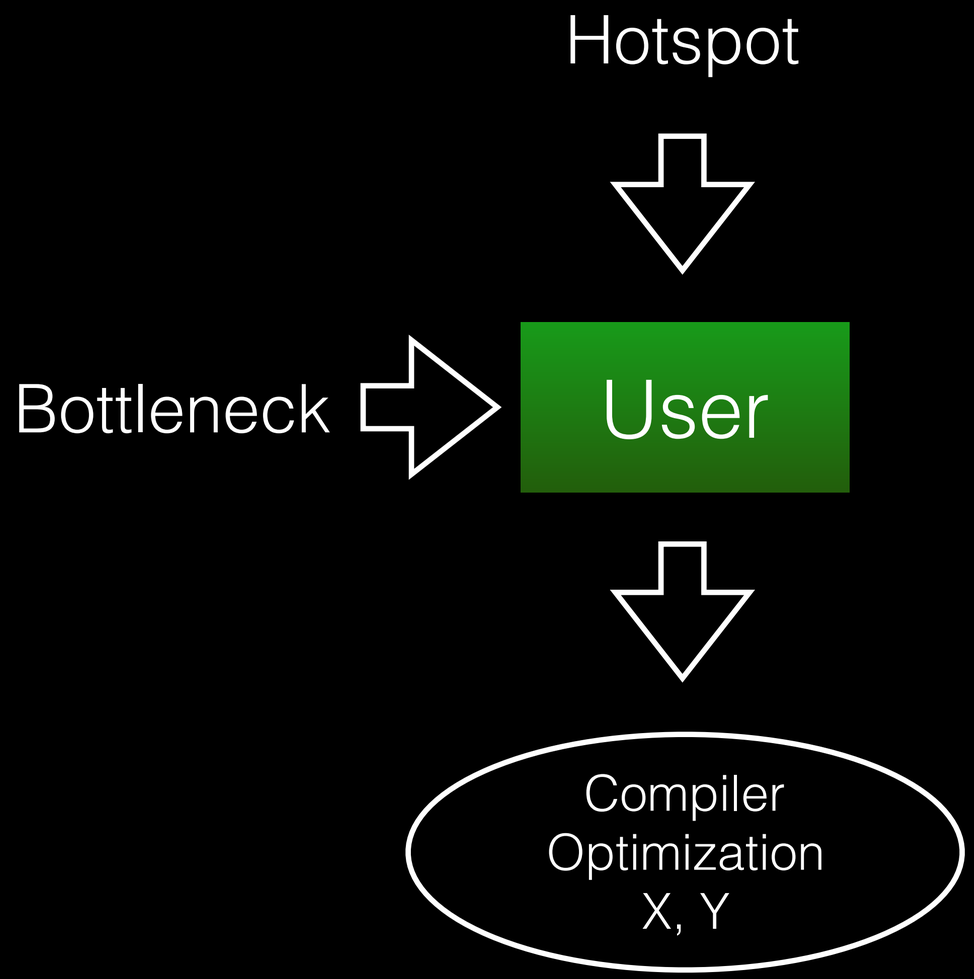
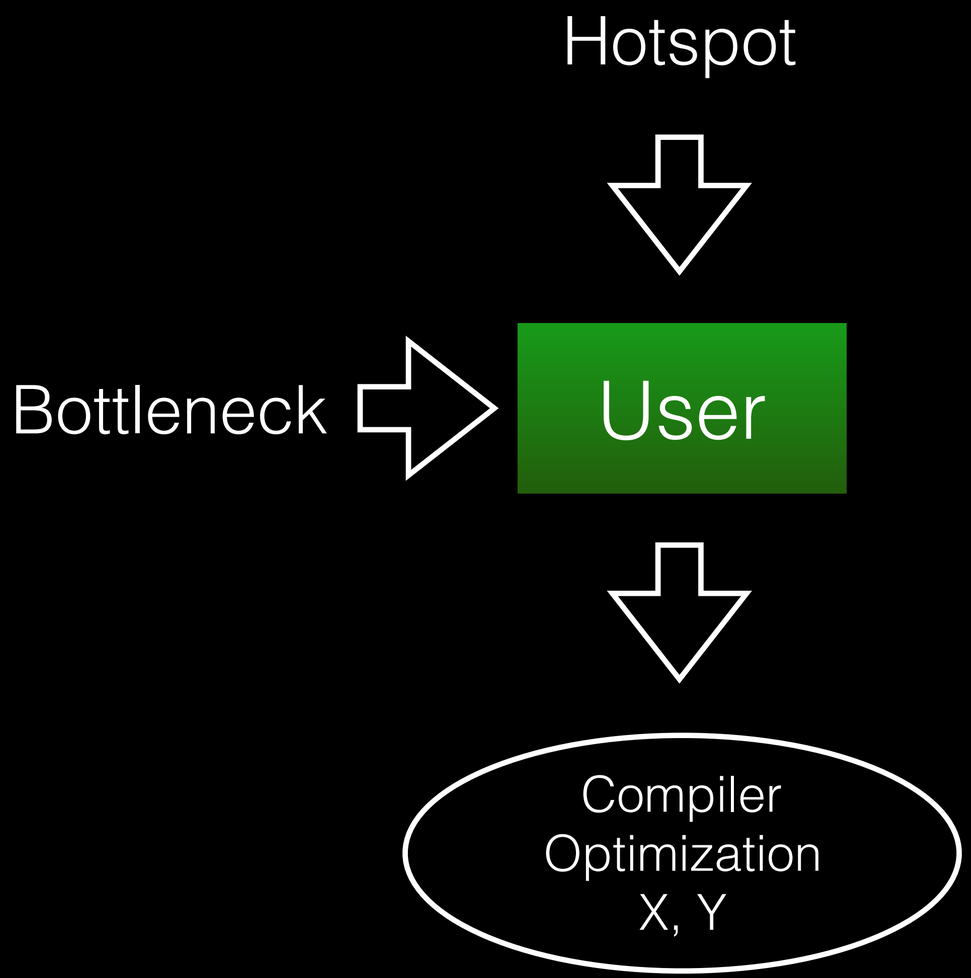
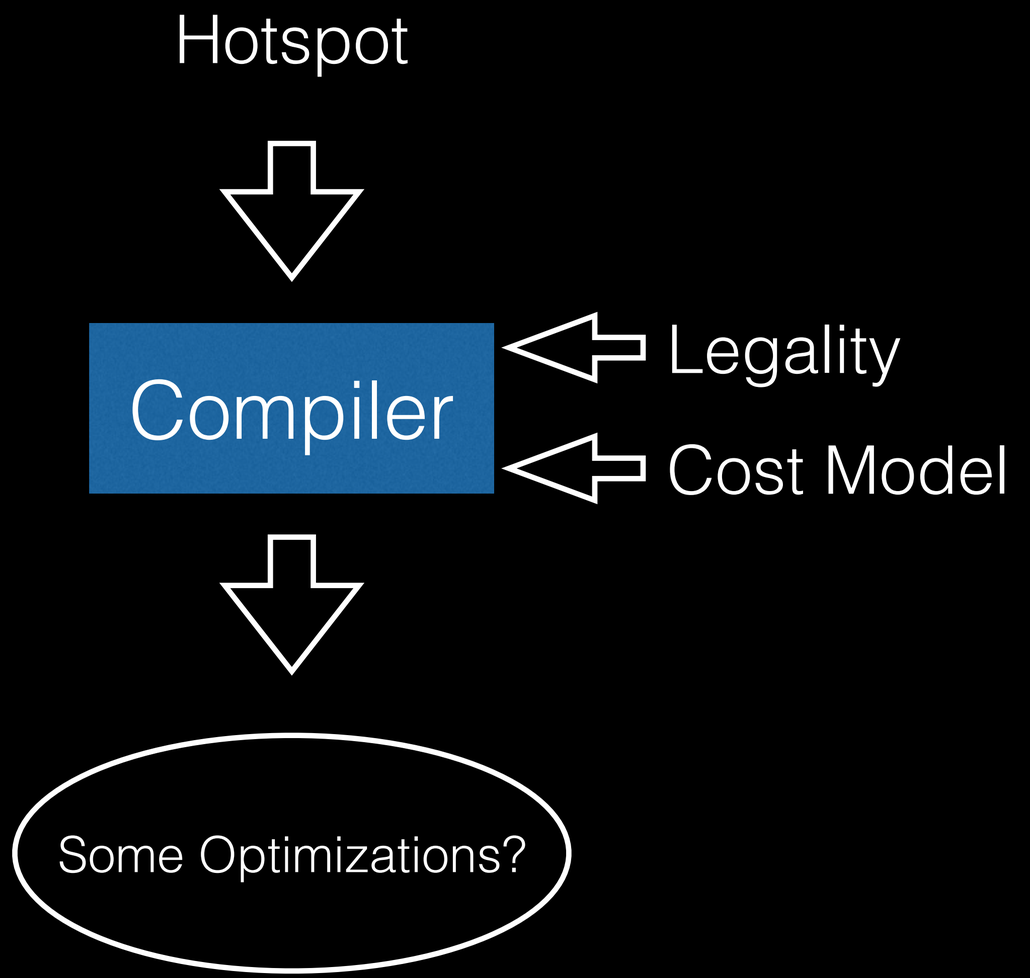
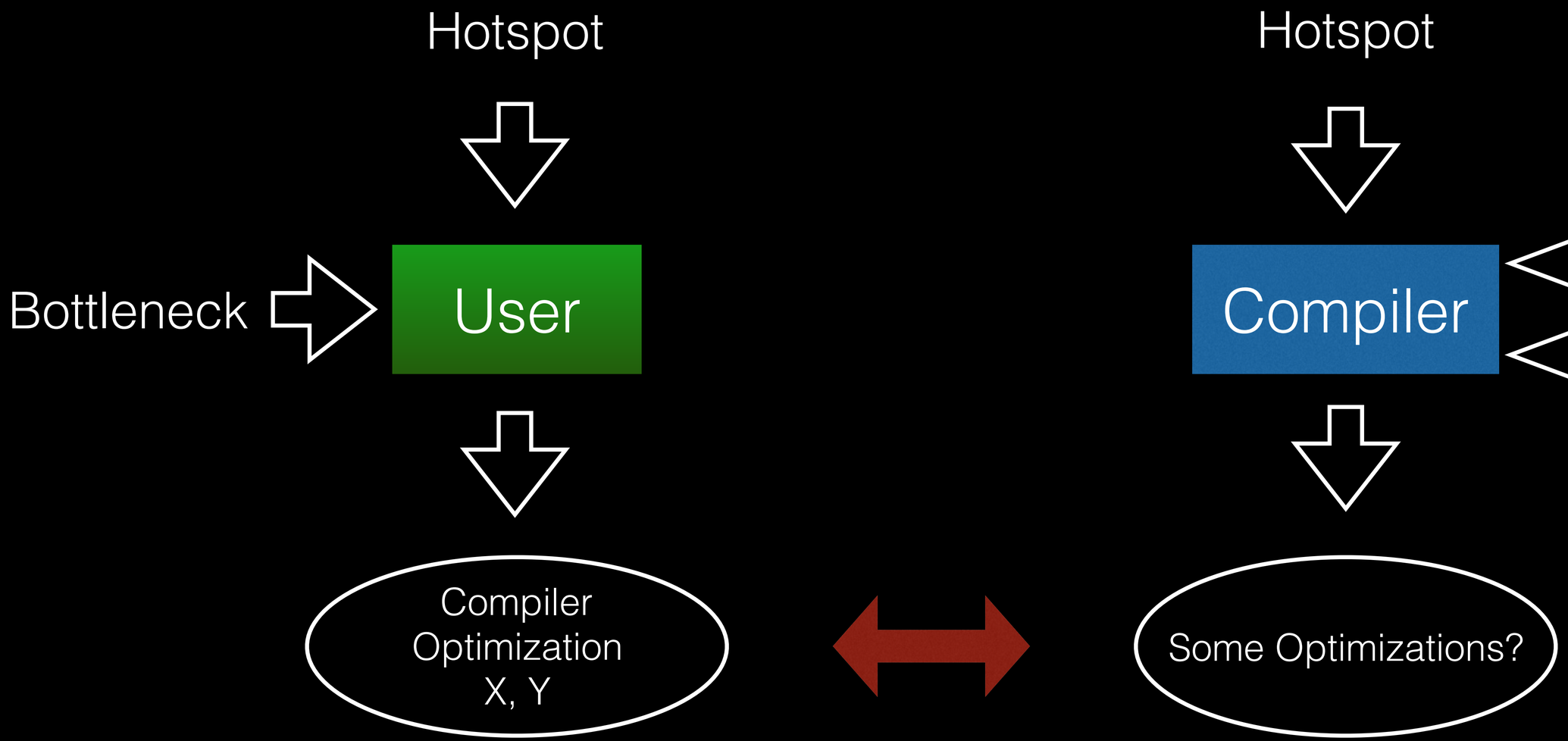
Compiler-assisted Performance Analysis

