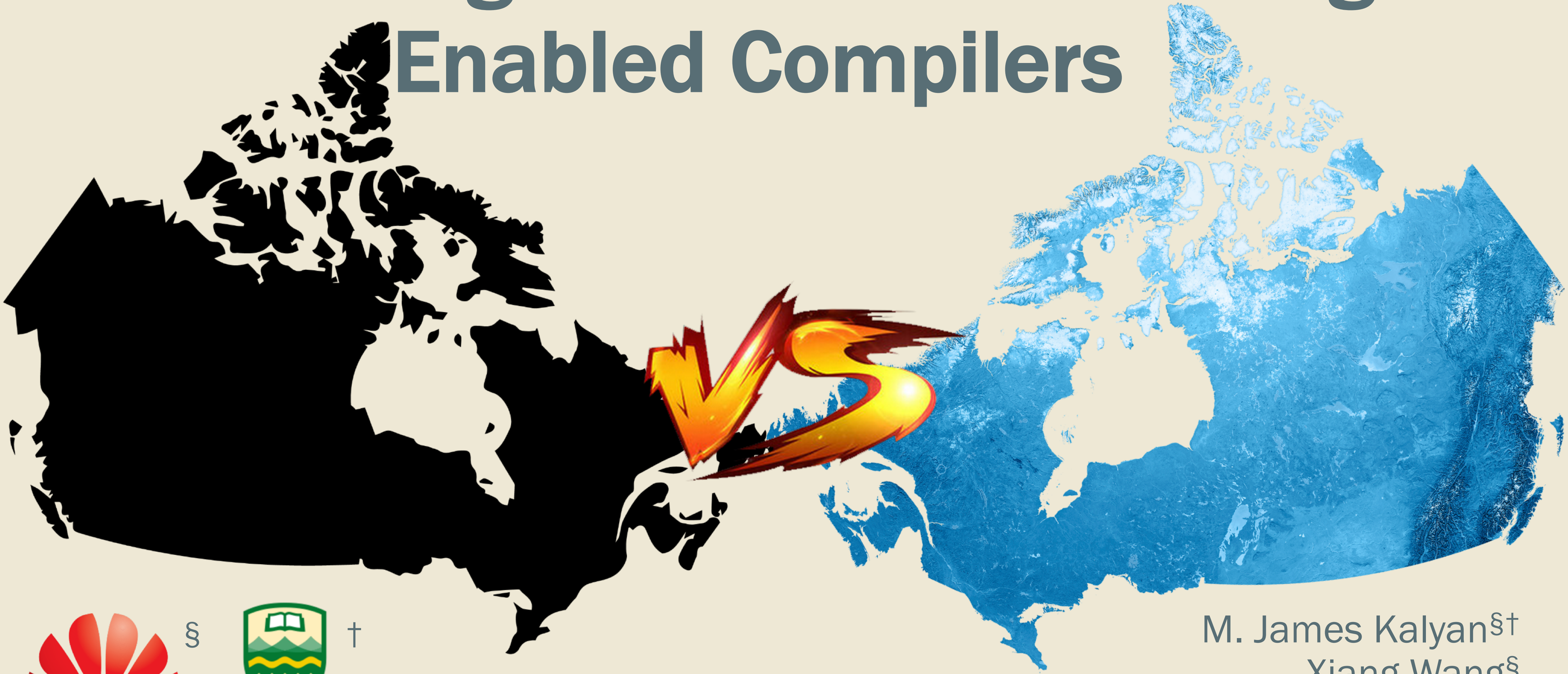
Code Region Based Auto-Tuning Enabled Compilers

