Graph-Matching-Based Simulation-Region Selection for Multiple Binaries

Charles Yount and Harish Patil, Intel Corporation
Mohammad S. Islam, Univ. of Texas, San Antonio
Aditya Srikanth, Univ. of Texas, Austin

ISPASS-2015
2015 IEEE International Symposium on Performance Analysis of Systems and Software
March 29-31, 2015
Hilton At Penn's Landing, Philadelphia, PA

Motivation and problem statement

Evaluate pre-Si performance differences between *n* binaries compiled from the same source

Application examples

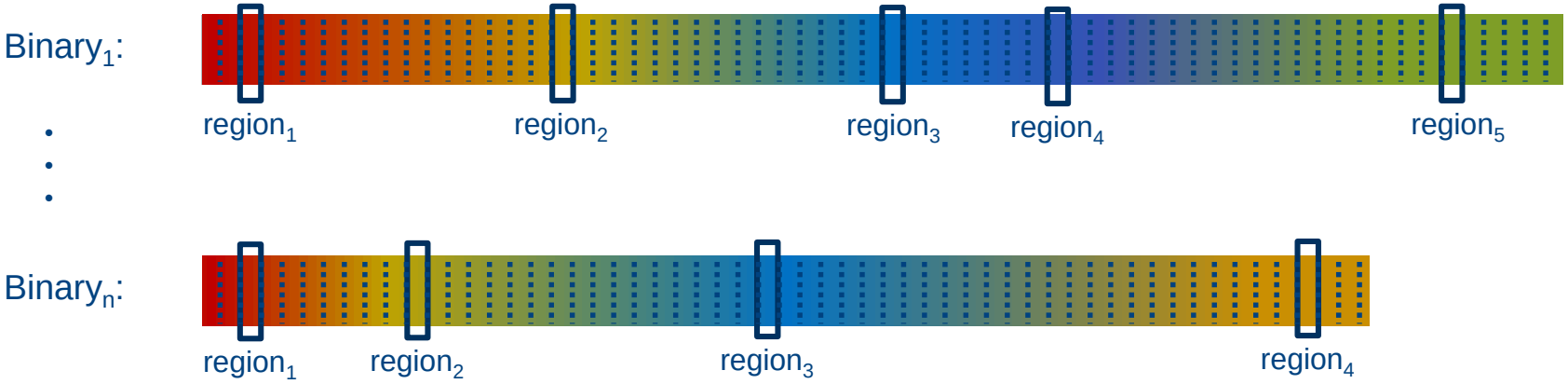
- Compare compilers and/or compiler optimizations
- Compare impact of different macro architectures and/or instruction set extensions

Requirements

- Find a set of representative simulation regions from each binary such that
  - Resulting speedup estimates are accurate
  - All regions are feasible for simulation
  - Regions are matched across binaries, representing the same semantic work in each
- Tolerate significant differences between binaries due to different optimizations, instruction sets, etc.
- Do not require debug symbols or modification of source code

Independent-SimPoint approach

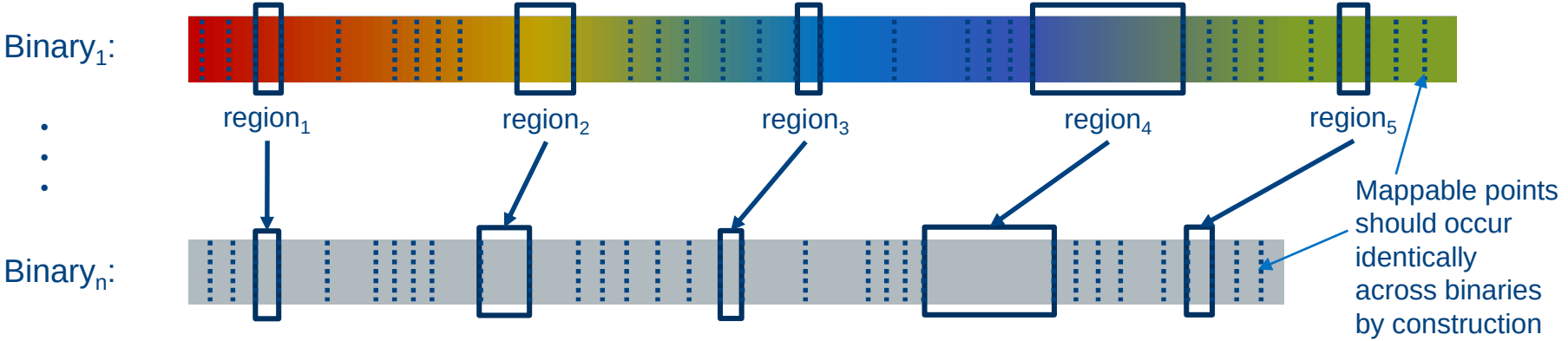
- Divide each binary execution into intervals of equal lengths
- Find phases and representative regions using k-means clustering independently for each binary



- Pro: all simulation regions are near desired length
- Con: regions differ both in number and semantic representation across binaries
- Con: speedup prediction can be unacceptable and minimal performance debug capability

Original Cross-Binary SimPoint (CBSP) solution*

- Find “mappable” routine and loop entry points where symbols and counts are *identical* across binaries
- Divide each binary execution into intervals of *variable lengths* bounded by these points
- Find phases and representative regions in $binary_1$, and map them to the other binaries