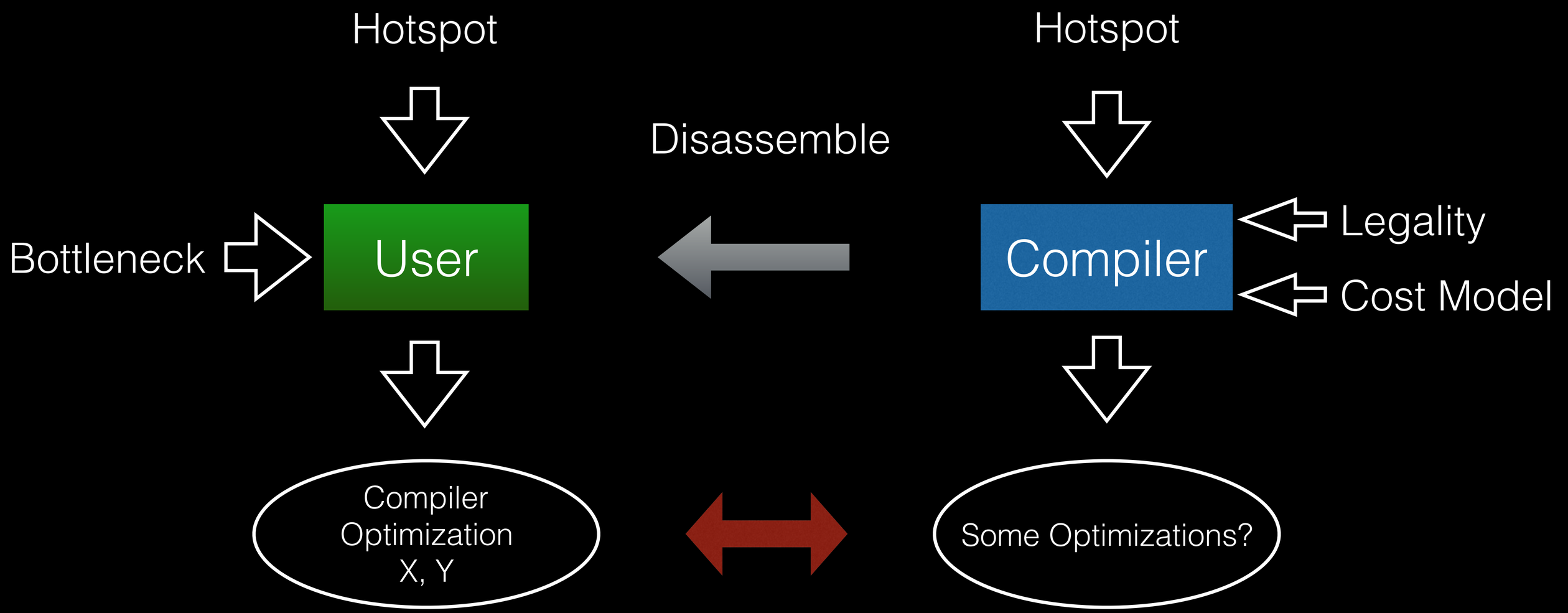
Adam Nemet
Apple
anemet@apple.com

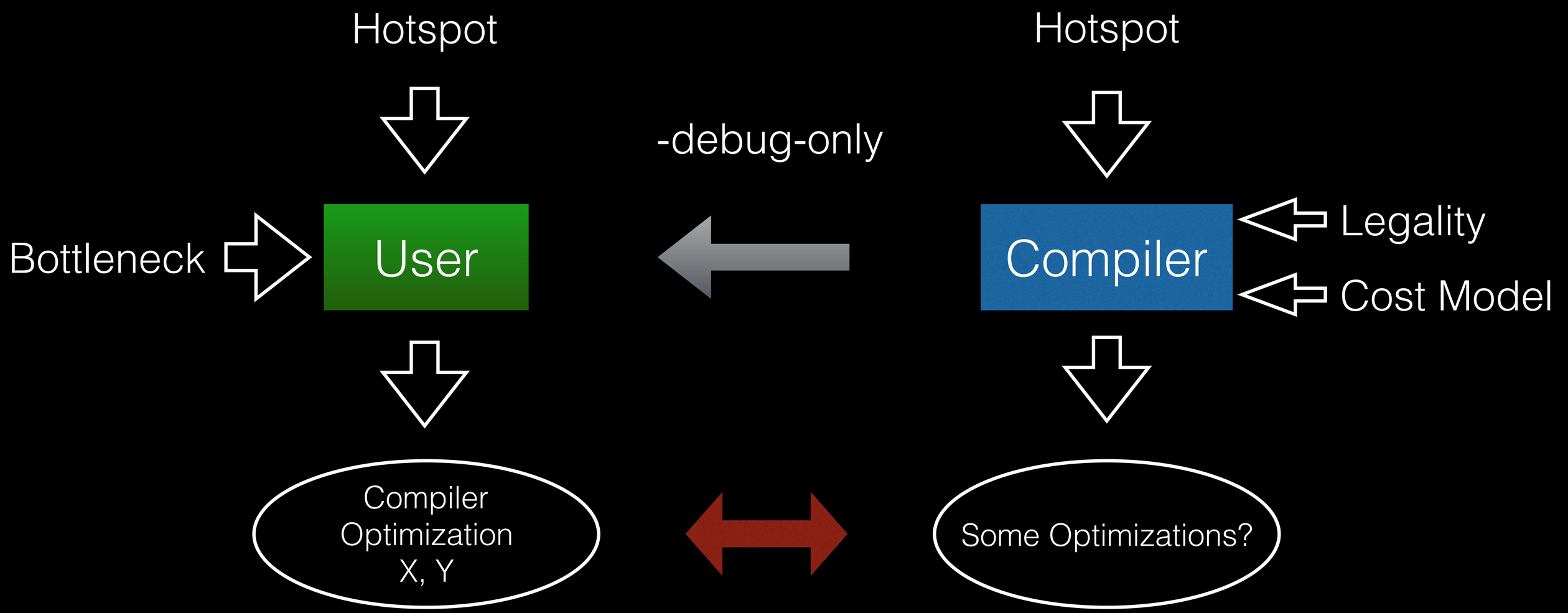

Hotspot
Bottleneck
User
Compiler
Optimization
X, Y

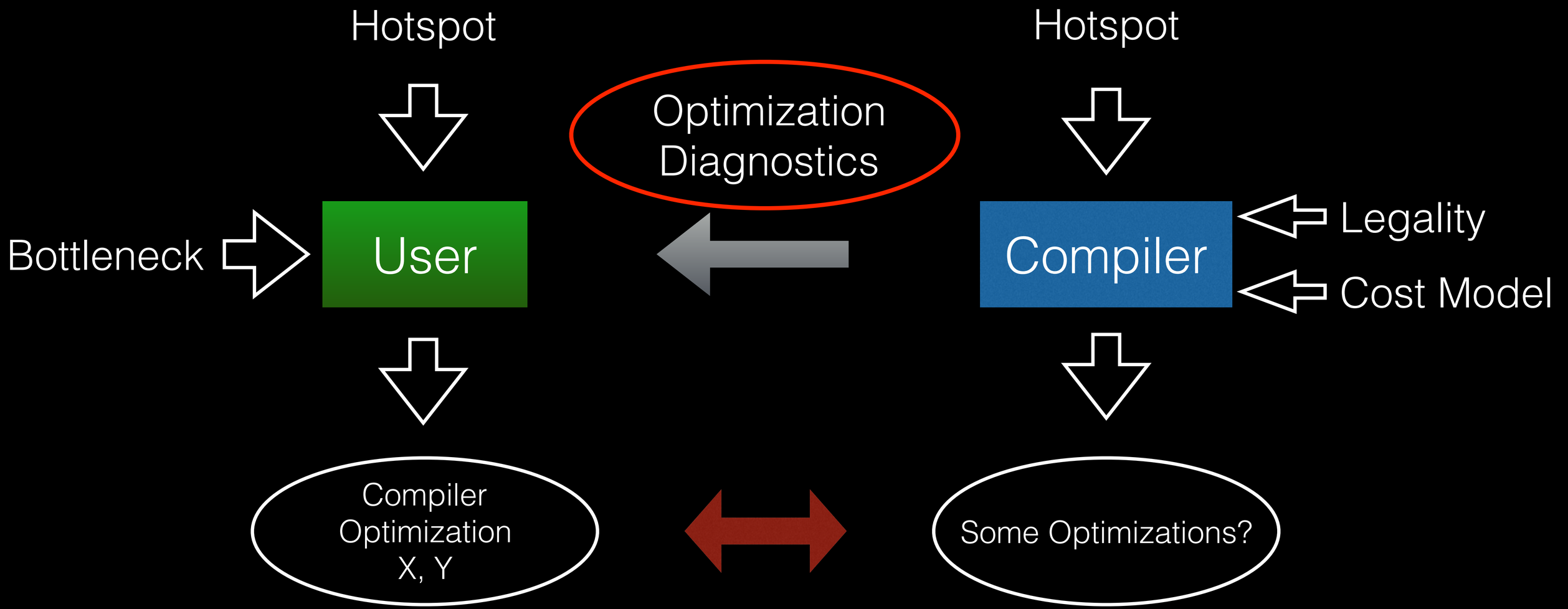

Hotspot
Bottleneck
User
Compiler
Optimization
X, Y


Hotspot
Compiler
Legality
Cost Model
Some Optimizations?


Hotspot
Bottleneck
User
Compiler
Optimization
X, Y
Hotspot
Compiler
Legality
Cost Model
Some Optimizations?


Hotspot
Hotspot
Disassemble
Bottleneck
User
Compiler
Legality
Cost Model
Compiler
Optimization
X, Y
Some Optimizations?


Hotspot
Hotspot
-debug-only
Bottleneck
User
Compiler
Legality
Cost Model
Compiler
Optimization
X, Y
Some Optimizations?


Hotspot
Hotspot
Optimization
Diagnostics
Bottleneck
User
Compiler
Legality
Cost Model
Compiler
Optimization
X, Y
Some Optimizations?

Optimization Diagnostics in LLVM

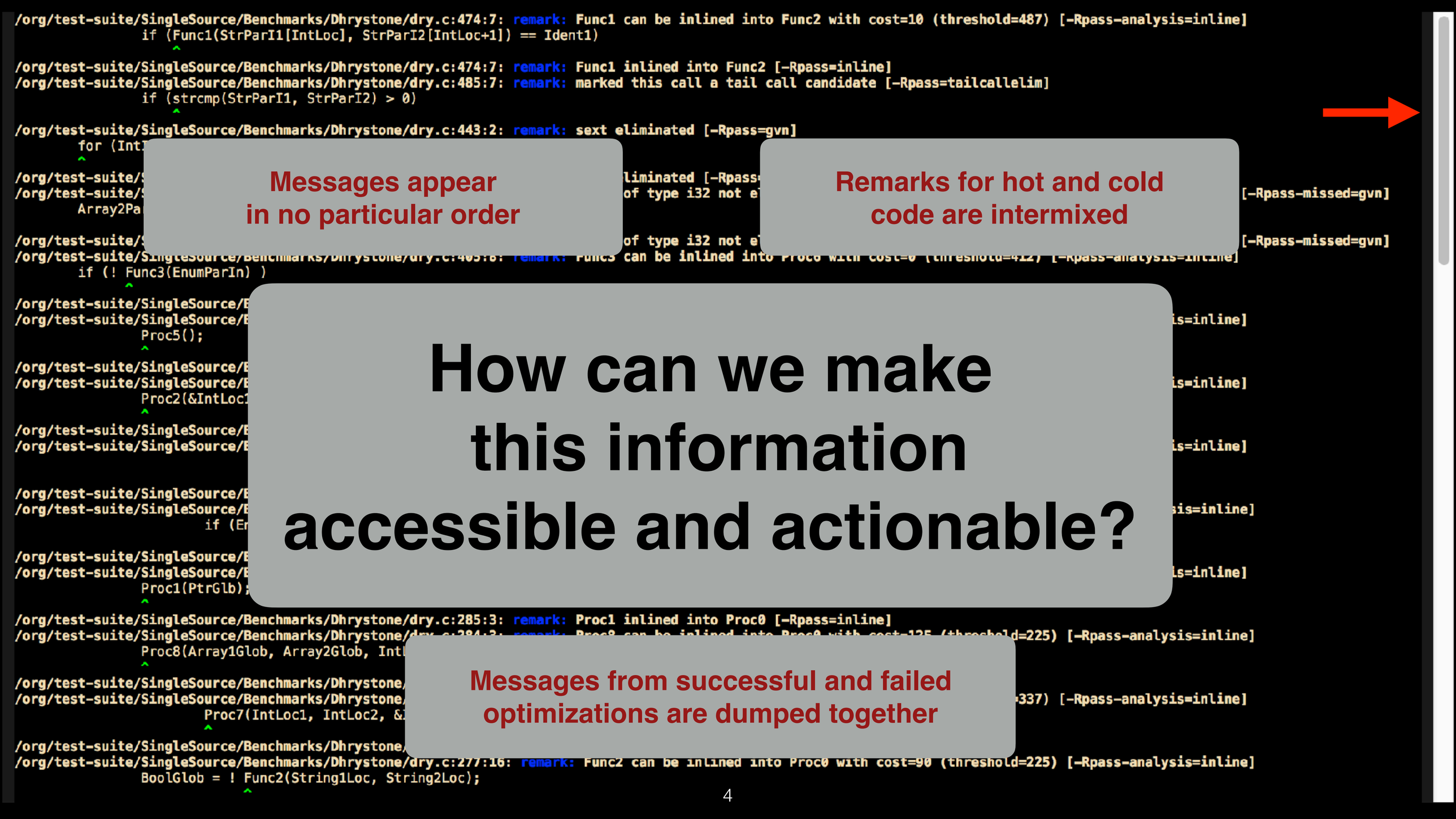
- Supported in LLVM
- Only a small number of passes emit them
- -Rpass options to enable them in the compiler output

```
foo.c:8:5: remark: accumulate inlined into compute_sum[-Rpass=inline]
    accumulate(arr[i], sum);
    ^
```