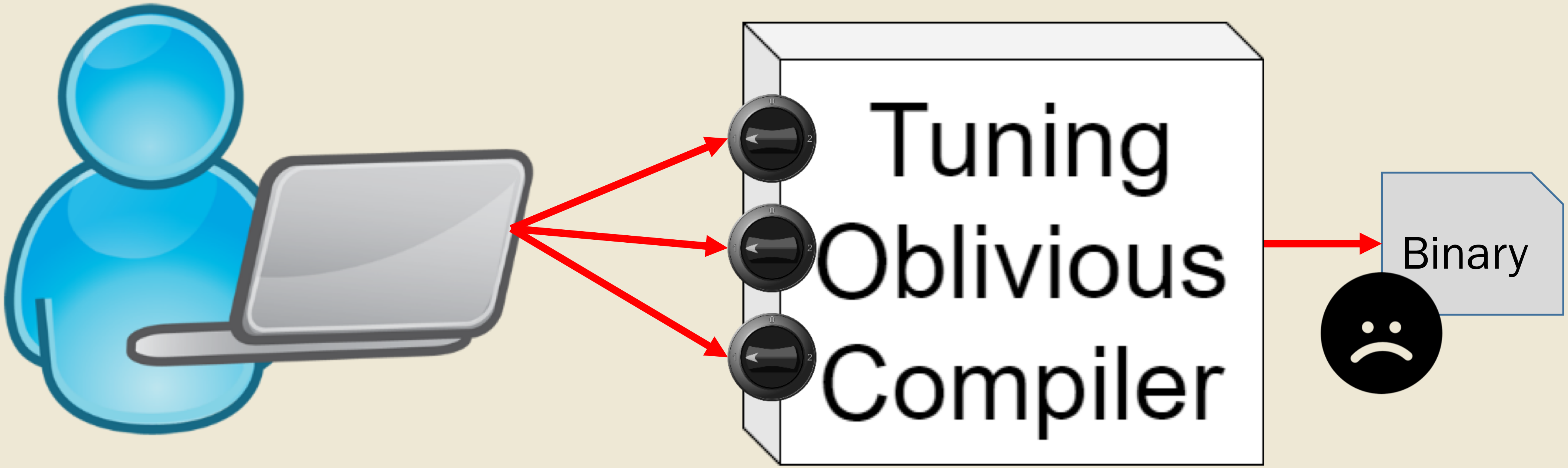


HUAWEI [§]

UNIVERSITY OF ALBERTA [†]

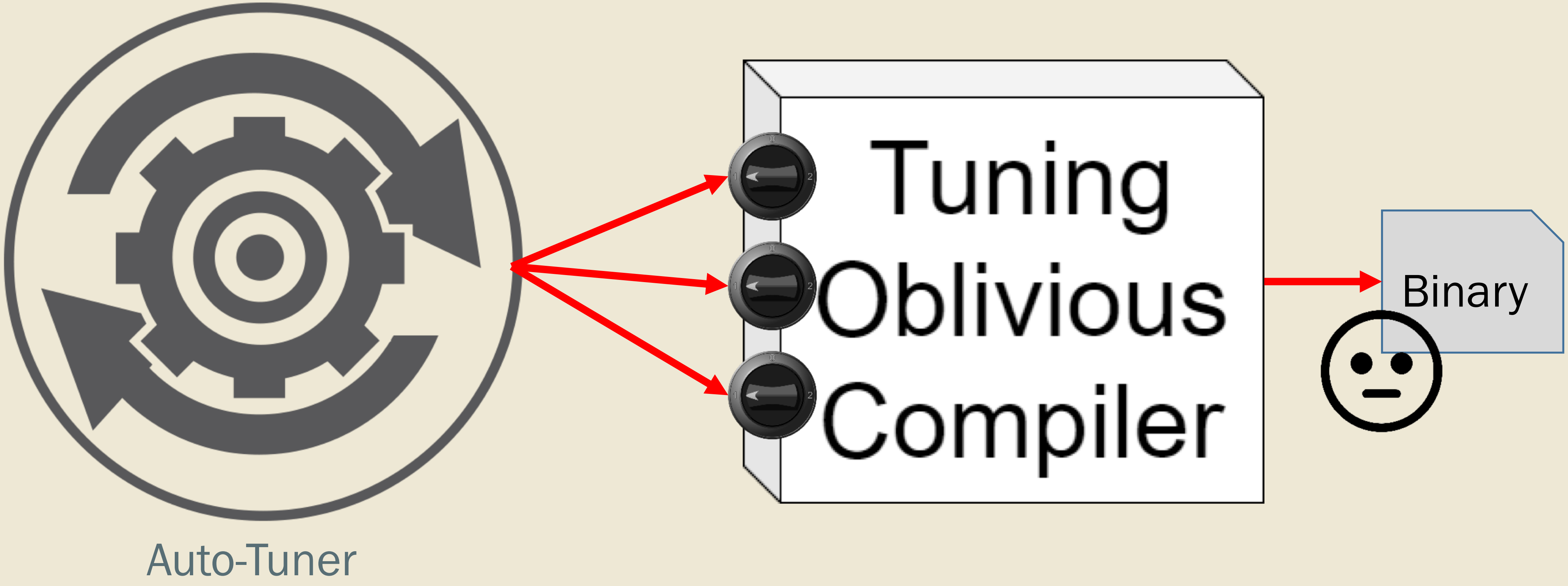
M. James Kalyan[§,†]
Xiang Wang[§]
Ahmed Eltantawy[§]
Yaoqing Gao[§]

