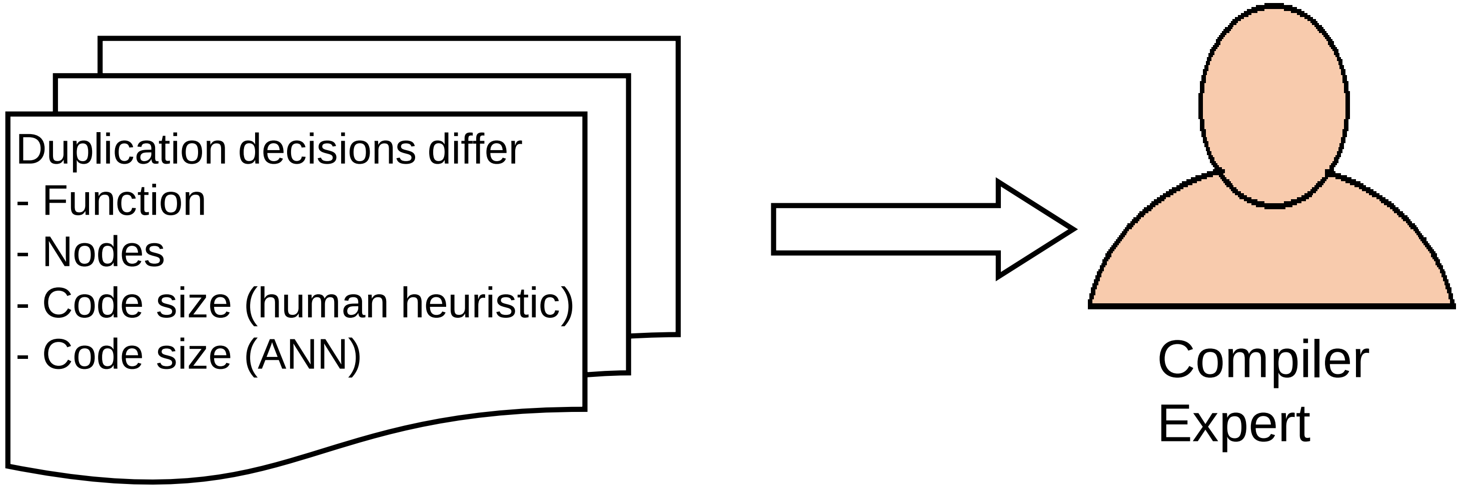
- Accurately predicts code size impact
  - 3 out of 4 predictions have **errors <10%**