Optimization Diagnostics in LLVM

- Supported in LLVM
- Only a small number of passes emit them
- -Rpass options to enable them in the compiler output
- For large programs, the output of -Rpass is noisy and unstructured

```
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:474:7: remark: Func1 can be inlined into Func2 with cost=10 (threshold=487) [-Rpass-analysis=inline]
              if (Func1(StrParI1[IntLoc], StrParI2[IntLoc+1]) == Ident1)
                  ^
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:474:7: remark: Func1 inlined into Func2 [-Rpass=inline]
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:485:7: remark: marked this call a tail call candidate [-Rpass=tailcallelim]
              if (strcmp(StrParI1, StrParI2) > 0)
                  ^
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:443:2: remark: sext eliminated [-Rpass=gvn]
       for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
       ^
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:443:2: remark: sext eliminated [-Rpass=gvn]
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:446:33: remark: load of type i32 not eliminated in favor of store because it is clobbered by store [-Rpass-missed=gvn]
       Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
                                      ^
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:446:33: remark: load of type i32 not eliminated in favor of store because it is clobbered by store [-Rpass-missed=gvn]
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:405:8: remark: Func3 can be inlined into Proc6 with cost=0 (threshold=412) [-Rpass-analysis=inline]
       if (! Func3(EnumParIn) )
             ^
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:405:8: remark: Func3 inlined into Proc6 [-Rpass=inline]
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:271:3: remark: Proc5 can be inlined into Proc0 with cost=10 (threshold=337) [-Rpass-analysis=inline]
              Proc5();
              ^
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:271:3: remark: Proc5 inlined into Proc0 [-Rpass=inline]
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:292:3: remark: Proc2 can be inlined into Proc0 with cost=25 (threshold=225) [-Rpass-analysis=inline]
              Proc2(&IntLoc1);
              ^
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:292:3: remark: Proc2 inlined into Proc0 [-Rpass=inline]
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:288:5: remark: Proc6 can be inlined into Proc0 with cost=-5 (threshold=412) [-Rpass-analysis=inline]
                           Proc6(Ident1, &EnumLoc);
                           ^
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:288:5: remark: Proc6 inlined into Proc0 [-Rpass=inline]
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:287:19: remark: Func1 can be inlined into Proc0 with cost=10 (threshold=487) [-Rpass-analysis=inline]
                     if (EnumLoc == Func1(CharIndex, 'C'))
                                    ^
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:287:19: remark: Func1 inlined into Proc0 [-Rpass=inline]
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:285:3: remark: Proc1 can be inlined into Proc0 with cost=15 (threshold=337) [-Rpass-analysis=inline]
              Proc1(PtrGlb);
              ^
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:285:3: remark: Proc1 inlined into Proc0 [-Rpass=inline]
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:284:3: remark: Proc8 can be inlined into Proc0 with cost=125 (threshold=225) [-Rpass-analysis=inline]
              Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3);
              ^
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:284:3: remark: Proc8 inlined into Proc0 [-Rpass=inline]
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:281:4: remark: Proc7 can be inlined into Proc0 with cost=-5 (threshold=337) [-Rpass-analysis=inline]
                     Proc7(IntLoc1, IntLoc2, &IntLoc3);
                     ^
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:281:4: remark: Proc7 inlined into Proc0 [-Rpass=inline]
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:277:16: remark: Func2 can be inlined into Proc0 with cost=90 (threshold=225) [-Rpass-analysis=inline]
              BoolGlob = ! Func2(String1Loc, String2Loc);
                           ^
```


/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:474:7: remark: Func1 can be inlined into Func2 with cost=10 (threshold=487) [-Rpass-analysis=inline]
if (Func1(StrParI1[IntLoc], StrParI2[IntLoc+1]) == Ident1)
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:474:7: remark: Func1 inlined into Func2 [-Rpass=inline]
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:485:7: remark: marked this call a tail call candidate [-Rpass=tailcallelim]
if (strcmp(StrParI1, StrParI2) > 0)
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:443:2: remark: sext eliminated [-Rpass=gvn]
Messages appear in no particular order
Remarks for hot and cold code are intermixed
How can we make this information accessible and actionable?
Proc5();
Proc2(&IntLoc
Proc1(PtrGlb);
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:285:3: remark: Proc1 inlined into Proc0 [-Rpass=inline]
Proc8(Array1Glob, Array2Glob, Int
Proc7(IntLoc1, IntLoc2, &
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:277:16: remark: Func2 can be inlined into Proc0 with cost=90 (threshold=225) [-Rpass-analysis=inline]
BoolGlob = ! Func2(String1Loc, String2Loc);
Messages from successful and failed optimizations are dumped together

Wish List

- **All in one place**: Optimizations Dashboard
- **At a glance**: See high-level interaction between optimizations for targeted low-level debugging
- **Filtering**: Noise-level should be minimized by focusing on the hot code
- **Integration**: Display hot code and the optimizations side-by-side

opt-viewer

Approach

- Extend existing optimization remark infrastructure
  - Add the new optimizations
  - Add ability to output remarks to a data file
- Visualize data in HTML
- Targeting compiler developers initially

Example

Line	Source
1	`void accumulate(int x, int *a) {`
2	`  *a += x;`
3	`}`
4	
5	`int compute_sum(int arr[], int n) {`
6	`  int sum = 0;`
7	`  for (int i = 0; i < n; ++i)`
8	`    accumulate(arr[i], &sum);`
9	`  return sum;`
10	`}`

Work Flow

```
$ clang -O3 -fsave-optimization-record -c foo.c
$ utils/opt-viewer/opt-viewer.py foo.opt.yaml html
$ open html/foo.c.html
```

Successful Optimizations

Line	Optimization	Source
1		`void accumulate(int x, int *a) {`
2		`  *a += x;`
3		`}`
4		
5		`int compute_sum(int arr[], int n) {`
6		`  int sum = 0;`
7		`  for (int i = 0; i < n; ++i)`
	loop-vectorize	vectorized loop (vectorization width: 4, interleaved count: 2)
8		`    accumulate(arr[i], &sum);`
		...line ...late can be inlined into compute_sum with cost=-5 (threshold=487)
		... inlined into compute_sum
9		
10		`}`

Remarks appear inline under the referenced line

Further details about the optimization

Name of the pass

Green for successful optimization

Successful Optimizations

Line	Optimization	Source
1		`void accumulate(int x, int *a) {`
2		`  *a += x;`
3		`}`
4		
5		`int compute_sum(int arr[], int n) {`
6		`  int sum = 0;`
7		`  for (int i = 0; i < n; ++i)`
	loop vectorize	vectorized loop (vectorization width: 4, interleaved count: 2)
8		`    accumulate(arr[i], &sum);`
		accumulate can be inlined into compute_sum with cost=-5 (threshold=487)
	inline	accumulate inlined into compute_sum
9		`  return sum;`
10		`}`

Column aligned with the expression

HTML link to facilitate further analysis

Successful Optimizations

Line	Optimization	Source
1		`void accumulate(int x, int *a) {`
2		`  *a += x;`
3		`}`
4		
5		`int compute_sum(int arr[], int n) {`
6		`  int sum = 0;`
		`  for (int i = 0; i < n; ++i)`
	vectorize	vectorized loop (vectorization width: 4, interleaved count: 2)
8		`    accumulate(arr[i], &sum);`
	inline	accumulate can be inlined into compute_sum with cost=-5 (threshold=487)
	inline	accumulate inlined into compute_sum
9		`  return sum;`
10		`}`

Remarks in white are Analysis remarks

Optimizations can expose interesting analyses

Missed Optimizations

Line	Source
1	`void accumulate(int x, int *a);`
2	
3	`int compute_sum(int arr[], int n) {`
4	`  int sum = 0;`
5	`  for (int i = 0; i < n; ++i)`
6	`    accumulate(arr[i], &sum);`
7	`  return sum;`
8	`}`

Missed Optimizations

Red means failed optimization

Line	Optimization	Source
1		`accumulate(int x, int *a);`
2		
3		`ute_sum(int arr[], int n) {`
4		`int sum = 0;`
5		`for (int i = 0; i < n; ++i)`
	loop-vectorize	loop not vectorized
6		`accumulate(arr[i], &sum);`
	inline	accumulate will not be inlined into compute_sum because its definition is unavailable
	loop-vectorize	loop not vectorized: call instruction cannot be vectorized
7		`return sum;`
8		`}`

LLVM Changes

old

new

Pass pipeline


```
ORE.emit(OptimizationRemarkAnalysis("inline", "CanBeInlined", Call)
         << NV("Callee", Callee) << " can be inlined into " << NV("Caller", Caller)
         << " with cost=" << NV("Cost", IC.getCost())
         << " threshold=" << NV("Threshold", Threshold));
```

OptimizationRemarkEmitter

`-Rpass-analysis=inline`

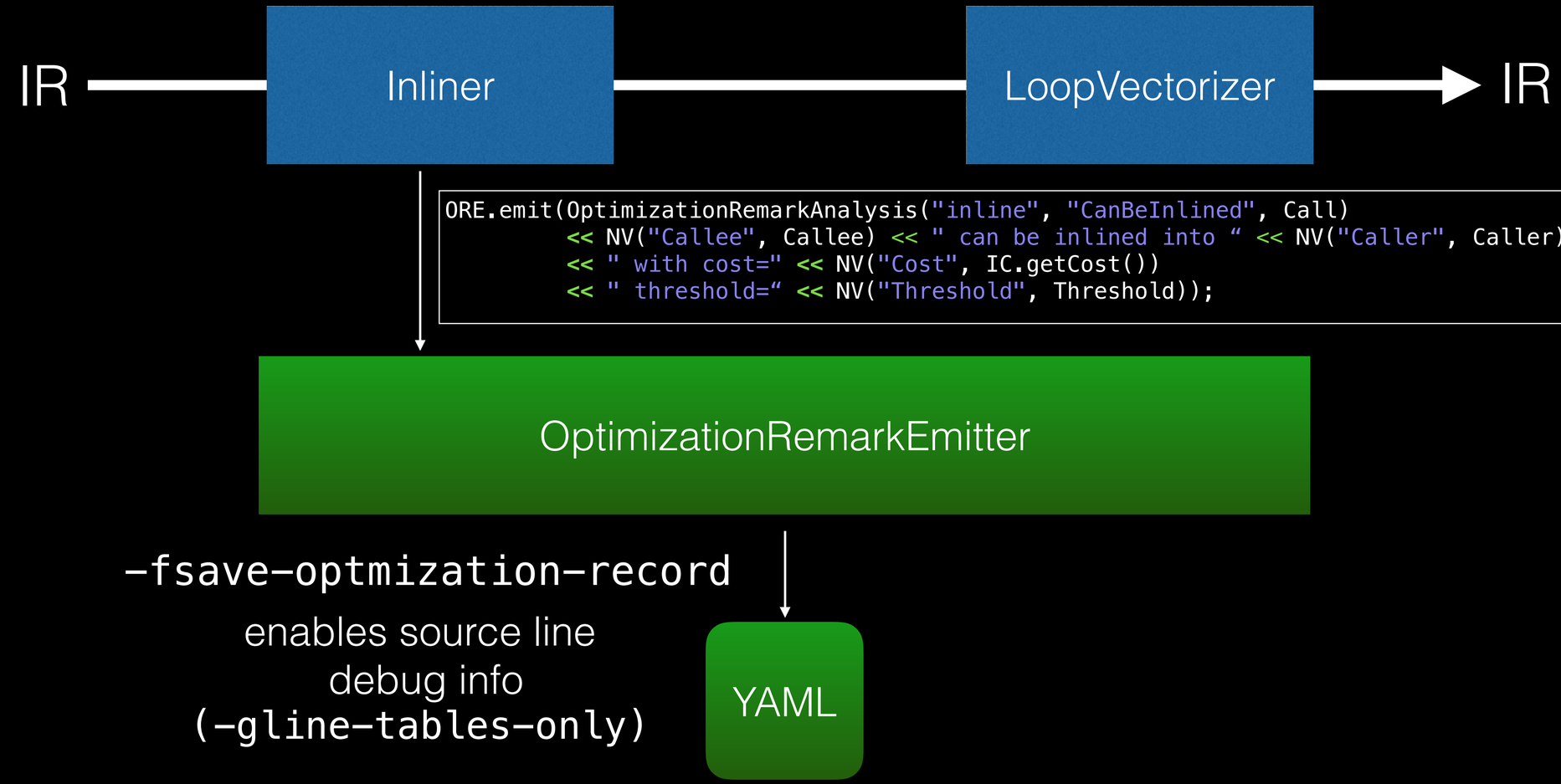
```
foo.c:8:5: remark: accumulate can be inlined into compute_sum with cost=-5 (threshold=487) [-Rpass-analysis=inline]
    accumulate(arr[i], sum);
    ^
```

LLVM Changes

old

new

Pass pipeline



LLVM Changes

old

new

Pass pipeline


```
ORE.emit(OptimizationRemarkAnalysis("inline", "CanBeInlined", Call)
         << NV("Callee", Callee) << " can be inlined into " << NV("Caller", Caller)
         << " with cost=" << NV("Cost", IC.getCost())
         << " thres
```

Optimiza

```
--- !Analysis
Pass:            inline
Name:            CanBeInlined
DebugLoc:        { File: s.cc, Line: 8, Column: 5 }
Function:        compute_sum
Args:
  - Callee:          accumulate
    DebugLoc:        { File: s.cc, Line: 1, Column: 0 }
  - String:          ' can be inlined into '
  - Caller:          compute_sum
    DebugLoc:        { File: s.cc, Line: 5, Column: 0 }
  - String:          ' with cost='
  - Cost:            '-5'
  - String:          ' (threshold='
  - Threshold:       '487'
  - String:          ')'
...
```

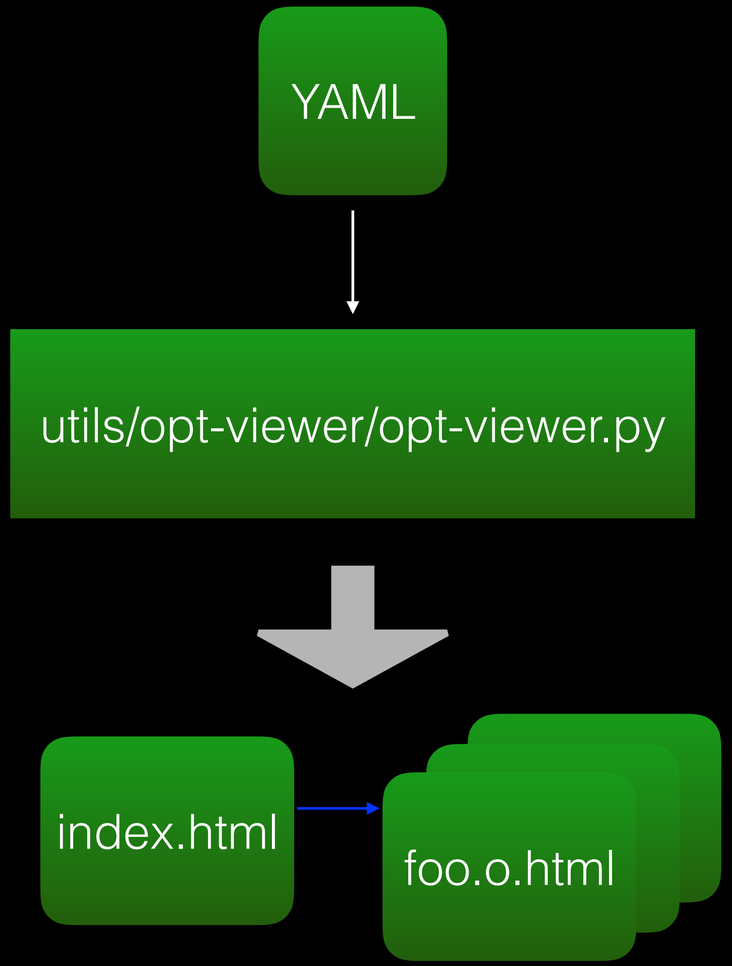
`-fsave-optmization-recor`

enables source line
debug info
(`-gline-tables-only`)