Motivation

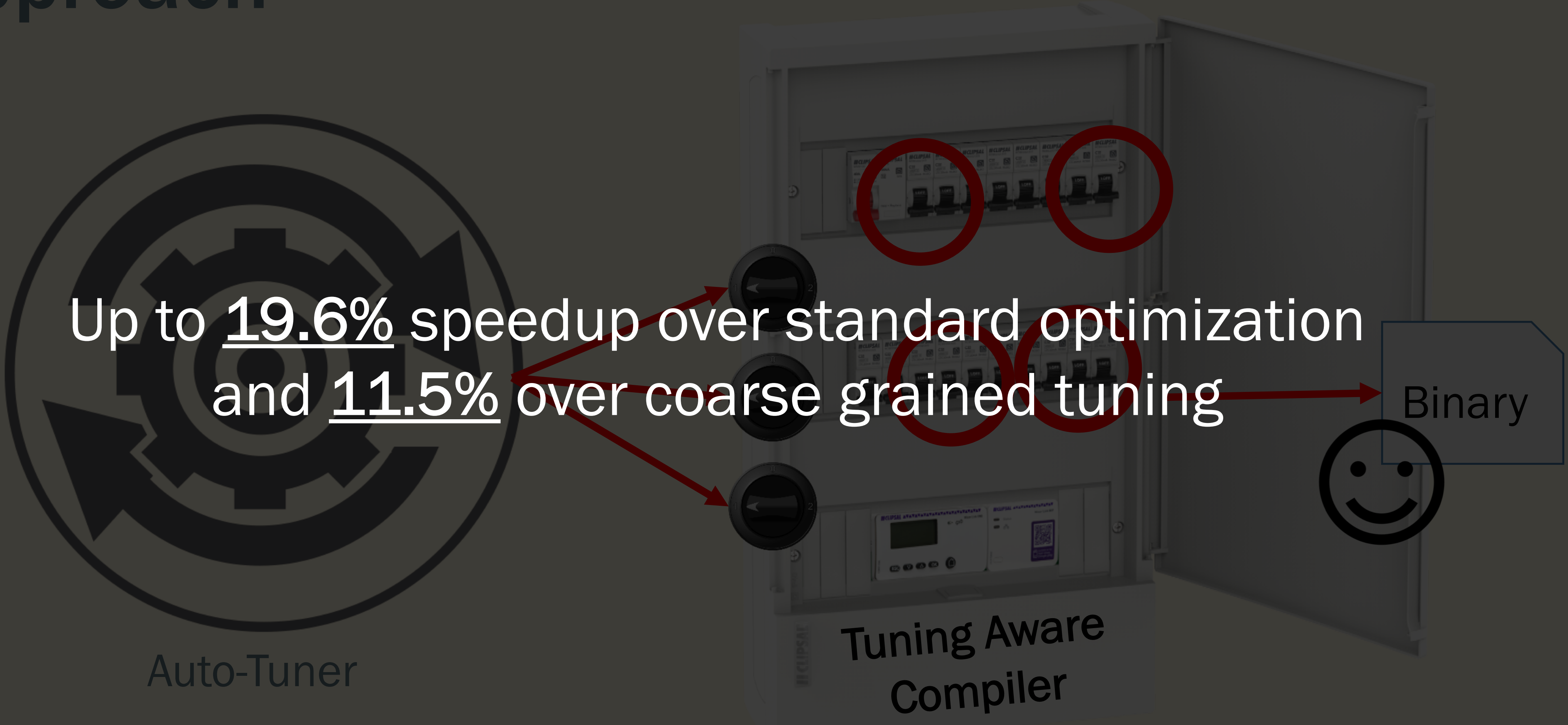


Developer

Motivation

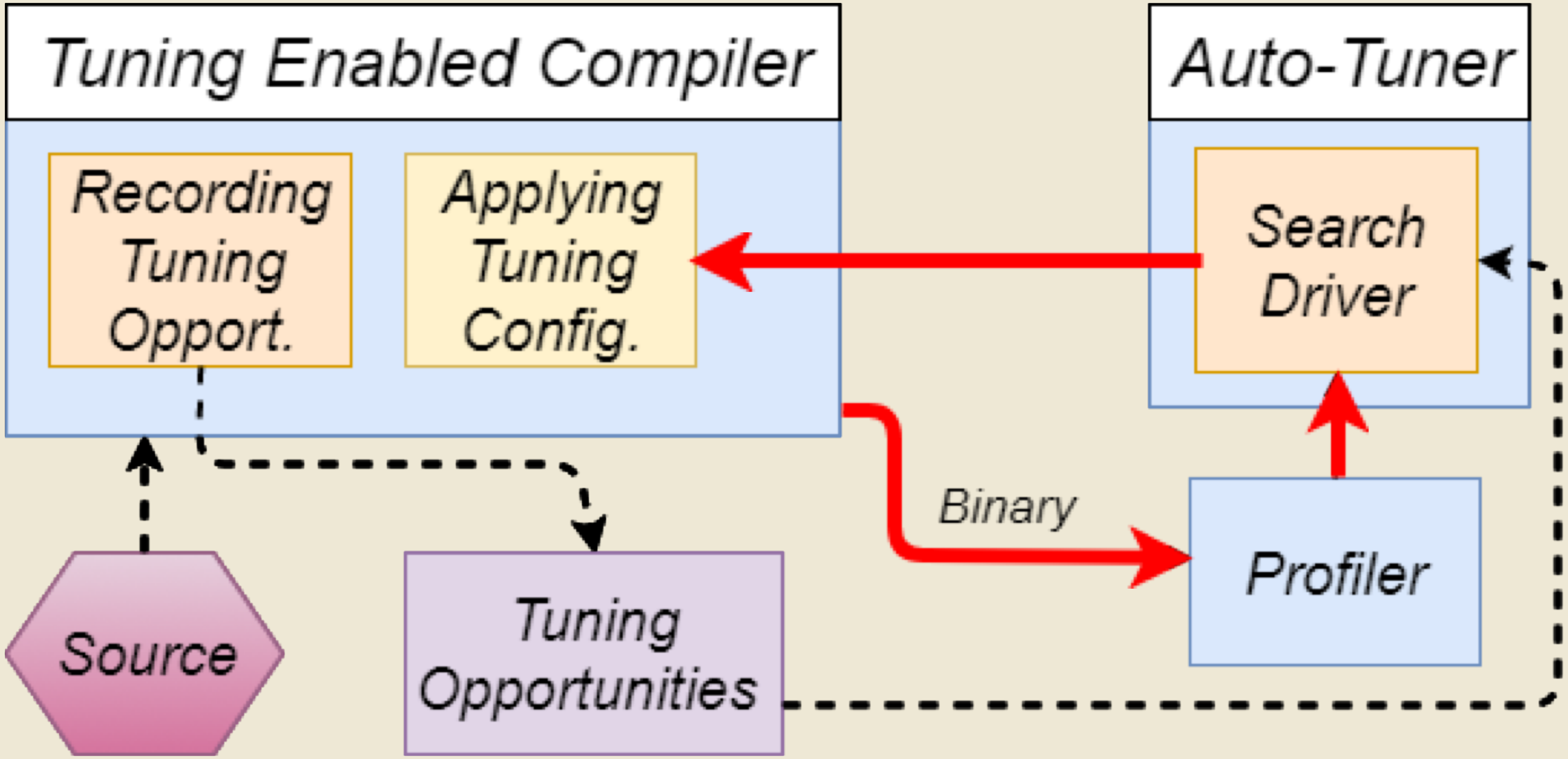


Approach



Up to 19.6% speedup over standard optimization and 11.5% over coarse grained tuning

High-Level



Tuning Enabled Compiler
Recording Tuning Opport.
Applying Tuning Config.