- Implemented a prototype predictor in the Graal compiler



PRODUCING HELPFUL OUTPUT

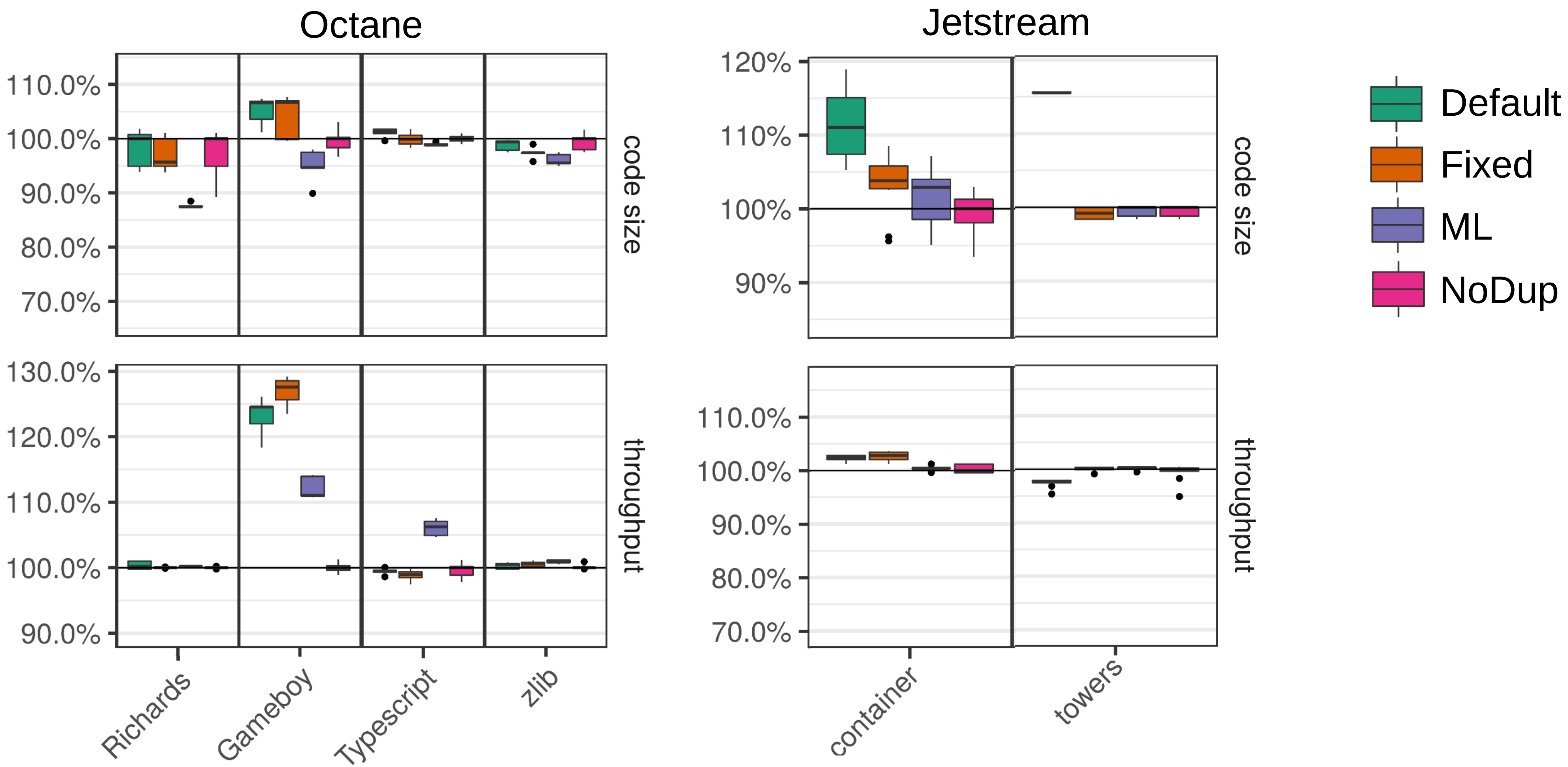
- Analysis mode in Graal
  - Prints differences in duplication decisions based on human model vs. learned model
  - Results provided to compiler expert



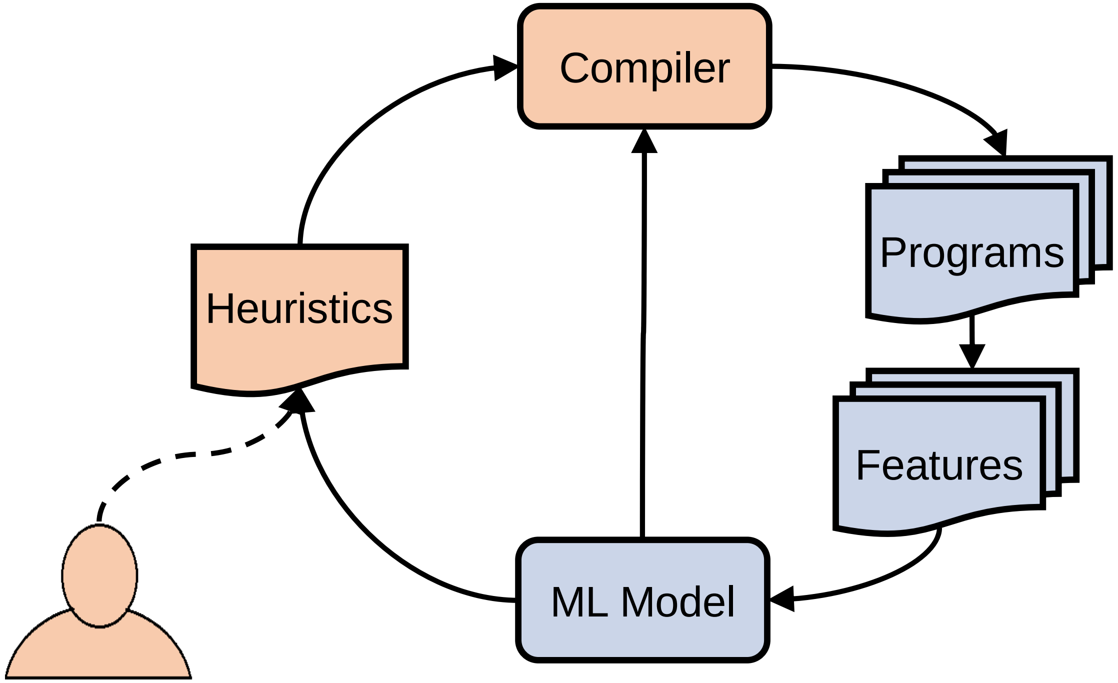
Duplication decisions differ
- Function
- Nodes
- Code size (human heuristic)
- Code size (ANN)