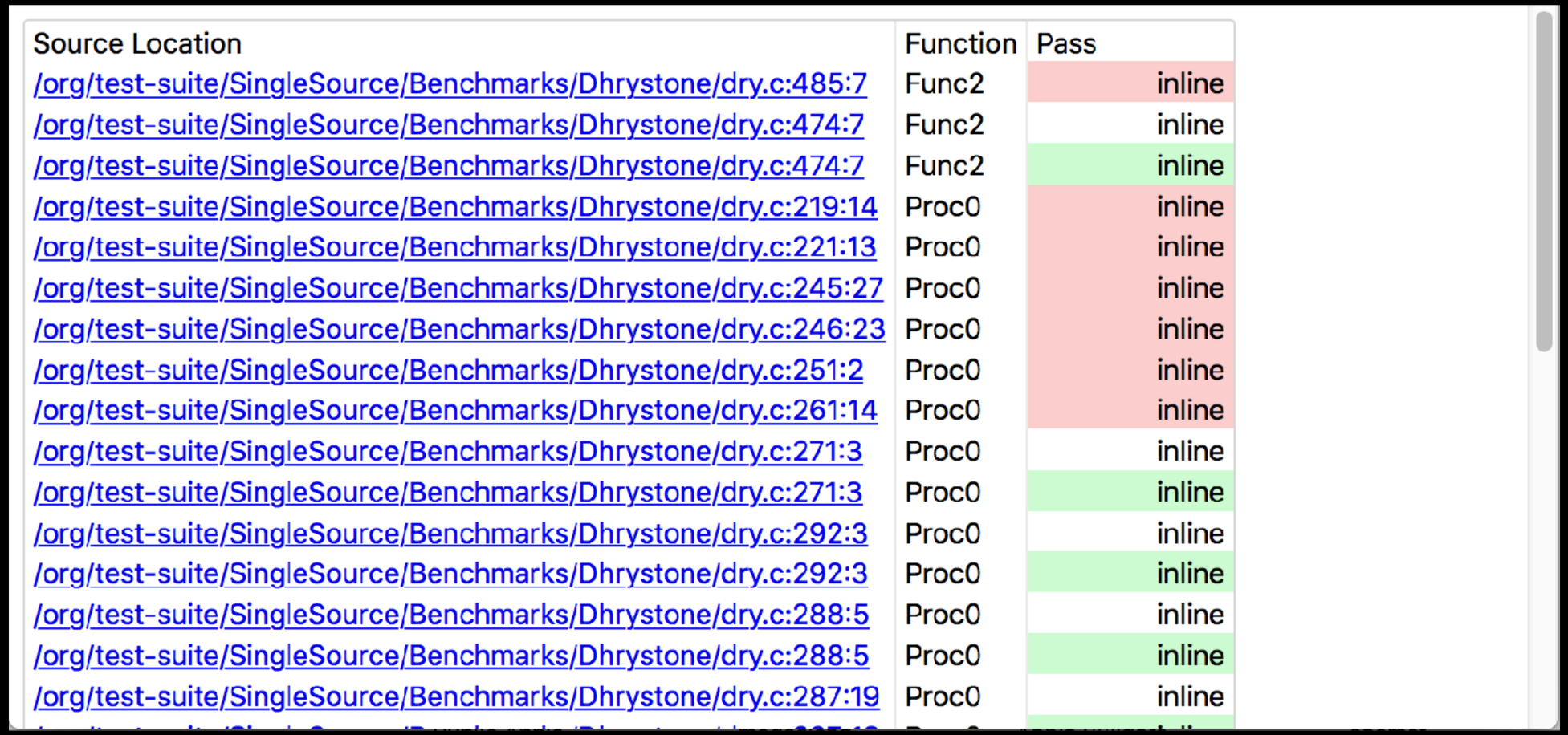
opt-viewer

old

new

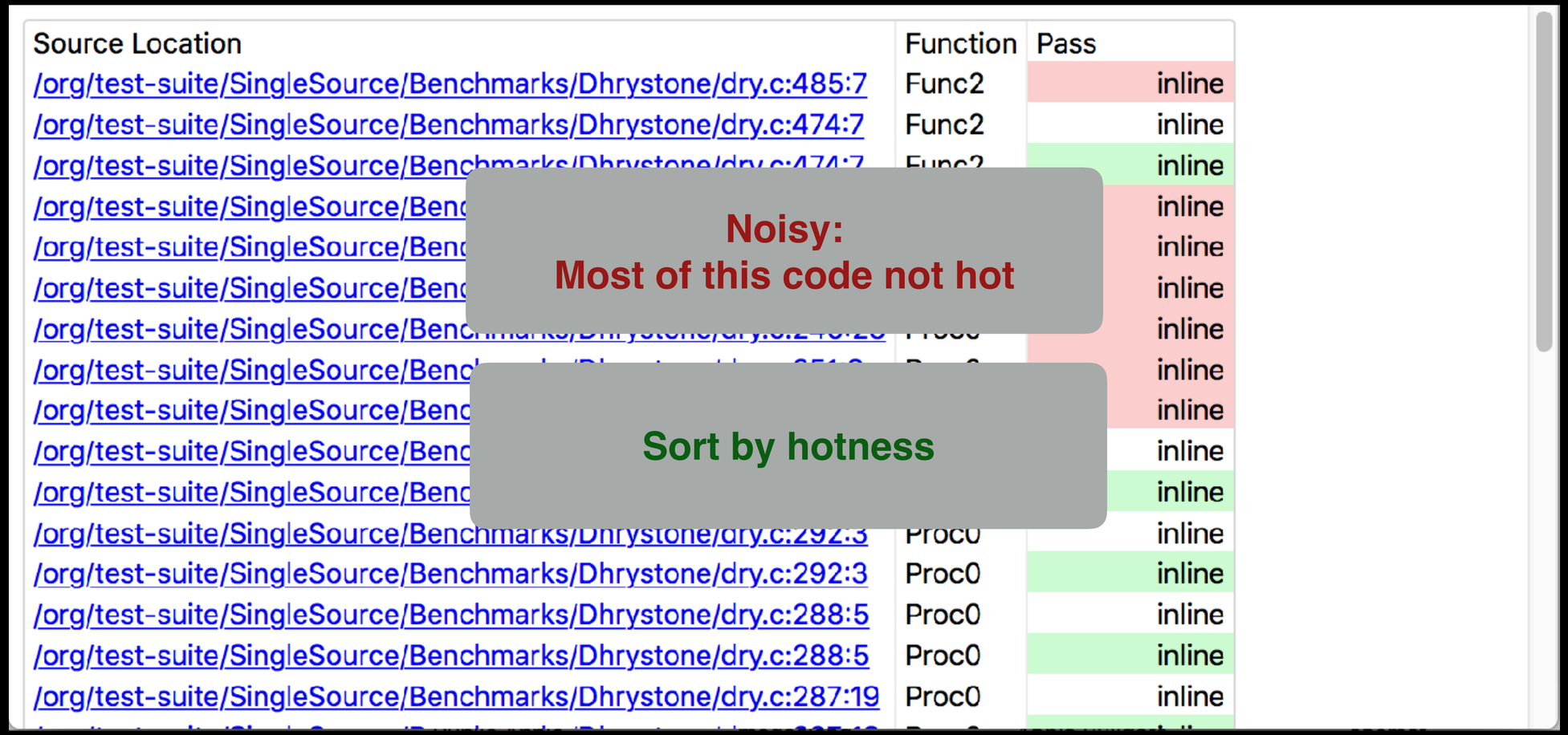


Index



Source Location	Function	Pass
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:485:7	Func2	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:474:7	Func2	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:474:7	Func2	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:219:14	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:221:13	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:245:27	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:246:23	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:251:2	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:261:14	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:271:3	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:271:3	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:292:3	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:292:3	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:288:5	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:288:5	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:287:19	Proc0	inline

Index

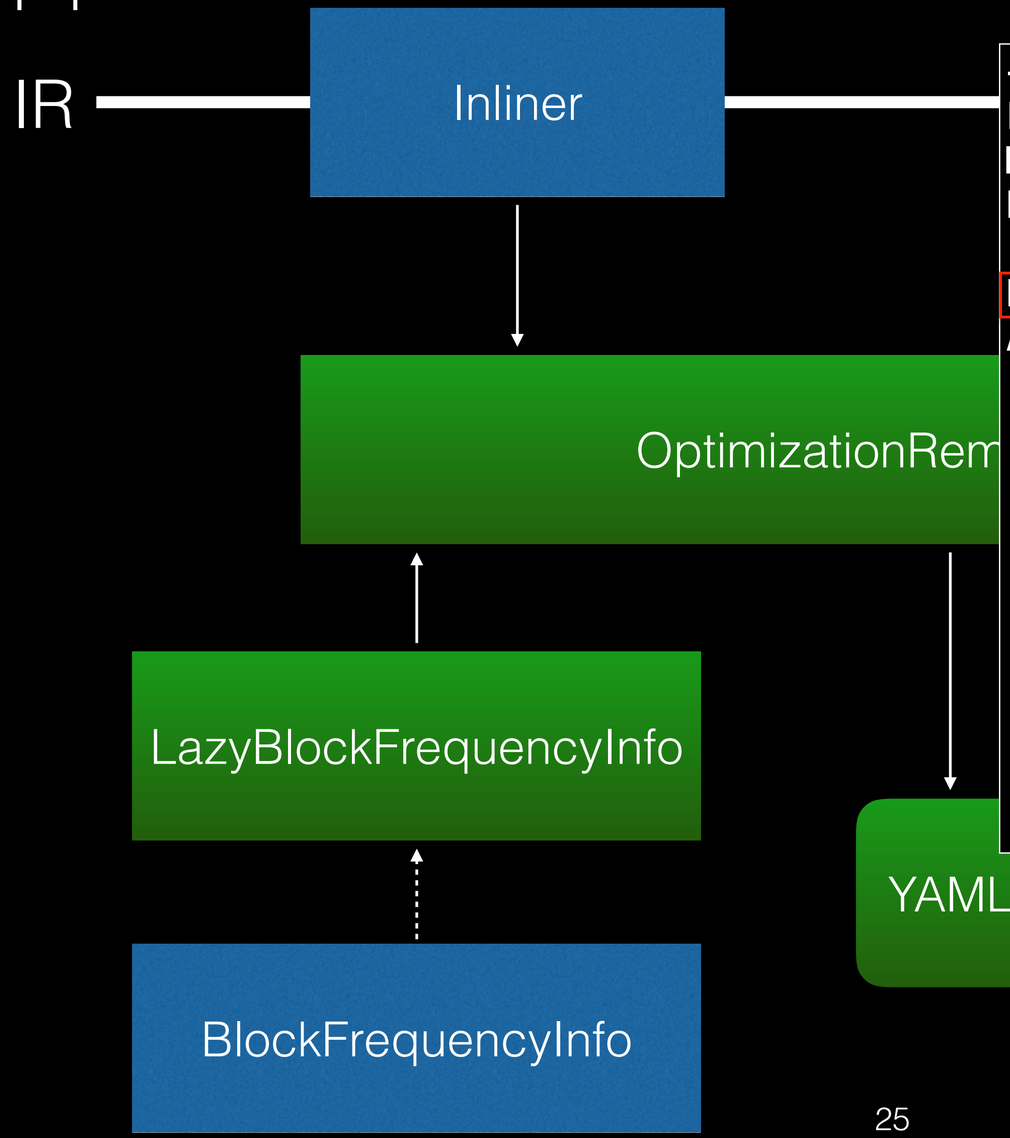

Source Location
Function
Pass
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:485:7
Func2
inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:474:7
Func2
inline
inline
inline
inline
inline
inline
inline
inline
inline
inline
inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:292:3
Proc0
inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:292:3
Proc0
inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:288:5
Proc0
inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:288:5
Proc0
inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:287:19
Proc0
inline
Noisy:
Most of this code not hot
Sort by hotness

Use PGO for Hotness

old

new

Pass pipeline




```
--- !Analysis
Pass:            inline
Name:            CanBeInlined
DebugLoc:        { File: s.cc, Line: 8, Column: 5 }
Function:        compute_sum
Hotness:         3
Args:
  - Callee:          accumulate
    DebugLoc:        { File: s.cc, Line: 1, Column: 0 }
  - String:          ' can be inlined into '
  - Caller:          compute_sum
    DebugLoc:        { File: s.cc, Line: 5, Column: 0 }
  - String:          ' with cost='
  - Cost:            '-5'
  - String:          ' (threshold='
  - Threshold:       '487'
  - String:          ')'
...
```

Hotness

Source Location	Hotness	Function	Pass
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:443:2	100%	Proc8	loop-vectorize
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:473:2	65%	Func2	loop-vectorize
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:473:2	65%	Func2	loop-vectorize
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:443:2	52%	Proc0	loop-delete
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:444:31	52%	Proc0	loop-idiom
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:287:19	36%	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:287:19	36%	Proc0	inline
/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:474:7	34%	Func2	inline

```
437                 REG OneToFifty  IntIndex;
438
439                 IntLoc = IntParI1 + 5;
440                 Array1Par[IntLoc] = IntParI2;
441                 Array1Par[IntLoc+1] = Array1Par[IntLoc];
442                 Array1Par[IntLoc+30] = IntLoc;
443                 for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
    52%   loop-delete   loop deleted
    100%  loop-vectorize   vectorized loop (vectorization width: 4, interleaved count: 2)
444                     Array2Par[IntLoc][IntIndex] = IntLoc;
    52%   loop-idiom   formed memset
445                 ++Array2Par[IntLoc][IntLoc-1];
446                 Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
    17%   gvn   load of type i32 eliminated
447                 IntGlob = 5;
448         }
```