Auto-Tuner
Search Driver
Binary
Profiler
Source
Tuning Opportunities

Code Region Tuning

- Any segment of IR that can be independently optimized
  - Loops
  - Modules
  - Basic Blocks

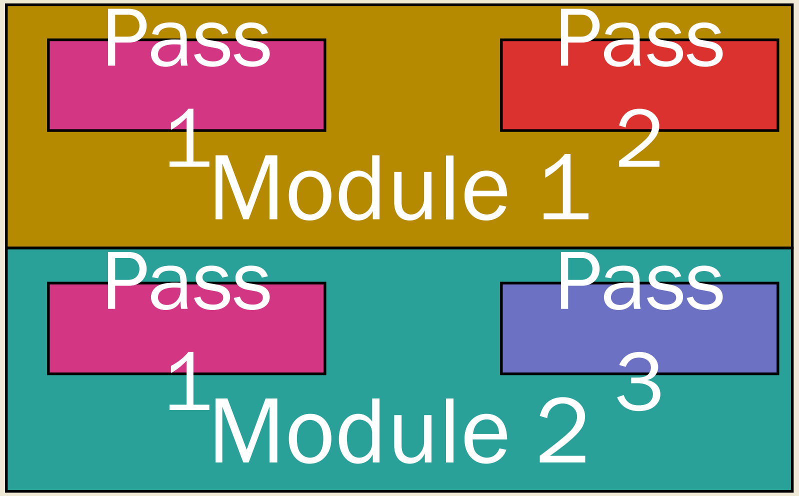
What is a code region?

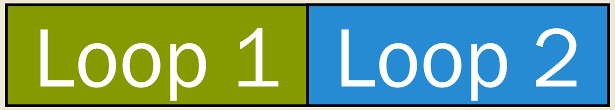