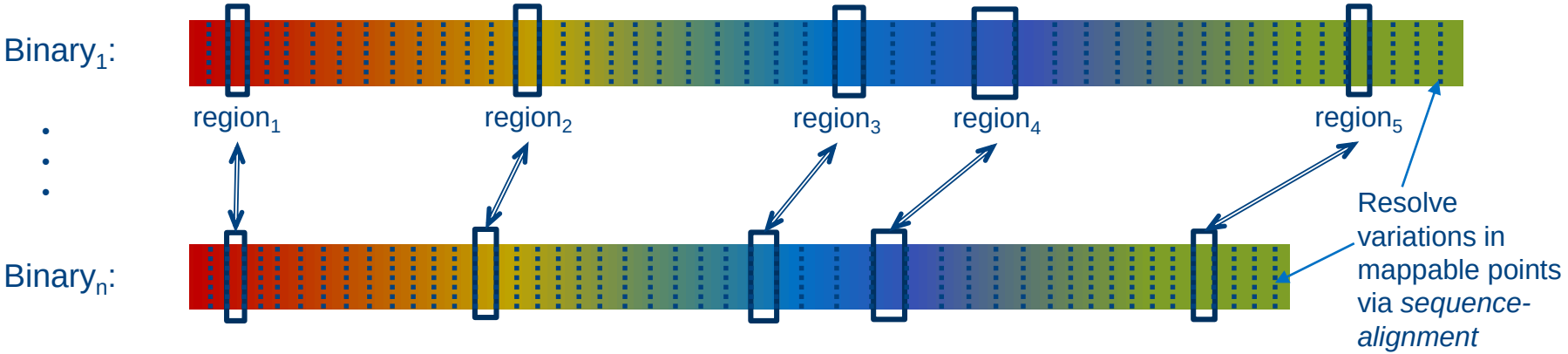


- Pro: regions are same in number and semantic representation, improving speedup and debugging
- Con: requires symbol information and limited variations in binaries
- Con: clustering accounts for variations over $binary_1$ execution only
- Con: simulation regions can be unfeasibly long due to strict mapping rules

*Perelman, Lau, Patil, Hamerly, Calder, Jaleel; ISPASS-07

New Cross-Binary SimPoints solution

- Find all possible “mappable” routine and loop entry points across binaries via *graph-matching*
- Divide each binary execution into intervals of (less) variable lengths bounded by these points
- Find phases and representative regions in unified profile across all binaries



- Pro: relaxes requirement for symbols and allows more divergent binaries
- Pro: clustering accounts for variations across the execution of all binaries
- Pro: alleviates region-length issue
- Enabled by applying new graph-matching and sequence-alignment algorithms…

Graph-matching

Dynamic Control-Flow Graph (DCFG) is created from execution of each binary

- Applying graph-matching to entire DCFG was found to be ineffective
- So, DCFG for each binary is decomposed hierarchically
  - One top-level call graph: each routine is a node; calls are edges
  - One sub-graph for each routine: each loop is a node; dominance (including nesting) defines edges
  - Graph-matching is applied to top-level call graph and then to each loop graph in matching routines
- Algorithm matches nodes, minimizing differences in graph topology and node meta-data

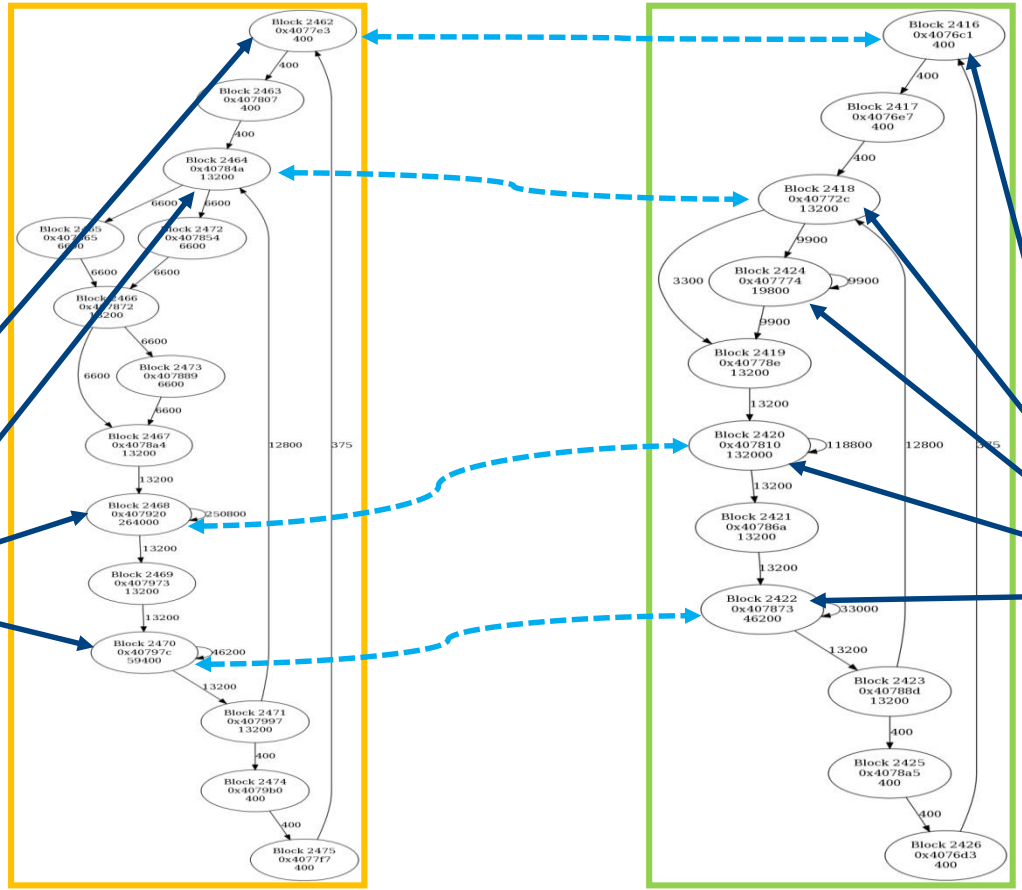
Meta-data difference factors between any two nodes