Display a menu

Relative to maximum hotness, NOT total time %

Optimizations Recorded

Function Inliner

Loop Vectorizer

Loop Unroller

LoopDataPrefetch

LICM

GVN

Loop Idiom

Loop Deletion

SLP Vectorizer

… more to follow

Test Drive
on
LLVM test suite

Improve & Evaluate

1. Does the information presented in this high-level view contain sufficient detail to reconstruct what happened?

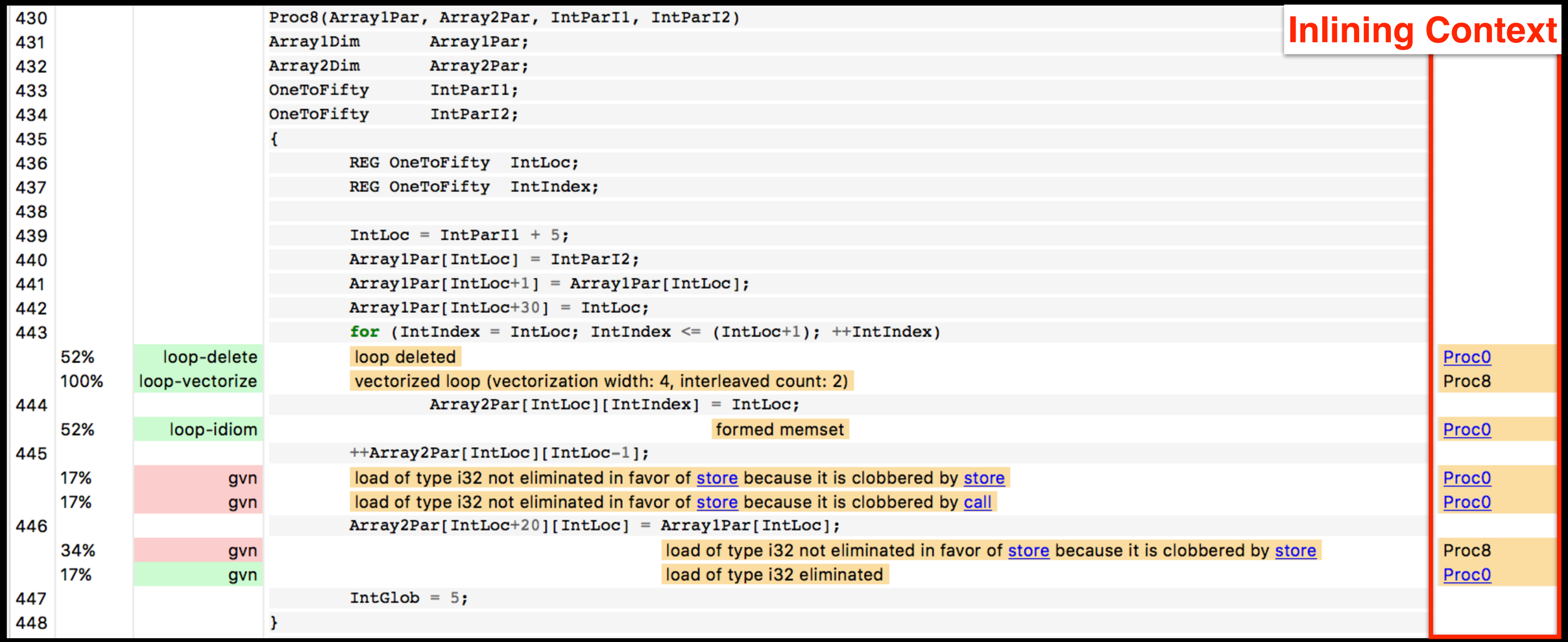
2. Can we discover the interactions between optimizations?

3. With the improved visibility, can we quickly find real performance opportunities?

DhryStone
(SingleSource/Benchmark)

Interaction of Optimizations

DhryStone


Inlining Context
Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
Array1Dim Array1Par;
Array2Dim Array2Par;
OneToFifty IntParI1;
OneToFifty IntParI2;
{
REG OneToFifty IntLoc;
REG OneToFifty IntIndex;
IntLoc = IntParI1 + 5;
Array1Par[IntLoc] = IntParI2;
Array1Par[IntLoc+1] = Array1Par[IntLoc];
Array1Par[IntLoc+30] = IntLoc;
for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
52% loop-delete
loop deleted
Proc0
100% loop-vectorize
vectorized loop (vectorization width: 4, interleaved count: 2)
Proc8
Array2Par[IntLoc][IntIndex] = IntLoc;
52% loop-idiom
formed memset
Proc0
++Array2Par[IntLoc][IntLoc-1];
17% gvn
load of type i32 not eliminated in favor of store because it is clobbered by store
Proc0
17% gvn
load of type i32 not eliminated in favor of store because it is clobbered by call
Proc0
Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
34% gvn
load of type i32 not eliminated in favor of store because it is clobbered by store
Proc8
17% gvn
load of type i32 eliminated
Proc0
IntGlob = 5;
}

DhryStone

```
430                     Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
431                     Array1Dim       Array1Par;
432                     Array2Dim       Array2Par;
433                     OneToFifty      IntParI1;
434                     OneToFifty      IntParI2;
435                     {
436                             REG OneToFifty  IntLoc;
437                             REG OneToFifty  IntIndex;
438
439                             IntLoc = IntParI1 + 5;
440                             Array1Par[IntLoc] = IntParI2;
441                             Array1Par[IntLoc+1] = Array1Par[IntLoc];
442                             Array1Par[IntLoc+30] = IntLoc;
443                             for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)

    100%  loop-vectorize      vectorized loop (vectorization width: 4, interleaved count: 2)   Proc8
444                                     Array2Par[IntLoc][IntIndex] = IntLoc;

445                             ++Array2Par[IntLoc][IntLoc-1];


446                             Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
    34%           gvn                                      load of type i32 not eliminated in favor of store because it is clobbered by store   Proc8

447                             IntGlob = 5;
448                     }
```

DhryStone

```
430                     Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
431                     Array1Dim       Array1Par;
432                     Array2Dim       Array2Par;
433                     OneToFifty      IntParI1;
434                     OneToFifty      IntParI2;
435                     {
436                             REG OneToFifty  IntLoc;
437                             REG OneToFifty  IntIndex;
438
439                             IntLoc = IntParI1 + 5;
440                             Array1Par[IntLoc] = IntParI2;
441                             Array1Par[IntLoc+1] = Array1Par[IntLoc];
442                             Array1Par[IntLoc+30] = IntLoc;
443                             for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)

    100%  loop-vectorize      vectorized loop (vectorization width: 4, interleaved count: 2)   Proc8
444                                     Array2Par[IntLoc][IntIndex] = IntLoc;

445                             ++Array2Par[IntLoc][IntLoc-1];


446                             Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
     34%              gvn       load of type i32 not eliminated in favor of store because it is clobbered by store   Proc8

447                             IntGlob = 5;
448                     }
```

DhryStone

```
430                     Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
431                     Array1Dim       Array1Par;
432                     Array2Dim       Array2Par;
433                     OneToFifty      IntParI1;
434                     OneToFifty      IntParI2;
435                     {
436                             REG OneToFifty  IntLoc;
437                             REG OneToFifty  IntIndex;
438
439                             IntLoc = IntParI1 + 5;
440                             Array1Par[IntLoc] = IntParI2;
441                             Array1Par[IntLoc+1] = Array1Par[IntLoc];
442                             Array1Par[IntLoc+30] = IntLoc;
443                             for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)

    100%  loop-vectorize      vectorized loop (vectorization width: 4, interleaved count: 2)   Proc8
444                                     Array2Par[IntLoc][IntIndex] = IntLoc;

445                             ++Array2Par[IntLoc][IntLoc-1];

446                             Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
    34%           gvn         load of type i32 not eliminated in favor of store because it is clobbered by store   Proc8

447                             IntGlob = 5;
448                     }
```

DhryStone

```
430                 Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
431                 Array1Dim       Array1Par;
432                 Array2Dim       Array2Par;
433                 OneToFifty      IntParI1;
434                 OneToFifty      IntParI2;
435                 {
436                         REG OneToFifty  IntLoc;
437                         REG OneToFifty  IntIndex;
438
439                         IntLoc = IntParI1 + 5;
440                         Array1Par[IntLoc] = IntParI2;
441                         Array1Par[IntLoc+1] = Array1Par[IntLoc];
442                         Array1Par[IntLoc+30] = IntLoc;
443                         for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
    52%  loop-delete             loop deleted                                          Proc0

444                                 Array2Par[IntLoc][IntIndex] = IntLoc;
    52%  loop-idiom                                 formed memset                     Proc0
445                         ++Array2Par[IntLoc][IntLoc-1];
    17%  gvn                    load of type i32 not eliminated in favor of store because it is clobbered by store   Proc0
    17%  gvn                    load of type i32 not eliminated in favor of store because it is clobbered by call    Proc0
446                         Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];

    17%  gvn                                    load of type i32 eliminated       Proc0
447                         IntGlob = 5;
448                 }
```

DhryStone

```
430                 Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
431                 Array1Dim       Array1Par;
432                 Array2Dim       Array2Par;
433                 OneToFifty      IntParI1;
434                 OneToFifty      IntParI2;
435                 {
436                         REG OneToFifty  IntLoc;
437                         REG OneToFifty  IntIndex;
438
439                         IntLoc = IntParI1 + 5;
440                         Array1Par[IntLoc] = IntParI2;
441                         Array1Par[IntLoc+1] = Array1Par[IntLoc];
442                         Array1Par[IntLoc+30] = IntLoc;
443                         for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
    52%  loop-delete             loop deleted                                          Proc0

444                                 Array2Par[IntLoc][IntIndex] = IntLoc;
    52%  loop-idiom                                 formed memset                     Proc0
445                         ++Array2Par[IntLoc][IntLoc-1];
    17%  gvn                    load of type i32 not eliminated in favor of store because it is clobbered by store   Proc0
    17%  gvn                    load of type i32 not eliminated in favor of store because it is clobbered by call    Proc0
446                         Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];

    17%  gvn                                    load of type i32 eliminated       Proc0
447                         IntGlob = 5;
448                 }
```

DhryStone

430			`Proc8(Array1Par, Array2Par, IntParI1, IntParI2)`	
431			`Array1Dim Array1Par;`	
432			`Array2Dim Array2Par;`	
433			`OneToFifty IntParI1;`	
434			`OneToFifty IntParI2;`	
435			`{`	
436			`REG OneToFifty IntLoc;`	
437			`REG OneToFifty IntIndex;`	
438				
439			`IntLoc = IntParI1 + 5;`	
440			`Array1Par[IntLoc] = IntParI2;`	
441			`Array1Par[IntLoc+1] = Array1Par[IntLoc];`	
442			`Array1Par[IntLoc+30] = IntLoc;`	
443			`for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)`	
	52%	loop-delete	loop deleted	Proc0
444			`Array2Par[IntLoc][IntIndex] = IntLoc;`	
	52%	loop-idiom	formed memset	Proc0
445			`++Array2Par[IntLoc][IntLoc-1];`	
	17%	gvn	load of type i32 not eliminated in favor of store because it is clobbered by store	Proc0
	17%	gvn	load of type i32 not eliminated in favor of store because it is clobbered by call	Proc0
446			`Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];`	
	17%	gvn	load of type i32 eliminated	Proc0
447			`IntGlob = 5;`	
448			`}`	

DhryStone

278			`while (IntLoc1 < IntLoc2)`	
	34%	loop-unroll	completely unrolled loop with 2 iterations	Proc0
279			`{`	
280			`IntLoc3 = 5 * IntLoc1 - IntLoc2;`	
281			`Proc7(IntLoc1, IntLoc2, &IntLoc3);`	
	18%	inline	Proc7 can be inlined into Proc0 with cost=-5 (threshold=337)	Proc0
	18%	inline	Proc7 inlined into Proc0	Proc0
282			`++IntLoc1;`	
283			`}`	
284			`Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3);`	
	18%	inline	Proc8 can be inlined into Proc0 with cost=125 (threshold=225)	Proc0
	18%	inline	Proc8 inlined into Proc0	Proc0
285			`Proc1(PtrGlb);`	
	18%	inline	Proc1 can be inlined into Proc0 with cost=15 (threshold=337)	Proc0
	18%	inline	Proc1 inlined into Proc0	Proc0
286			`for (CharIndex = 'A'; CharIndex <= Char2Glob; ++CharIndex)`	
	34%	licm	load hoisted	Proc0
	17%	gvn	load of type i8 not eliminated in favor of store because it is clobbered by store	Proc0
	11%	gvn	load of type i8 eliminated	Proc0

DhryStone

```
430                 Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
431                 Array1Dim       Array1Par;
432                 Array2Dim       Array2Par;
433                 OneToFifty      IntParI1;
434                 OneToFifty      IntParI2;
435                 {
436                         REG OneToFifty  IntLoc;
437                         REG OneToFifty  IntIndex;
438
439                         IntLoc = IntParI1 + 5;
440                         Array1Par[IntLoc] = IntParI2;
441                         Array1Par[IntLoc+1] = Array1Par[IntLoc];
442                         Array1Par[IntLoc+30] = IntLoc;
443                         for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
    52%  loop-delete             loop deleted                                          Proc0

444                                 Array2Par[IntLoc][IntIndex] = IntLoc;
    52%  loop-idiom                                 formed memset                     Proc0
445                         ++Array2Par[IntLoc][IntLoc-1];
    17%  gvn                    load of type i32 not eliminated in favor of store because it is clobbered by store   Proc0
    17%  gvn                    load of type i32 not eliminated in favor of store because it is clobbered by call    Proc0
446                         Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];

    17%  gvn                                    load of type i32 eliminated       Proc0
447                         IntGlob = 5;
448                 }
```