Code Region Based
Auto-Tuning

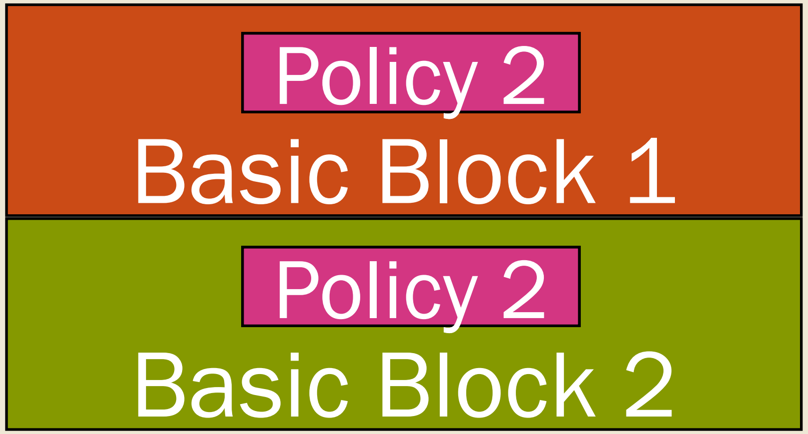
HUAWEI UNIVERSITY OF ALBERTA

Tuning Parameters

- Optimization pass selection/order
- Loop Unroll/peel count
- Machine scheduling policy
- Support for more additional tuning parameters was limited by development time