- Edit distance between symbol names, if available
- Symmetric difference between sets of line numbers, if available
- Difference in execution counts (calls for routines, iterations for loops)
- Difference between in-degrees and out-degrees

Loop-matching example from 410.bwaves

3 nested loops from binary A compiled with SSE4.2 instruction set (128-bit SIMD)

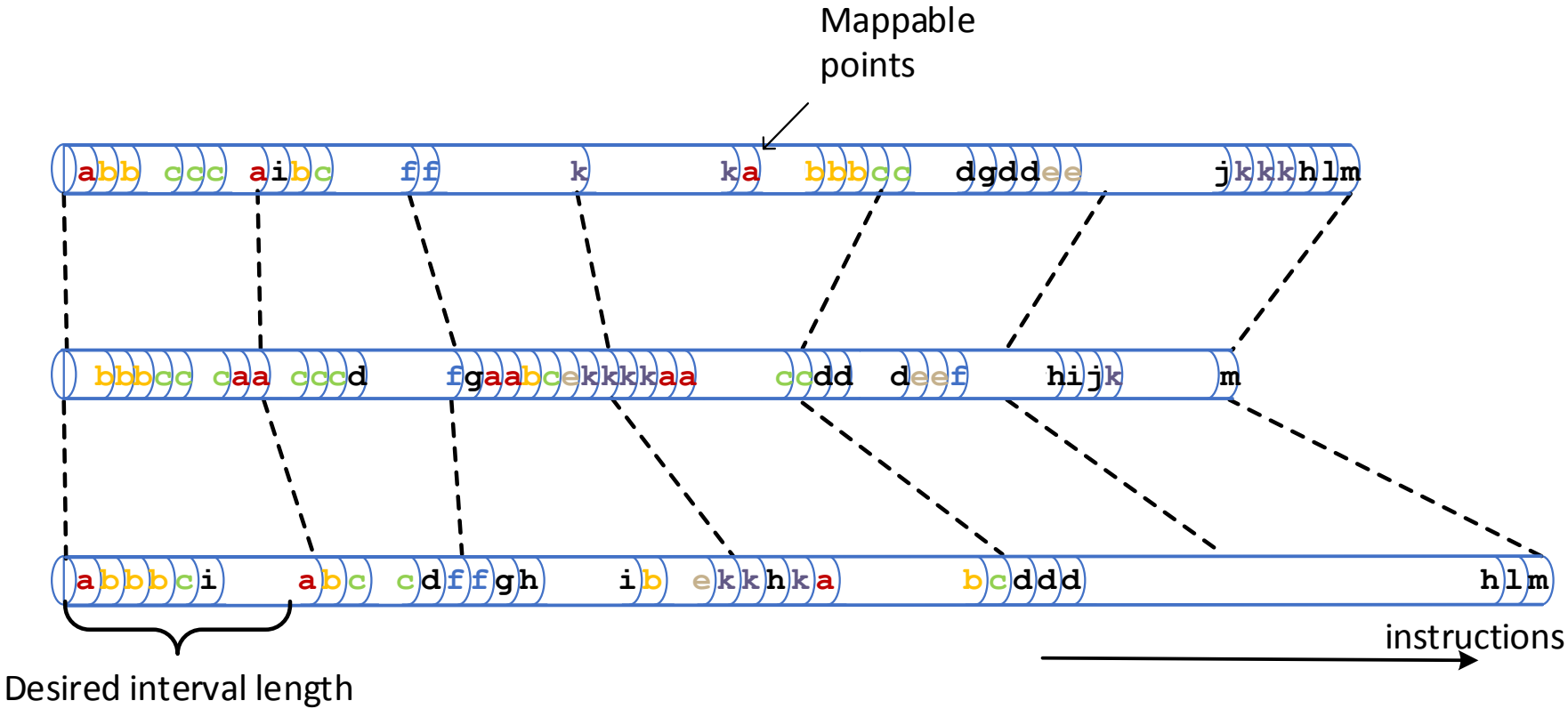
Iterations:
- Outer: 400
- Middle: 13,200
- Inner: 264,000
- Epilog: 59,400



3 nested loops from binary B compiled with AVX2 instruction set (256-bit SIMD)