DhryStone: Summary

- Without low-level debugging, quickly reconstructed what happened
- Even though it involved interaction between multiple optimizations
  - Inlining and Alias Analysis/GVN
- Missed optimizations: Extra analysis to manage with false positives
  1. Filter trivially false positives
  2. Expose enough information for quick detection by user

Freebench/distray
(MultiSource/Benchmarks)

Finding Performance Opportunity

```
211                 static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
        99%  loop-vectorize   loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                    /* Translate object into "line-space" */
        100%         licm     failed to hoist load with loop-invariant address because the loop may invalidate its value
240                         Pos.y -= LinP->y;
        100%         licm     failed to hoist load with loop-invariant address because the loop may invalidate its value
241                         Pos.z -= LinP->z;
        100%         licm     failed to hoist load with loop-invariant address because the loop may invalidate its value
        20%           gvn     getelementptr eliminated by PRE
242                         A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
        100%         licm     failed to hoist load with loop-invariant address because the loop may invalidate its value
        100%         licm     failed to hoist load with loop-invariant address because the loop may invalidate its value
        100%         licm     failed to hoist load with loop-invariant address because the loop may invalidate its value
        100%          gvn     load of type double not eliminated in favor of load because it is clobbered by store
        100%          gvn     load of type double not eliminated in favor of load because it is clobbered by store
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
245                         if( (A = C + B*B) > 0.0 ) {          /* ...else no hit */
246                             A = sqrt(A);
        9%          inline      sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;          /* Calculate intersection point */
        5%    slp-vectorizer        Stores SLP vectorized
251                                 Pnt->y = LinD->y*t;
        5%           licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
        5%            gvn           load of type double eliminated in favor of load
252                                 Pnt->z = LinD->z*t;
        5%           licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
        5%            gvn           load of type double eliminated in favor of load
        100%          gvn           load of type double not eliminated in favor of load because it is clobbered by store
        100%          gvn           load of type double not eliminated in favor of load because it is clobbered by store
253                                 Norm->x = Pnt->x-Pos.x;      /* Calcualate surface normal */
        5%           licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
        5%            gvn           load of type double eliminated in favor of fmul
254                                 Norm->y = Pnt->y-Pos.y;
```

```
211                 static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
    99%  loop-vectorize  loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                    /* Translate object into "line-space" */
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
240                         Pos.y -= LinP->y;
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
241                         Pos.z -= LinP->z;
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    20%   gvn            getelementptr eliminated by PRE
242                         A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  gvn            load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn            load of type double not eliminated in favor of load because it is clobbered by store
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
245                         if( (A = C + B*B) > 0.0 ) {          /* ...else no hit */
246                             A = sqrt(A);
    9%    inline         sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;          /* Calculate intersection point */
    5%    slp-vectorizer     Stores SLP vectorized
251                                 Pnt->y = LinD->y*t;
    5%    licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%    gvn            load of type double eliminated in favor of load
252                                 Pnt->z = LinD->z*t;
    5%    licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%    gvn            load of type double eliminated in favor of load
    100%  gvn            load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn            load of type double not eliminated in favor of load because it is clobbered by store
253                                 Norm->x = Pnt->x-Pos.x;      /* Calcualate surface normal */
    5%    licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%    gvn            load of type double eliminated in favor of fmul
254                                 Norm->y = Pnt->y-Pos.y;
```

```
211                 static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
    99%  loop-vectorize  loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                   /* Translate object into "line-space" */
    100%  licm  failed to hoist load with loop-invariant address because the loop may invalidate its value
240                         Pos.y -= LinP->y;
    100%  licm  failed to hoist load with loop-invariant address because the loop may invalidate its value
241                         Pos.z -= LinP->z;
    100%  licm  failed to hoist load with loop-invariant address because the loop may invalidate its value
    20%   gvn   getelementptr eliminated by PRE
242                         A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
    100%  licm  failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm  failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm  failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  gvn   load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn   load of type double not eliminated in favor of load because it is clobbered by store
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
245                         if( (A = C + B*B) > 0.0 ) {         /* ...else no hit */
246                             A = sqrt(A);
    9%   inline  sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;         /* Calculate intersection point */
    5%   slp-vectorizer  Stores SLP vectorized
251                                 Pnt->y = LinD->y*t;
    5%   licm  failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn   load of type double eliminated in favor of load
252                                 Pnt->z = LinD->z*t;
    5%   licm  failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn   load of type double eliminated in favor of load
    100%  gvn  load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn  load of type double not eliminated in favor of load because it is clobbered by store
253                                 Norm->x = Pnt->x-Pos.x;     /* Calcualate surface normal */
    5%   licm  failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn   load of type double eliminated in favor of fmul
254                                 Norm->y = Pnt->y-Pos.y;
```

```
211                 static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
    99%  loop-vectorize     loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                   /* Translate object into "line-space" */
    100%  licm                           failed to hoist load with loop-invariant address because the loop may invalidate its value
240                         Pos.y -= LinP->y;
    100%  licm                           failed to hoist load                                                    value
241                         Pos.z -= LinP->z;
    100%  licm                           failed to hoist load                                                    value
    20%   gvn                            getelementptr elimi
242                         A = 1.0 / (LinD->x*LinD->x + Li
    100%  licm                               failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm                                   failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm                                       failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  gvn                                        load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn                                        load of type double not eliminated in favor of load because it is clobbered by store
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
245                         if( (A = C + B*B) > 0.0 ) {         /* ...else no hit */
246                             A = sqrt(A);
    9%    inline                sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;         /* Calculate intersection point */
    5%    slp-vectorizer            Stores SLP vectorized
251                                 Pnt->y = LinD->y*t;
    5%    licm                                  failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%    gvn                                   load of type double eliminated in favor of load
252                                 Pnt->z = LinD->z*t;
    5%    licm                                  failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%    gvn                                   load of type double eliminated in favor of load
    100%  gvn                                   load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn                                   load of type double not eliminated in favor of load because it is clobbered by store
253                                 Norm->x = Pnt->x-Pos.x;     /* Calcualate surface normal */
    5%    licm                                  failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%    gvn                                   load of type double eliminated in favor of fmul
254                                 Norm->y = Pnt->y-Pos.y;
```

Not modified via LinP, maybe writes through other pointers

```
211                 static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
    99%  loop-vectorize    loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                   /* Translate object into "line-space" */
    100%  licm                 failed to hoist load with loop-invariant address because the loop may invalidate its value
240                         Pos.y -= LinP->y;
    100%  licm                 failed to hoist load with loop-invariant address because the loop may invalidate its value
241                         Pos.z -= LinP->z;
    100%  licm                 failed to hoist load with loop-invariant address because the loop may invalidate its value
    20%   gvn                  getelementptr eliminated by PRE
242                         A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
    100%  licm                 failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm                     failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm                         faile                                                    y invalidate its value
    100%  gvn                          load                                                     bered by store
    100%  gvn                          load                                                     bered by store
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos
245                         if( (A = C + B*B) > 0.0 ) {         /* ...else no hit */
246                             A = sqrt(A);
    9%   inline                sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;         /* Calculate intersection point */
    5%   slp-vectorizer            Stores SLP vectorized
251                                 Pnt->y = LinD->y*t;
    5%   licm                         failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn                          load of type double eliminated in favor of load
252                                 Pnt->z = LinD->z*t;
    5%   licm                         failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn                          load of type double eliminated in favor of load
    100%  gvn                        load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn                        load of type double not eliminated in favor of load because it is clobbered by store
253                                 Norm->x = Pnt->x-Pos.x;     /* Calcualate surface normal */
    5%   licm                         failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn                          load of type double eliminated in favor of fmul
254                                 Norm->y = Pnt->y-Pos.y;
```

Not modified via LinD, maybe writes through other pointers

```
211                 static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
    99%  loop-vectorize  loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                   /* Translate object into "line-space" */
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
240                         Pos.y -= LinP->y;
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
241                         Pos.z -= LinP->z;
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    20%   gvn            getelementptr eliminated by PRE
242                         A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  gvn            load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn            load of type double not eliminated in favor of load because it is clobbered by store
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
245                         if( (A = C + B*B) > 0.0 ) {         /* ...else no hit */
246                             A = sqrt(A);
    9%   inline          sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;         /* Calculate intersection point */
    5%   slp-vectorizer  Stores SLP vectorized
251                                 Pnt->y = LinD->y*t;
    5%   licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn            load of type double eliminated in favor of load
252                                 Pnt->z = LinD->z*t;
    5%   licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn            load of type double eliminated in favor of load
    100% gvn            load of type double not eliminated in favor of load because it is clobbered by store
    100% gvn            load of type double not eliminated in favor of load because it is clobbered by store
253                                 Norm->x = Pnt->x-Pos.x;     /* Calcualate surface normal */
    5%   licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn            load of type double eliminated in favor of fmul
254                                 Norm->y = Pnt->y-Pos.y;
```

```
211                 static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
```

Reads and writes don't alias

```
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
        99%  loop-vectorize   loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                    /* Translate object into "line-space" */
        100%  licm            failed to hoist load with loop-invariant address because the loop may invalidate its value
240                         Pos.y -= LinP->y;
        100%  licm            failed to hoist load with loop-invariant address because the loop may invalidate its value
241                         Pos.z -= LinP->z;
        100%  licm            failed to hoist load with loop-invariant address because the loop may invalidate its value
        20%   gvn             getelementptr eliminated by PRE
242                         A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
        100%  licm            failed to hoist load with loop-invariant address because the loop may invalidate its value
        100%  licm            failed to hoist load with loop-invariant address because the loop may invalidate its value
        100%  licm            failed to hoist load with loop-invariant address because the loop may invalidate its value
        100%  gvn             load of type double not eliminated in favor of load because it is clobbered by store
        100%  gvn             load of type double not eliminated in favor of load because it is clobbered by store
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
245                         if( (A = C + B*B) > 0.0 ) {          /* ...else no hit */
246                             A = sqrt(A);
        9%    inline            sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;          /* Calculate intersection point */
        5%    slp-vectorizer        Stores SLP vectorized
251                                 Pnt->y = LinD->y*t;
        5%    licm                  failed to hoist load with loop-invariant address because the loop may invalidate its value
        5%    gvn                   load of type double eliminated in favor of load
252                                 Pnt->z = LinD->z*t;
        5%    licm                  failed to hoist load with loop-invariant address because the loop may invalidate its value
        5%    gvn                   load of type double eliminated in favor of load
        100%  gvn                   load of type double not eliminated in favor of load because it is clobbered by store
        100%  gvn                   load of type double not eliminated in favor of load because it is clobbered by store
253                                 Norm->x = Pnt->x-Pos.x;      /* Calcualate surface normal */
        5%    licm                  failed to hoist load with loop-invariant address because the loop may invalidate its value
        5%    gvn                   load of type double eliminated in favor of fmul
254                                 Norm->y = Pnt->y-Pos.y;
```

Loop versioning
with array overlap checks?

```
211                 static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
    99%  loop-vectorize  loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                    /* Translate object into "line-space" */
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
240                         Pos.y -= LinP->y;
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
241                         Pos.z -= LinP->z;
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    20%   gvn            getelementptr eliminated by PRE
242                         A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  gvn            load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn            load of type double not eliminated in favor of load because it is clobbered by store
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
245                         if( (A = C + B*B) > 0.0 ) {          /* ...else no hit */
246                             A = sqrt(A);
    9%    inline         sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;          /* Calculate intersection point */
    5%    slp-vectorizer       Stores SLP vectorized
251                                 Pnt->y = LinD->y*t;
    5%    licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%    gvn            load of type double eliminated in favor of load
252                                 Pnt->z = LinD->z*t;
    5%    licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%    gvn            load of type double eliminated in favor of load
    100%  gvn            load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn            load of type double not eliminated in favor of load because it is clobbered by store
253                                 Norm->x = Pnt->x-Pos.x;      /* Calcualate surface normal */
    5%    licm           failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%    gvn            load of type double eliminated in favor of fmul
254                                 Norm->y = Pnt->y-Pos.y;
```

LICM-based LoopVersioning (`-enable-loop-versioning-licm`)

```
211                 static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
    99%  version-licm       Loop memory access not suitable
    99%  loop-vectorize     loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                   /* Translate object into "line-space" */
    100%  licm                          failed to hoist load with loop-invariant address because the loop may invalidate its value
240                         Pos.y -= LinP->y;
    100%  licm                          failed to hoist load with loop-invariant address because the loop may invalidate its value
241                         Pos.z -= LinP->z;
    100%  licm                          failed to hoist load with loop-invariant address because the loop may invalidate its value
    20%   gvn                           getelementptr eliminated by PRE
242                         A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
    100%  licm                            failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm                                          failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm                                                                failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  gvn                                                                 load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn                                                                 load of type double not eliminated in favor of load because it is clobbered by store
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
245                         if( (A = C + B*B) > 0.0 ) {         /* ...else no hit */
246                             A = sqrt(A);
    9%   inline                  sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;         /* Calculate intersection point */
    5%   slp-vectorizer                 Stores SLP vectorized
251                                 Pnt->y = LinD->y*t;
    5%   licm                                  failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn                                   load of type double eliminated in favor of load
252                                 Pnt->z = LinD->z*t;
    5%   licm                                  failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn                                   load of type double eliminated in favor of load
    100%  gvn                                  load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn                                  load of type double not eliminated in favor of load because it is clobbered by store
253                                 Norm->x = Pnt->x-Pos.x;     /* Calcualate surface normal */
    5%   licm                                  failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn                                   load of type double eliminated in favor of fmul
```