Code Region Auto-Tuning

- Prerequisites:
  - *Identify the code regions* of a given source and the possible optimizations on those code regions
  - *Auto-tune*: automatically make optimization decisions about the code regions
  - *Apply the optimization* decisions when compiling

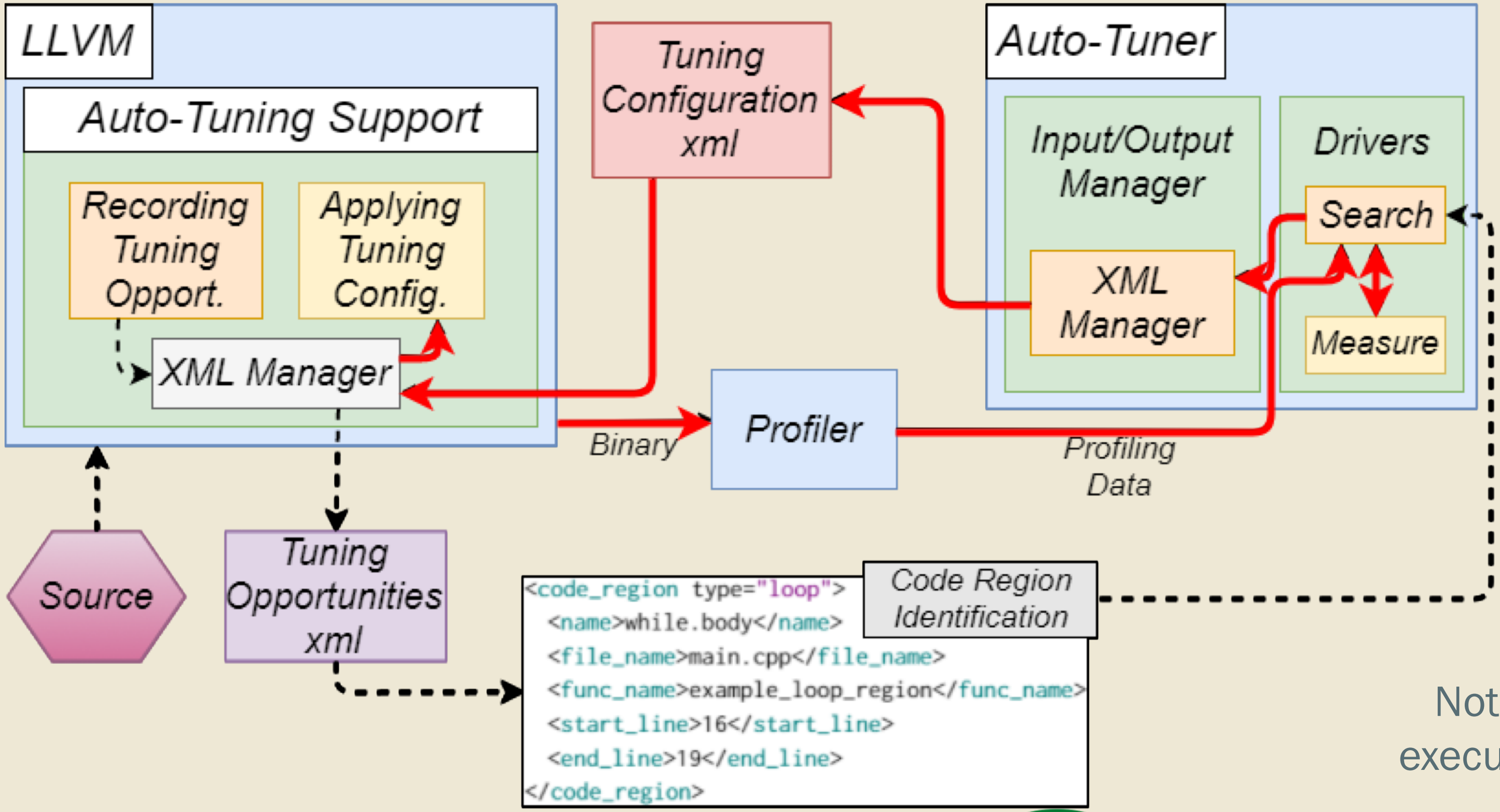
This is what we call *enabling the compiler for auto-tuning*, which is a necessary step for code region based auto-tuning