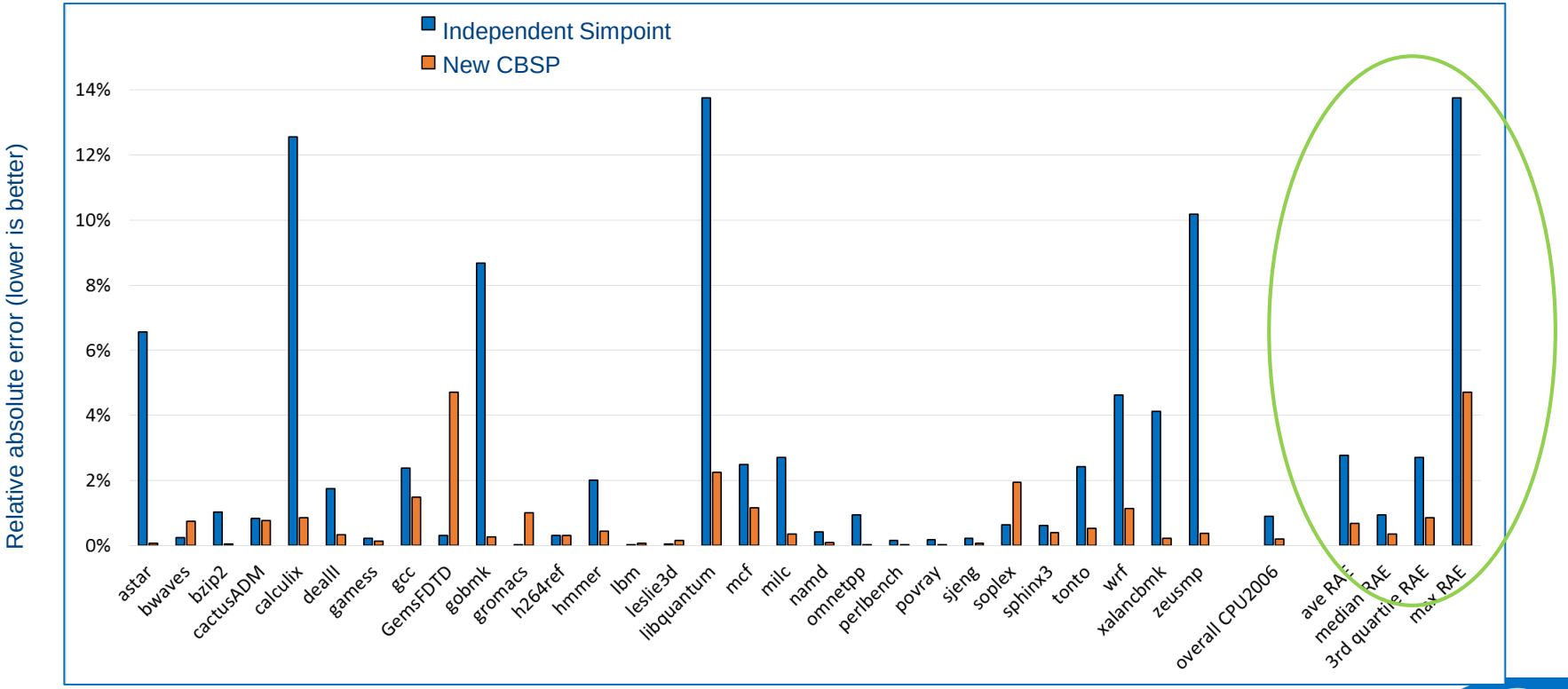
Iterations:
- Outer: 400
- Middle: 13,200
- Prolog: 19,800
- Inner: 132,000
- Epilog: 46,200

Sequence alignment



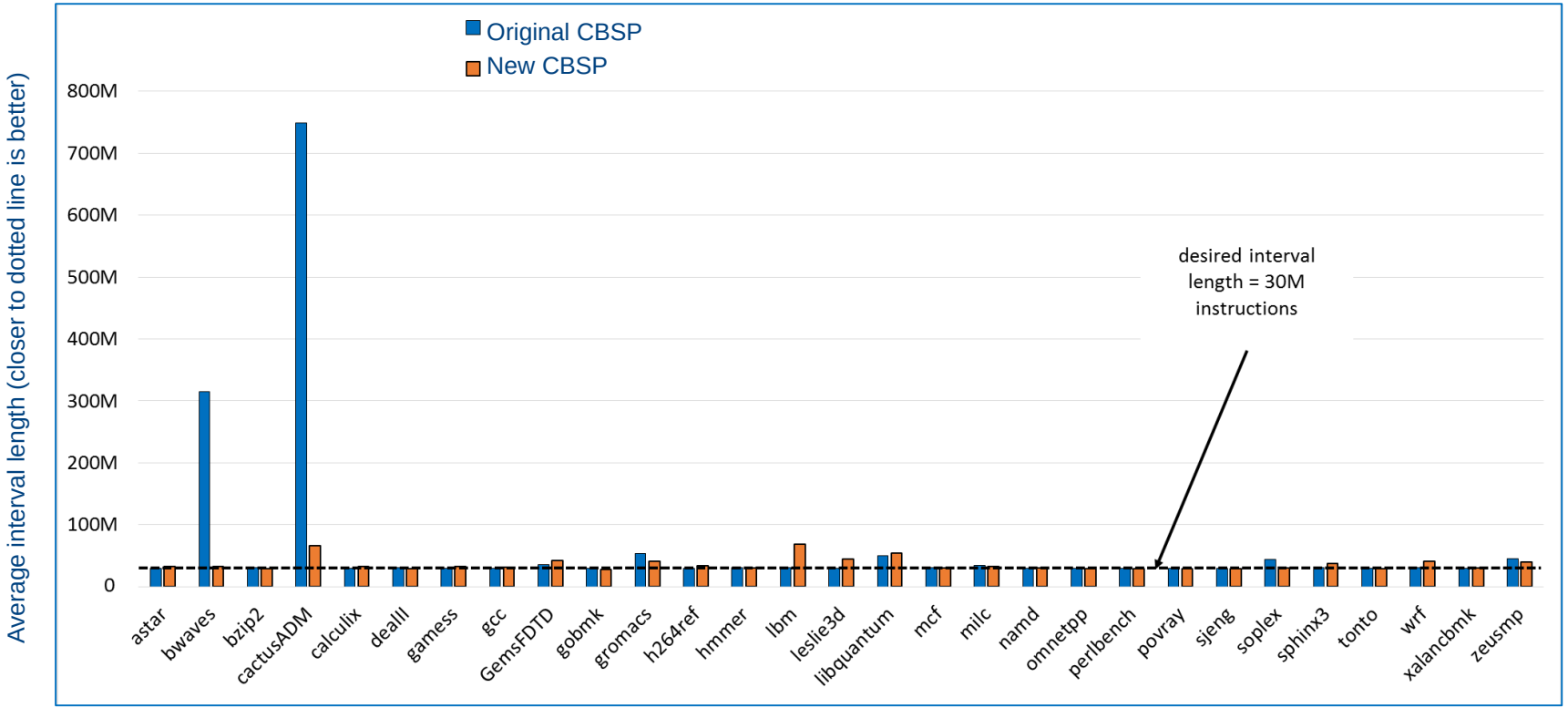
Quality of independent SimPoint vs. new CBSP