Performance opportunity if we can improve this pass

```
211                 static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
    99%  version-licm       Loop memory access not suitable
    99%  loop-vectorize     loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                   /* Translate object into "line-space" */
    100%  licm                  failed to hoist load with loop-invariant address because the loop may invalidate its value
240                         Pos.y -= LinP->y;
    100%  licm                  failed to hoist load with loop-invariant address because the loop may invalidate its value
241                         Pos.z -= LinP->z;
    100%  licm                  failed to hoist load with loop-invariant address because the loop may invalidate its value
    20%   gvn                   getelementptr eliminated by PRE
242                         A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
    100%  licm                      failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm                          failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  licm                              failed to hoist load with loop-invariant address because the loop may invalidate its value
    100%  gvn                               load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn                               load of type double not eliminated in favor of load because it is clobbered by store
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
245                         if( (A = C + B*B) > 0.0 ) {         /* ...else no hit */
246                             A = sqrt(A);
    9%   inline                  sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;         /* Calculate intersection point */
    5%   slp-vectorizer             Stores SLP vectorized
251                                 Pnt->y = LinD->y*t;
    5%   licm                          failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn                           load of type double eliminated in favor of load
252                                 Pnt->z = LinD->z*t;
    5%   licm                          failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn                           load of type double eliminated in favor of load
    100%  gvn                       load of type double not eliminated in favor of load because it is clobbered by store
    100%  gvn                       load of type double not eliminated in favor of load because it is clobbered by store
253                                 Norm->x = Pnt->x-Pos.x;     /* Calcualate surface normal */
    5%   licm                          failed to hoist load with loop-invariant address because the loop may invalidate its value
    5%   gvn                           load of type double eliminated in favor of fmul
```

Approximate the opportunity by manually modifying the source

```
211                 static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
      99%  version-licm     Loop memory access not suitable
      99%  loop-vectorize   loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                   /* Translate object into "line-space" */
      100%  licm            failed to hoist load with loop-invariant address because the loop may invalidate its value
240                         Pos.y -= LinP->y;
      100%  licm            failed to hoist load with loop-invariant address because the loop may invalidate its value
241                         Pos.z -= LinP->z;
      100%  licm            failed to hoist load with loop-invariant address because the loop may invalidate its value
      20%   gvn             getelementptr eliminated by PRE
242                         A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
      100%  licm            failed to hoist load with loop-invariant address because the loop may invalidate its value
      100%  licm            failed to hoist load with loop-invariant address because the loop may invalidate its value
      100%  licm            failed to hoist load with loop-invariant address because the loop may invalidate its value
      100%  gvn             load of type double not eliminated in favor of load because it is clobbered by store
      100%  gvn             load of type double not eliminated in favor of load because it is clobbered by store
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
245                         if( (A = C + B*B) > 0.0 ) {         /* ...else no hit */
246                             A = sqrt(A);
      9%   inline               sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;         /* Calculate intersection point */
      5%   slp-vectorizer           Stores SLP vectorized
251                                 Pnt->y = LinD->y*t;
      5%   licm                        failed to hoist load with loop-invariant address because the loop may invalidate its value
      5%   gvn                         load of type double eliminated in favor of load
252                                 Pnt->z = LinD->z*t;
      5%   licm                        failed to hoist load with loop-invariant address because the loop may invalidate its value
      5%   gvn                         load of type double eliminated in favor of load
      100%  gvn                       load of type double not eliminated in favor of load because it is clobbered by store
      100%  gvn                       load of type double not eliminated in favor of load because it is clobbered by store
253                                 Norm->x = Pnt->x-Pos.x;     /* Calcualate surface normal */
      5%   licm                        failed to hoist load with loop-invariant address because the loop may invalidate its value
      5%   gvn                         load of type double eliminated in favor of fmul
```

```
211                 static double IntersectObjs( VECTOR * restrict LinP, VECTOR * restrict LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
        99%  loop-vectorize  loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                   /* Translate object into "line-space" */
        100%          licm                  load hoisted
240                         Pos.y -= LinP->y;
        100%          licm                  load hoisted
241                         Pos.z -= LinP->z;
        100%          licm                  load hoisted
        20%            gvn                  load eliminated by PRE
        20%            gvn                  getelementptr eliminated by PRE
242                         A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
        100%          licm                    load hoisted
        100%          licm                     fmul hoisted
        100%          licm                                  load hoisted
        100%          licm                                   fmul hoisted
        100%          licm                                              fadd hoisted
        100%          licm                                                           load hoisted
        100%          licm                                                            fmul hoisted
        100%          licm                                                                      fadd hoisted
        100%          licm                 fdiv hoisted
        20%            gvn                                                            load of type double eliminated in favor of load
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
245                         if( (A = C + B*B) > 0.0 ) {         /* ...else no hit */
246                             A = sqrt(A);
        9%           inline                 sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;         /* Calculate intersection point */
251                                 Pnt->y = LinD->y*t;
        5%            licm                                  failed to hoist load with loop-invariant address because load is conditionally executed
        5%             gvn                                  load of type double eliminated in favor of phi
252                                 Pnt->z = LinD->z*t;
        5%            licm                                  failed to hoist load with loop-invariant address because load is conditionally executed
        5%             gvn                                  load of type double eliminated in favor of phi
253                                 Norm->x = Pnt->x-Pos.x;     /* Calcualate surface normal */
```

```
211                 static double IntersectObjs( VECTOR * restrict LinP, VECTOR * restrict LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
        99%  loop-vectorize  loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                   /* Translate object into "line-space" */
        100%          licm                load hoisted
240                         Pos.y -= LinP->y;
        100%          licm                load hoisted
241                         Pos.z -= LinP->z;
        100%          licm                load hoisted
        20%           gvn                 load eliminated by PRE
        20%           gvn                 getelementptr eliminated by PRE
242                         A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
        100%          licm                load hoisted
        100%          licm                 fmul hoisted
        100%          licm                                load hoisted
        100%          licm                                 fmul hoisted
        100%          licm                              fadd hoisted
        100%          licm                                                load hoisted
        100%          licm                                                 fmul hoisted
        100%          licm                                              fadd hoisted
        100%          licm                fdiv hoisted
        20%           gvn                                                 load of type double eliminated in favor of load
243                         B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
244                         C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
245                         if( (A = C + B*B) > 0.0 ) {         /* ...else no hit */
246                             A = sqrt(A);
        9%           inline                 sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                             if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                             if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                                 t = ttmp;
250                                 Pnt->x = LinD->x*t;         /* Calculate intersection point */
251                                 Pnt->y = LinD->y*t;
        5%           licm                                    failed to hoist load with loop-invariant address because load is conditionally executed
        5%           gvn                                     load of type double eliminated in favor of phi
252                                 Pnt->z = LinD->z*t;
        5%           licm                                    failed to hoist load with loop-invariant address because load is conditionally executed
        5%           gvn                                     load of type double eliminated in favor of phi
253                                 Norm->x = Pnt->x-Pos.x;     /* Calcualate surface normal */
```

```
211                 static double IntersectObjs( VECTOR * restrict LinP, VECTOR * restrict LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
        99%  loop-vectorize  loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                    /* Translate object into "line-space" */
        100%          licm                load hoisted
240                         Pos.y -= LinP->y;
        100%          licm                load hoisted
241
        100%          licm
        20%           gvn
        20%           gvn
242
        100%          licm
        100%          licm
        100%          licm
        100%          licm
        100%          licm
        100%          licm
        100%          licm
        100%          licm
        100%          licm
        20%           gvn
243
244
245
246                 A = sqrt(A);
        9%          inline  sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                 if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                 if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                     t = ttmp;
250                     Pnt->x = LinD->x*t;        /* Calculate intersection point */
251                     Pnt->y = LinD->y*t;
        5%          licm  failed to hoist load with loop-invariant address because load is conditionally executed
        5%          gvn   load of type double eliminated in favor of phi
252                     Pnt->z = LinD->z*t;
        5%          licm  failed to hoist load with loop-invariant address because load is conditionally executed
        5%          gvn   load of type double eliminated in favor of phi
253                     Norm->x = Pnt->x-Pos.x;    /* Calcualate surface normal */
```

Dynamic Instruction Count Reduced by **11%**

```
211                 static double IntersectObjs( VECTOR * restrict LinP, VECTOR * restrict LinD,
212                                              VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
213                 {
237                     for( objn = 0; objn < NUMOBJS; objn++ ) {
        99%  loop-vectorize  loop not vectorized: vectorization is not beneficial and is not explicitly forced
238                         Pos = objs[objn].pos;
239                         Pos.x -= LinP->x;                   /* Translate object into "line-space" */
        100%          licm                load hoisted
240                         Pos.y -= LinP->y;
        100%          licm                load hoisted
241
        100%          licm
        20%           gvn
        20%           gvn
242
        100%          licm
        100%          licm
        100%          licm
        100%          licm
        100%          licm
        100%          licm
        100%          licm
        100%          licm
        100%          licm
        20%           gvn
243
244
245
246                 A = sqrt(A);
        9%          inline  sqrt will not be inlined into IntersectObjs because its definition is unavailable
247                 if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
248                 if( (EPSILON<ttmp) && ( (ttmp<t)||(t<0.0) ) ) {
249                     t = ttmp;
250                     Pnt->x = LinD->x*t;         /* Calculate intersection point */
251                     Pnt->y = LinD->y*t;
        5%          licm    failed to hoist load with loop-invariant address because load is conditionally executed
        5%          gvn     load of type double eliminated in favor of phi
252                     Pnt->z = LinD->z*t;
        5%          licm    failed to hoist load with loop-invariant address because load is conditionally executed
        5%          gvn     load of type double eliminated in favor of phi
253                     Norm->x = Pnt->x-Pos.x;     /* Calcualate surface normal */
```

Performance headroom

11%

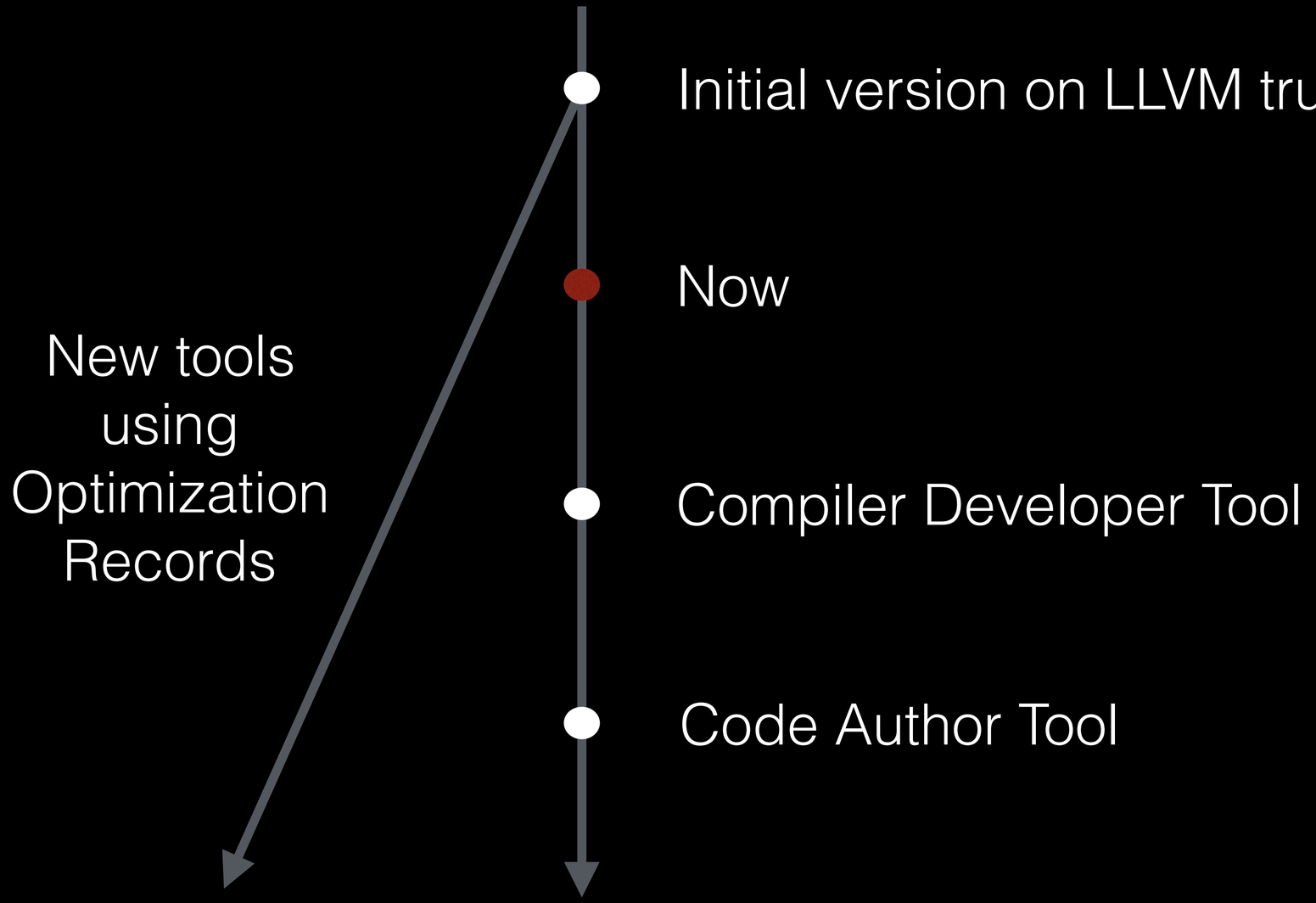
Freebench/distray: Summary

- Found optimization opportunity while staying in the high-level view
  - Reconstructed the reason for missed optimization
  - High-level view exposed that the gain may be substantial
  - Got immediate feedback of the desired effect on the prototype
  - Identified the pass for low-level debugging