Compiler
Expert

BENCHMARK PERFORMANCE (SELECTION)



IMPROVING COMPILER OPTIMIZATIONS BY EMPLOYING MACHINE LEARNING



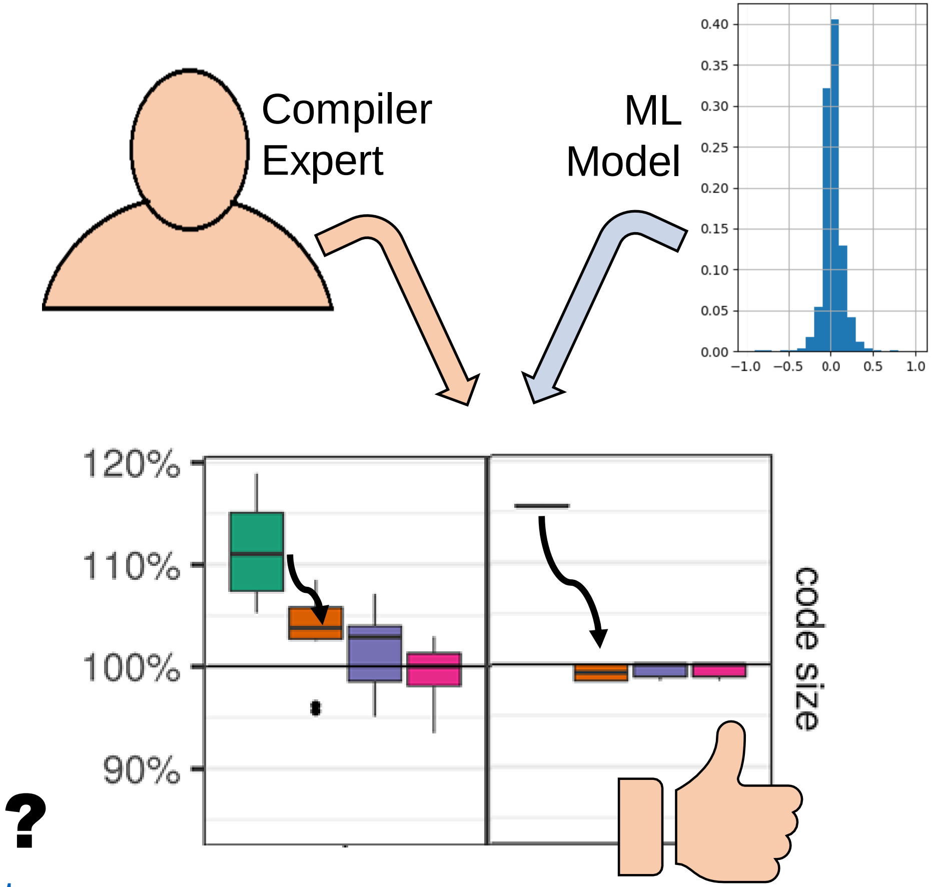
Compiler

Programs

Features

ML Model

Heuristics



Compiler Expert

ML Model

QUESTIONS ?

raphael.mosaner@jku.at

www.ssw.jku.at/General/Staff/Mosaner