- AVX2-to-MICAVX512 speedup on CPU2006 with Intel compiler



Interval-length of original CBSP vs. new CBSP

- AVX2-to-MICAVX512 speedup on CPU2006 with Intel compiler



Summary of new Cross-Binary SimPoint method

Benefits compared to independent SimPoint

- Lower speedup-estimation error
- Paired equal-work regions and graph-matching data can be used for performance debug

Benefits compared to original CBSP work

- Graph-matching enables finding more mappable points: without symbols, between routines and loops with different call and iteration counts, etc.
- Alleviates simulation-length issue

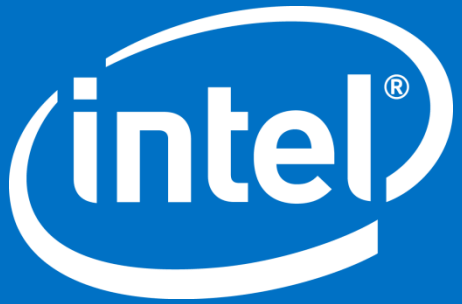
To follow up

For more information on Pin and PinPlay

- Visit http://pintool.org/
- Visit http://pinplay.org/
- Attend PinPlay tutorial at PLDI in Portland, OR, June 14, 2015
  - We plan to cover DCFG generation in this tutorial

For slides, questions, suggestions, information on availability of software

- Email chuck.yount@intel.com
- Email harish.patil@intel.com

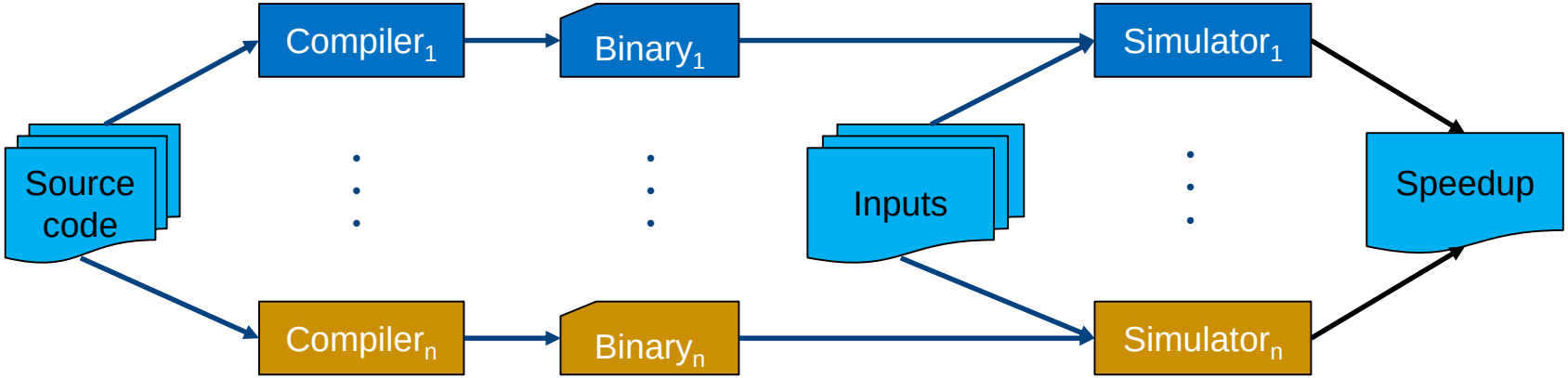

intel®

Backup

Problem statement

Efficiently and effectively compare the pre-Si performance of *n* binaries (run with the same inputs) compiled differently from the same source code