How to enable auto-tuning on code regions?

HUAWEI UNIVERSITY OF ALBERTA

Code Region Auto-Tuning

(for the diagrammatically inclined)



The tuned binary is compiled and profiled, the performance is given as feedback to the search driver

Note: the dotted lines are executed once per tuning run

HUAWEI UNIVERSITY OF ALBERTA

Methodology

- We built our tuning mechanism using:
  - OpenTuner
  - LLVM 4.0
- Search algorithms: OpenTuner's built-in AUC Bandit meta-technique cycling between:
  - Differential Evolution, Random Nelder-Mead, Greedy Hill Climbing
- Results are shown on the industry benchmarks: CoreMark, HPCG, and Livermore Loops, running on an x86 CPU

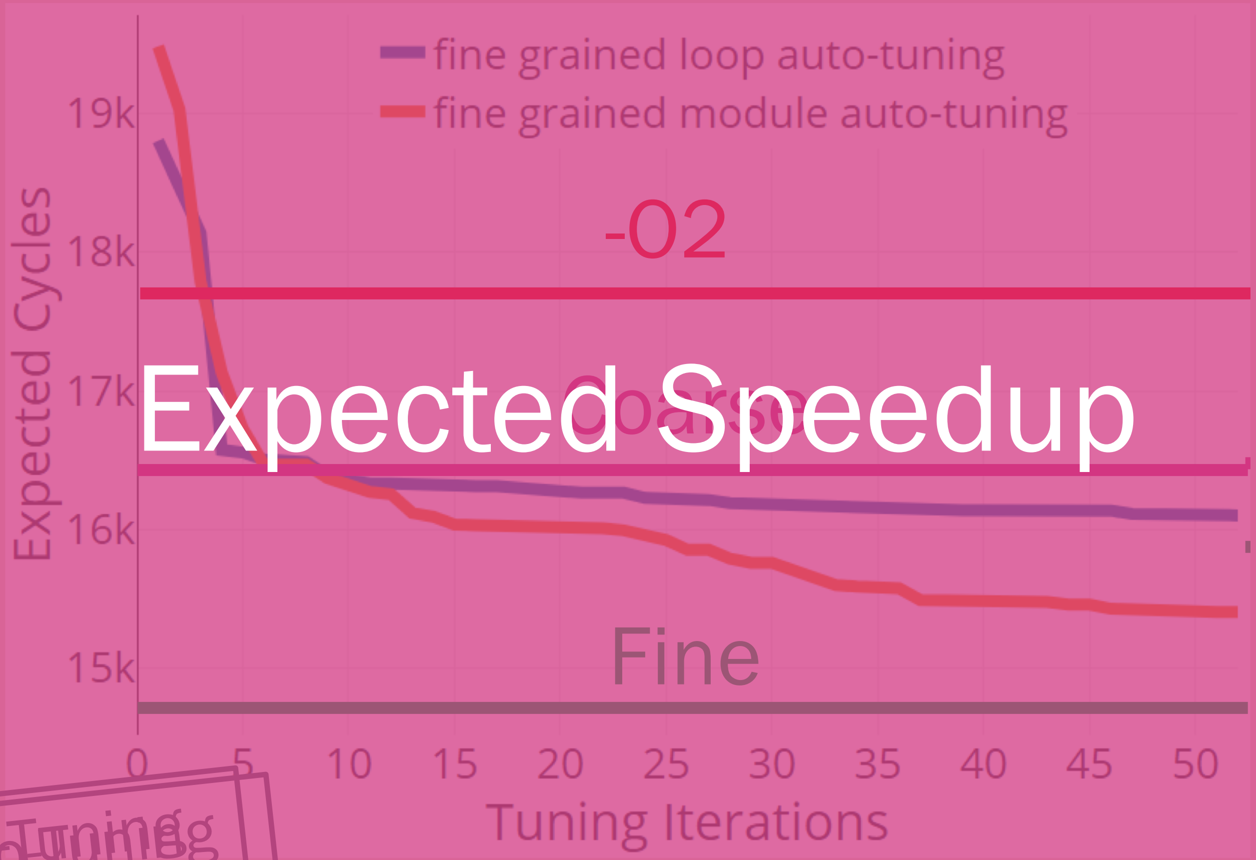
HUAWEI UNIVERSITY OF ALBERTA

Experimental Results (CoreMark)

Name	Description	Coarse Scope	Fine Scope	Best Speedup	
				Over Coarse	Over –O2
Phase ordering	Ordering of optimization passes (LLVM IR)	All modules	Per module	1.115x	1.196x
Loop unrolling/peeling	Factor to unroll/peel loops by (LLVM IR)	All loops	Per loop	1.036x	1.106x
Machine scheduling policy	Scheduling rule for instructions (x86 machine IR)	All basic blocks	Per basic block	1.001x	1.003x

Results for CoreMark on x86

Experimental Results (CoreMark)



Potential Speedup

Expected Speedup

Loop Auto Tuning

Module Auto Tuning

Iteration time =
time(configuration choice)
+ time(compile)
+ time(runtime) ≈ 45s

HUAWEI

UNIVERSITY OF ALBERTA

Experimental Results (others)

- HPCG
  - 5% speedup over coarse grained while tuning loops
- Livermore Loops
  - 2% speedup over coarse grained while tuning loops