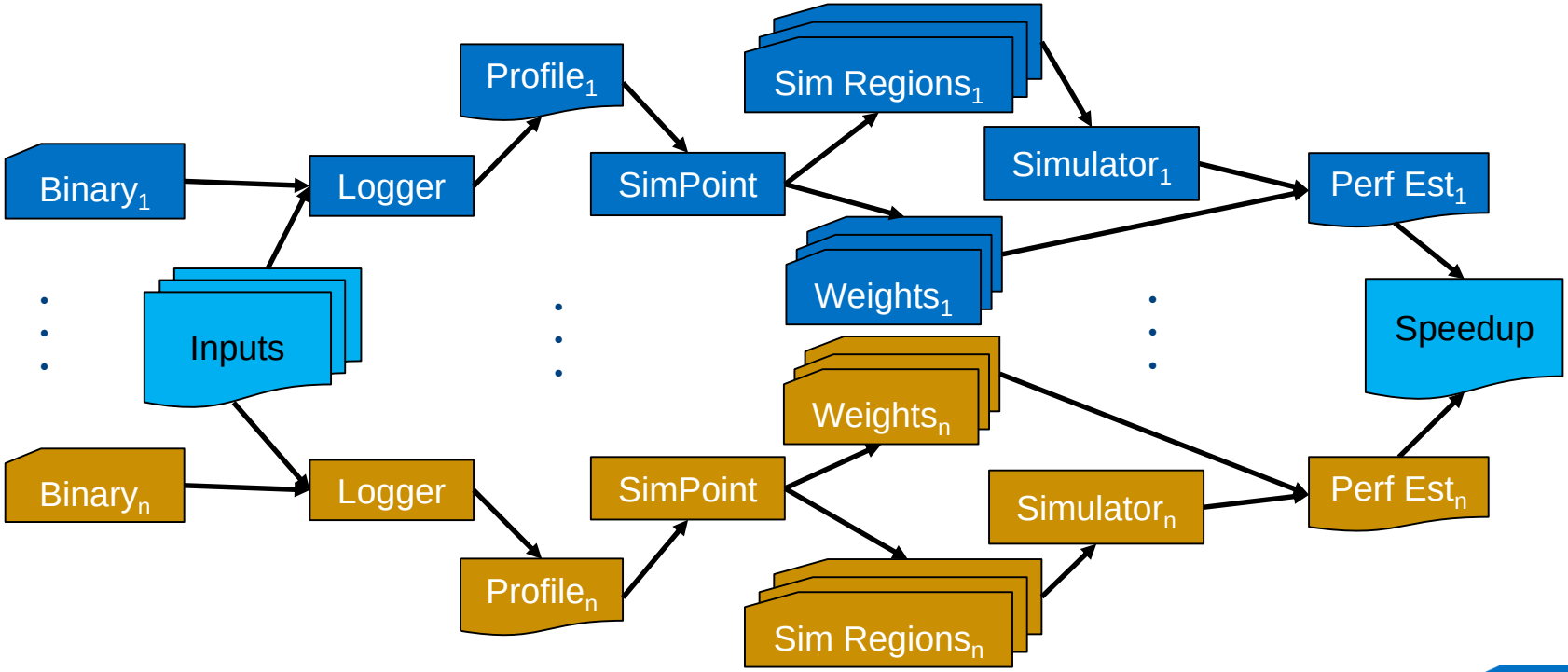
- Conceptual flow:



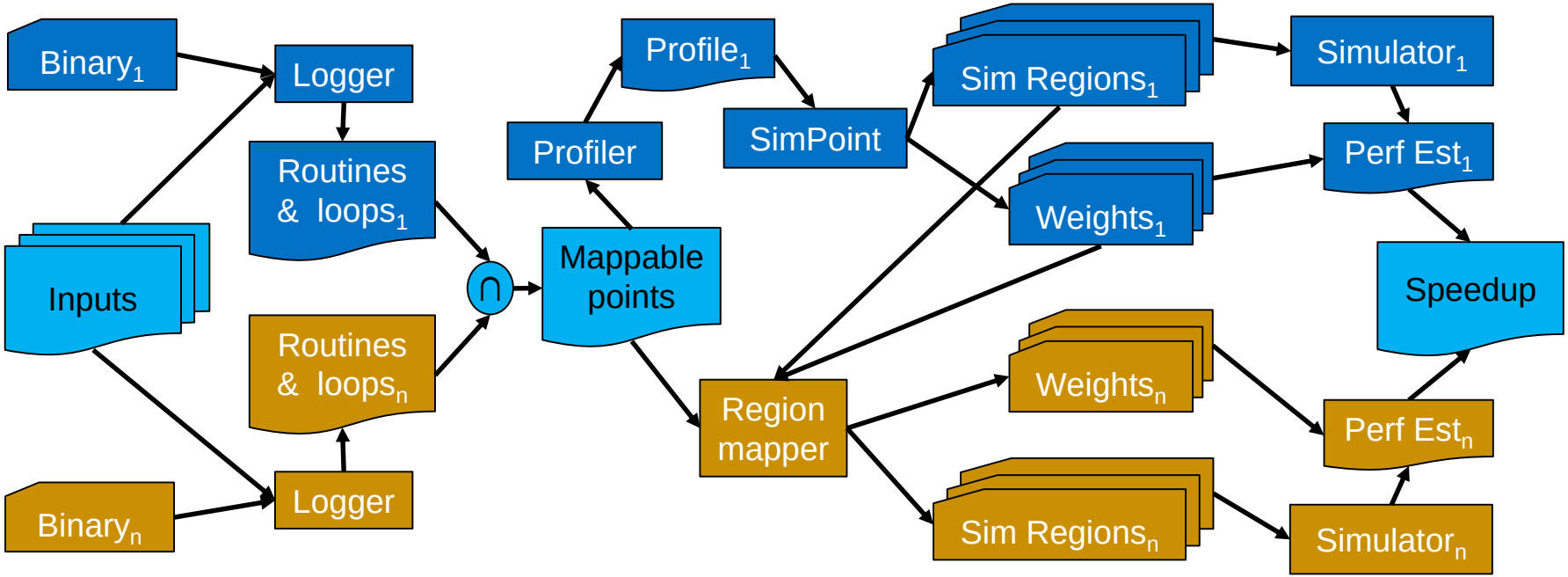
- Typical pre-Si limitation: cannot simulate entire run due to low simulation speed
- Need to find representative samples of the execution to simulate

Straight-forward SimPoint implementation

Run popular SimPoint simulation-region selection tool on each binary separately

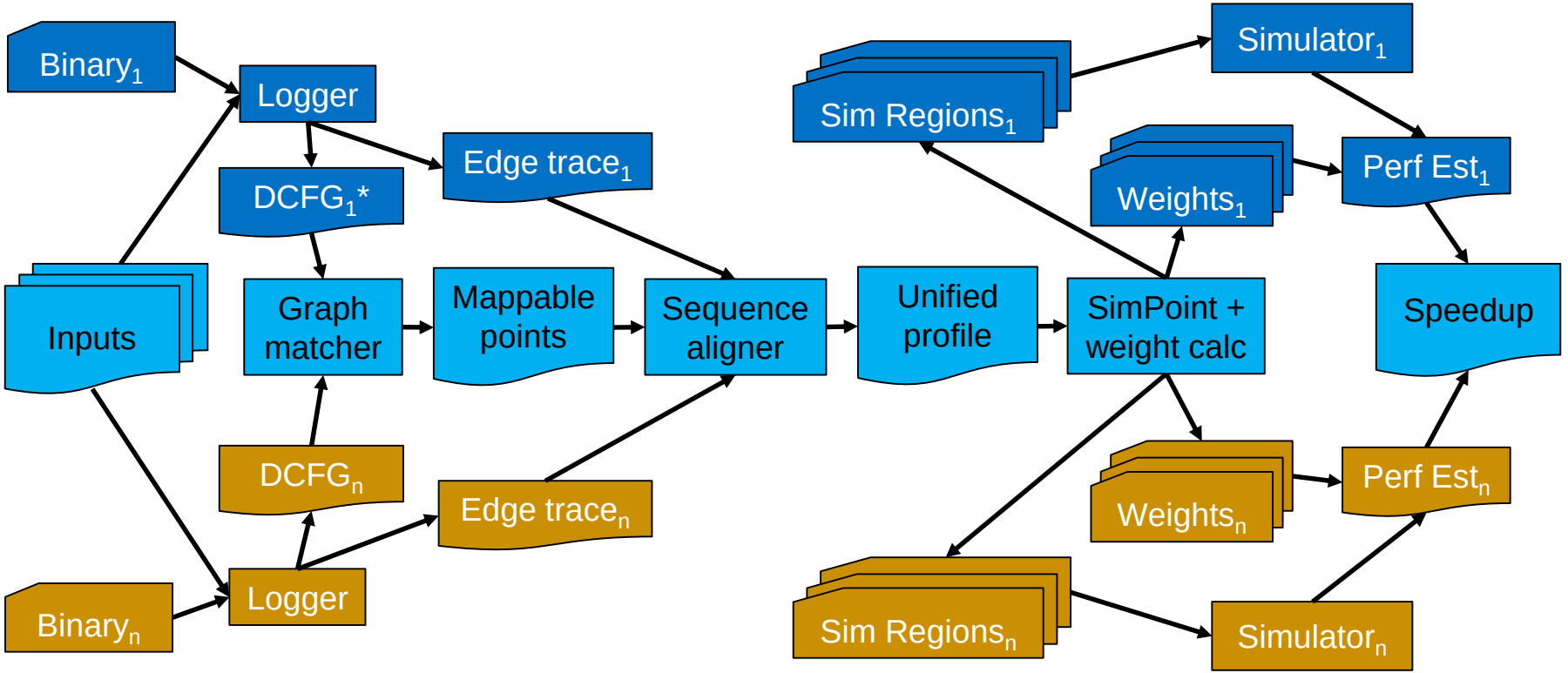


Original Cross-Binary SimPoint* implementation



*CBSP: Perelman, Lau, Patil, Hamerly, Calder, Jaleel; ISPASS-07

New Cross-Binary SimPoint implementation

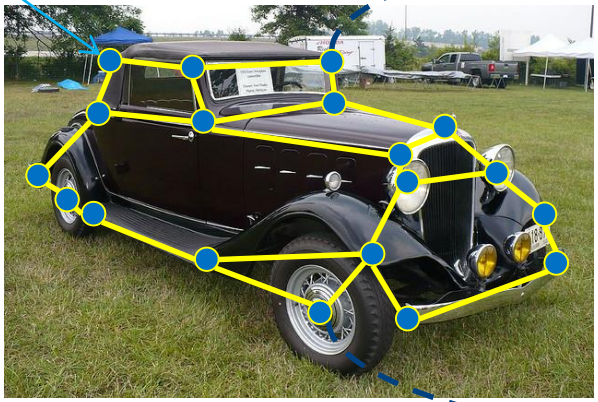


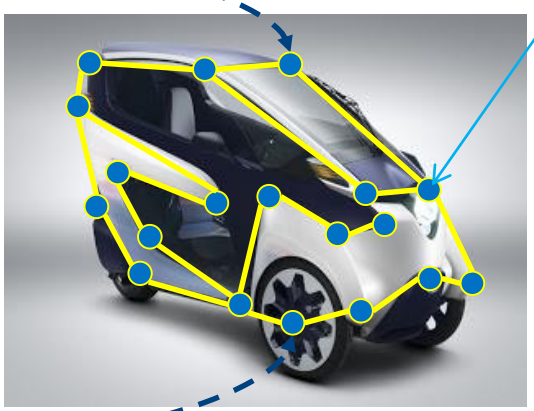
*DCFG: Dynamic Control-Flow Graph (CFG + execution counts)