HUAWEI UNIVERSITY OF ALBERTA

Related Work

- Code Region Oblivious Auto-Tuning
  - Compiler as a black box
  - Compiler Auto-Tuning Survey (2018)
  - GCC flag tuning with CK-autotuning framework
- Isolated Code Region Based Auto-Tuning
  - Predicting Unroll Factors Using Supervised Classification
- Code Region Based Auto-Tuning
  - Region-Aware Multi-Objective Auto-Tuner for Parallel Programs (2017)
  - Code region based thread count tuning for parallelization

HUAWEI UNIVERSITY OF ALBERTA

Limitations/Future Work

- Have not identified/implemented many code regions or fine grained optimizations
  - Support more code region types and optimizations
- Optimizations disrupt the IR—can lose track of CRIDs
  - Auto-tuning stages
- Iterative compiler auto-tuning is time-expensive and must be done per program
  - RNN/RL approach for predicting compiler configurations

A new host of challenges

HUAWEI UNIVERSITY OF ALBERTA

Future Work: Predictive Tuning Challenges

- Predict configurations for code regions of <u>arbitrary type</u>
  - Features to describe any code region (while minimizing noise)
- Feature extraction (encompass code region and program info)
- Label vectors of variable size (pass sequences)
- Stage based tuning is remaining issue

HUAWEI UNIVERSITY OF ALBERTA

Summary

- Problem:
  - Current compiler auto-tuning methods are missing out on performance peaks
- Approach:
  - Enabled code region based (fine grained) tuning within the compiler
- Results:
  - Observed speedup over standard optimization and coarse grained tuning