Graph-Matching concept

Technique used in computer vision and other fields

Color=black
Object=corner





Color=white
Object=oval

- Minimize meta-data differences between matched nodes (with weight α)
- Minimize topological differences between matched edges (with weight 1-α)

Sequence-alignment

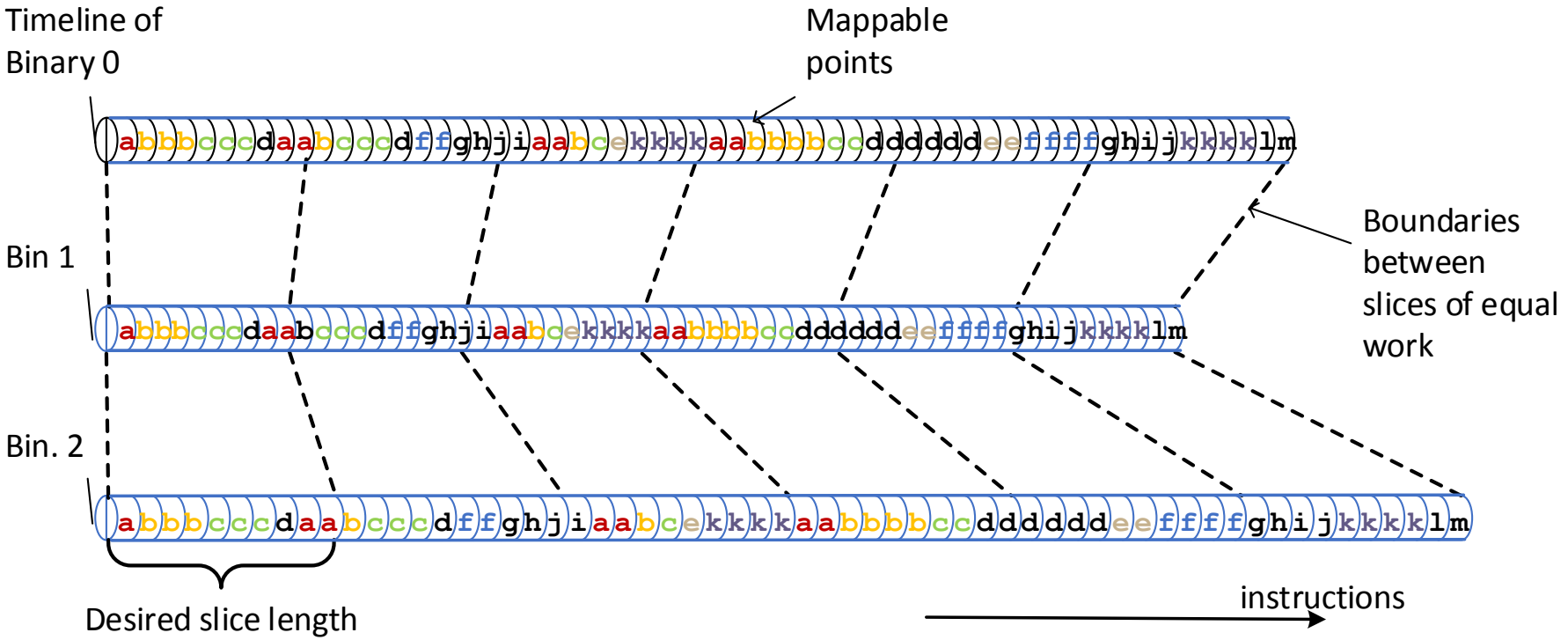
Divide the execution trace of each binary into intervals

- Use the graph-matching data to divide intervals so that each matching set of intervals across the binaries represents [approximately] the same work
- Create the same number of intervals in each of the *n* binaries
- Target the length (number of instructions executed) of each interval to be near a target set by the user
- Output a frequency-vector file for SimPoint containing routine and loop counts in each interval across all binaries

Heuristics required to handle differences in execution due to

- Different compiler optimizations: loop unrolling, loop reordering, in-lining, etc.
- Different instruction-set architectures, libraries, etc.
- Different SIMD vector widths, masking vs. conditional code, etc.

Ideal sequence-alignment



Experimental methodology

- Compile each CPU2006 benchmark with Intel® compiler at “O3” optimization
  - One binary using “-xCORE-AVX2” (256-bit vectors)
  - One binary using “-xMIC-AVX512” (512-bit vectors, masking, more new features)
- Determine *actual* AVX512/AVX2 speedup for each “ref” benchmark by executing each binary on the CMP$im Pin tool and dividing the number of cycles from the AVX512 run by that of the AVX2 run
- Determine quality of new CBSP technique
  - Create DCFG, edge-trace and whole-program logs for each benchmark using a PinPlay-enabled Pin tool
  - Apply graph-matching and sequence alignment on each using target length of 30M instructions
  - Run SimPoint tool on profile and run CMP$im on each SimPoint-selected region
  - Calculate *estimated* AVX512/AVX2 speedup using simulation regions and weights
  - Calculate relative absolute error (RAE) between actual and estimated speedup
- Determine RAE of independent SimPoint using similar calculations for quality comparison
- Create simulation regions using original CBSP technique for interval-length comparison