Check Out More Examples

http://lab.llvm.org:8080/artifacts/opt-view_test-suite

Development Timeline

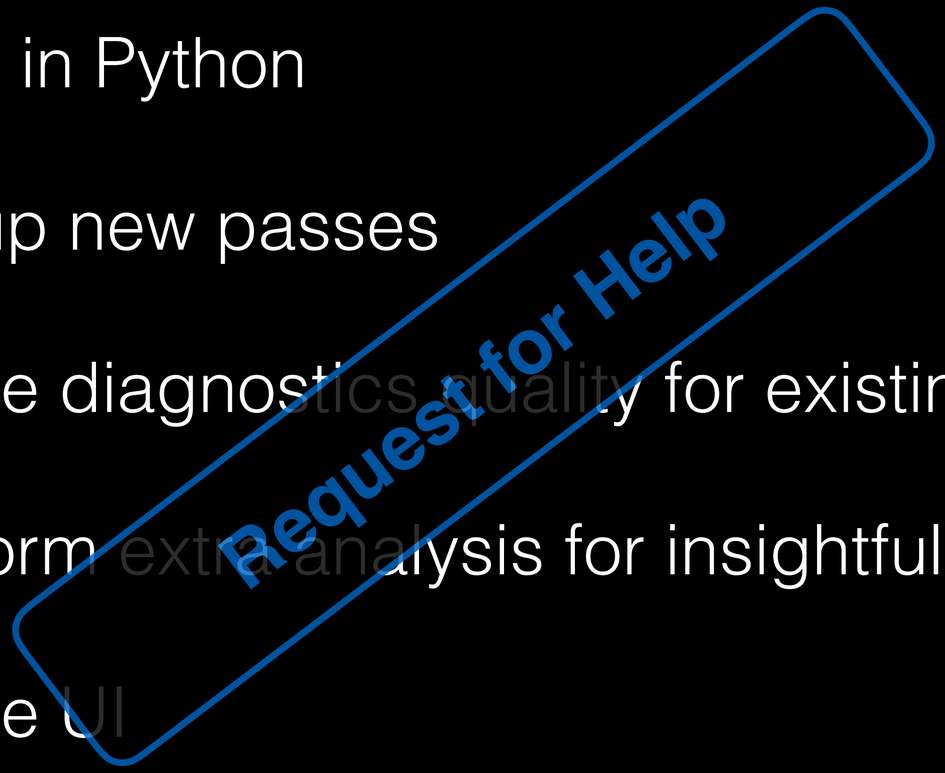


Compiler Developer Tool: Status

- Written in Python
- Hook up new passes
- Improve diagnostics quality for existing passes
  - Perform extra analysis for insightful messages
- Improve UI

Compiler Developer Tool: Status

- Written in Python
- Hook up new passes
- Improve diagnostics quality for existing passes
  - Perform extra analysis for insightful messages
- Improve UI

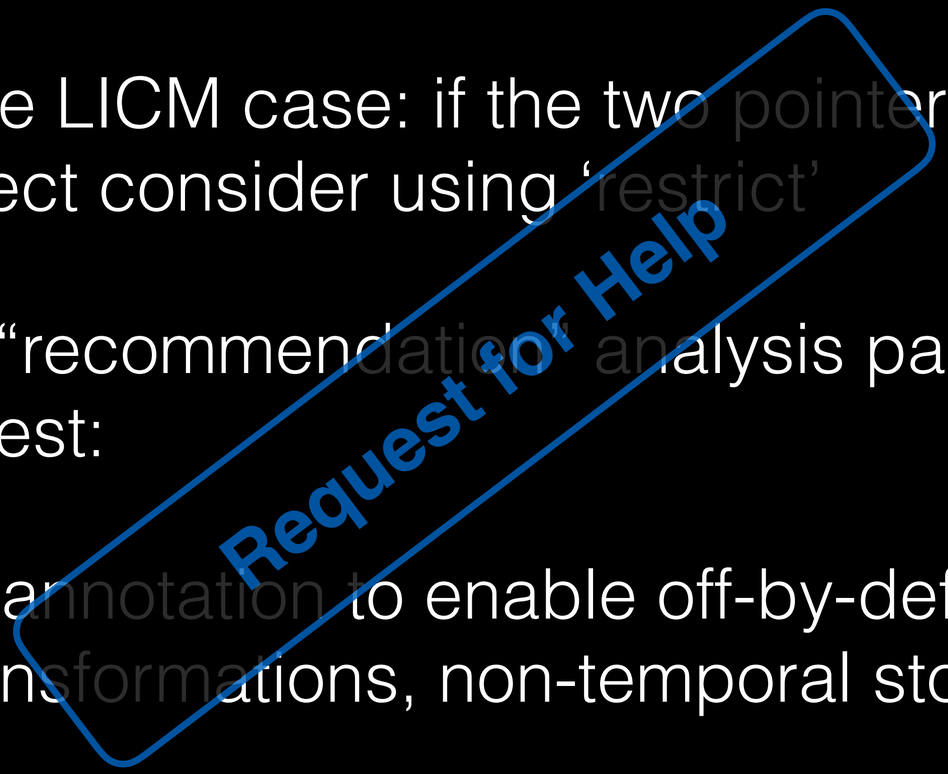


Code Author Tool: Wishlist

- Suggest specific actions
  - E.g. for the LICM case: if the two pointers can never point to the same object consider using ‘restrict’
  - Add new “recommendation” analysis passes to detect opportunity and suggest:
    - Source annotation to enable off-by-default passes (aggressive loop transformations, non-temporal stores)
    - Refactoring: data transformations

Code Author Tool: Wishlist

- Suggest specific actions
  - E.g. for the LICM case: if the two pointers can never point to the same object consider using ‘restrict’
  - Add new “recommendation” analysis passes to detect opportunity and suggest:
    - Source annotation to enable off-by-default passes (aggressive loop transformations, non-temporal stores)
    - Refactoring: data transformations



Optimization Records: New Tools

- `llvm-opt-report`
- Performance regression analysis
- Optimization statistics with the ability to zoom into the particular optimization
- Bottom-up search for performance opportunities
  - See all the LICM opportunities like in Freebench/distray

Optimization Records: New Tools

- `llvm-opt-report`
- Performance regression analysis
- Optimizatio
  optimizatio
- Bottom-up
  - See all the LICM opportunities like in Freebench/distray

```
SELECT   benchmark, hotspot, hotness
FROM     optimizations
WHERE    pass = ‘licm’ AND
         type = ‘missed’ AND
         name = ‘LoadWithLoopInvariantAddressInvalidated’
ORDER BY hotness
```

Optimization Records: New Tools

- `llvm-opt-report`
- Performance regression analysis
- Optimization statistics with the ability to zoom into the particular optimization
- Bottom-up search for performance opportunities
  - See all the LICM opportunities like in Freebench/distray
  - Allows finding opportunities that occur with high frequency but not in the hottest code

Acknowledgement

- Tyler Nowicki
- John McCall
- Hal Finkel

Q&A

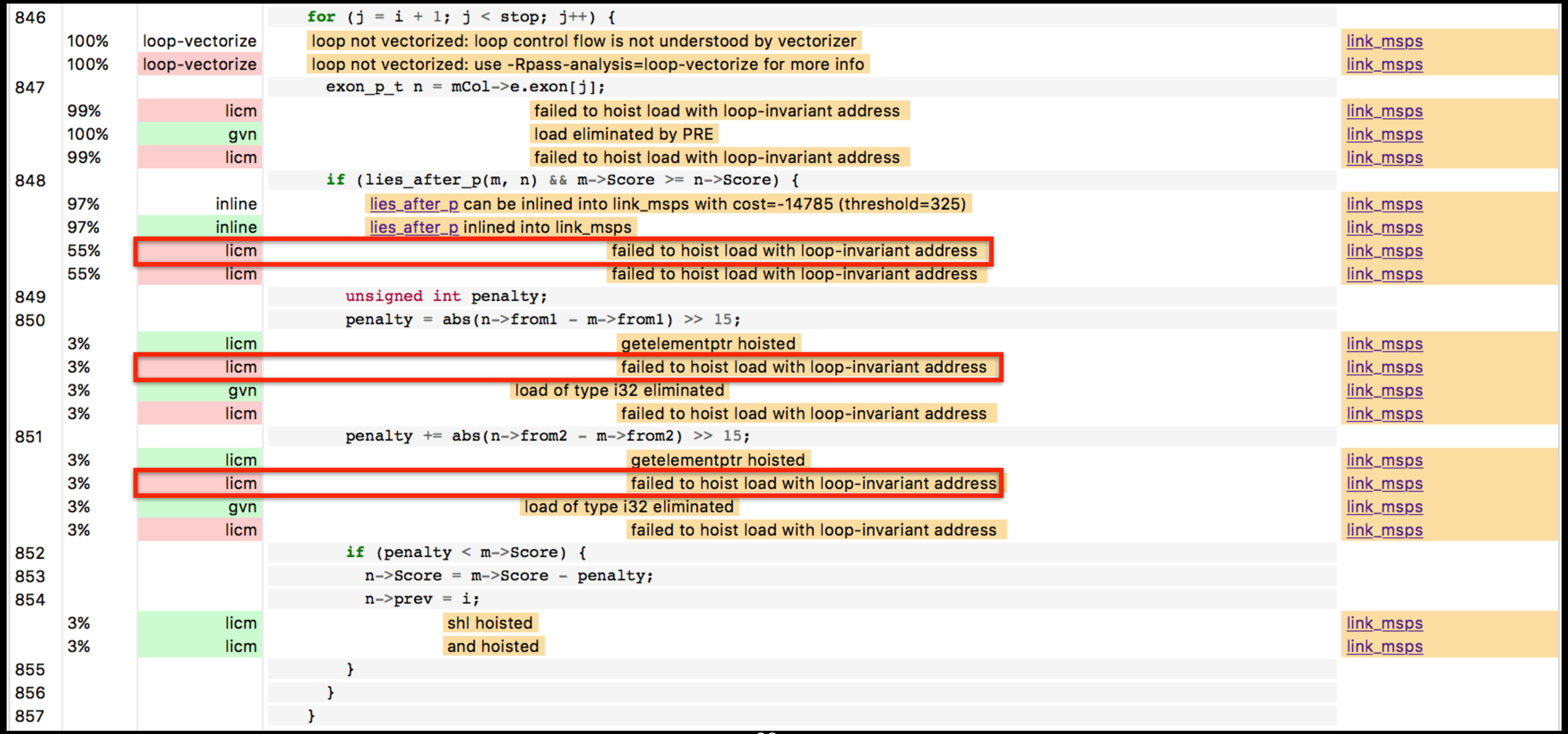
SIBsim4
(MultiSource/Applications)

Finding Performance Opportunity

SIBsim4

846			`for (j = i + 1; j < stop; j++) {`	
	100%	loop-vectorize	loop not vectorized: loop control flow is not understood by vectorizer	link_msps
	100%	loop-vectorize	loop not vectorized: use -Rpass-analysis=loop-vectorize for more info	link_msps
847			`exon_p_t n = mCol->e.exon[j];`	
	99%	licm	failed to hoist load with loop-invariant address	link_msps
	100%	gvn	load eliminated by PRE	link_msps
	99%	licm	failed to hoist load with loop-invariant address	link_msps
848			`if (lies_after_p(m, n) && m->Score >= n->Score) {`	
	97%	inline	lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)	link_msps
	97%	inline	lies_after_p inlined into link_msps	link_msps
	55%	licm	failed to hoist load with loop-invariant address	link_msps
	55%	licm	failed to hoist load with loop-invariant address	link_msps
849			`unsigned int penalty;`	
850			`penalty = abs(n->from1 - m->from1) >> 15;`	
	3%	licm	getelementptr hoisted	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
	3%	gvn	load of type i32 eliminated	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
851			`penalty += abs(n->from2 - m->from2) >> 15;`	
	3%	licm	getelementptr hoisted	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
	3%	gvn	load of type i32 eliminated	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
852			`if (penalty < m->Score) {`	
853			`n->Score = m->Score - penalty;`	
854			`n->prev = i;`	
	3%	licm	shl hoisted	link_msps
	3%	licm	and hoisted	link_msps
855			`}`	
856			`}`	
857			`}`	

SIBsim4


846 for (j = i + 1; j < stop; j++) {
100% loop-vectorize loop not vectorized: loop control flow is not understood by vectorizer link_msps
100% loop-vectorize loop not vectorized: use -Rpass-analysis=loop-vectorize for more info link_msps
847 exon_p_t n = mCol->e.exon[j];
99% licm failed to hoist load with loop-invariant address link_msps
100% gvn load eliminated by PRE link_msps
99% licm failed to hoist load with loop-invariant address link_msps
848 if (lies_after_p(m, n) && m->Score >= n->Score) {
97% inline lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325) link_msps
97% inline lies_after_p inlined into link_msps link_msps
55% licm failed to hoist load with loop-invariant address link_msps
55% licm failed to hoist load with loop-invariant address link_msps
849 unsigned int penalty;
850 penalty = abs(n->from1 - m->from1) >> 15;
3% licm getelementptr hoisted link_msps
3% licm failed to hoist load with loop-invariant address link_msps
3% gvn load of type i32 eliminated link_msps
3% licm failed to hoist load with loop-invariant address link_msps
851 penalty += abs(n->from2 - m->from2) >> 15;
3% licm getelementptr hoisted link_msps
3% licm failed to hoist load with loop-invariant address link_msps
3% gvn load of type i32 eliminated link_msps
3% licm failed to hoist load with loop-invariant address link_msps
852 if (penalty < m->Score) {
853 n->Score = m->Score - penalty;
854 n->prev = i;
3% licm shl hoisted link_msps
3% licm and hoisted link_msps
855 }
856 }
857 }

SIBsim4

846			`for (j = i + 1; j < stop; j++) {`	
	100%	loop-vectorize	loop not vectorized: loop control flow is not understood by vectorizer	link_msps
	100%	loop-vectorize	loop not vectorized: use -Rpass-analysis=loop-vectorize for more info	link_msps
847			`exon_p_t n = mCol->e.exon[j];`	
	99%	licm	failed to hoist load with loop-invariant address	link_msps
	100%	gvn	load eliminated by PRE	link_msps
	99%	licm	failed to hoist load with loop-invariant address	link_msps
848			`if (lies_after_p(m, n) && m->Score >= n->Score) {`	
	97%	inline	lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)	link_msps
	97%	inline	lies_after_p inlined into link_msps	link_msps
	55%	licm	failed to hoist load with loop-invariant address	link_msps
	55%	licm	failed to hoist load with loop-invariant address	link_msps
849			`unsigned int penalty;`	
850			`penalty = abs(n->from1 - m->from1) >> 15;`	
	3%	licm	getelementptr hoisted	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
	3%	gvn	load of type i32 eliminated	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
851			`penalty += abs(n->from2 - m->from2) >> 15;`	
	3%	licm	getelementptr hoisted	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
	3%	gvn	load of type i32 eliminated	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
852			`if (penalty < m->Score) {`	
853			`n->Score = m->Score - penalty;`	
854			`n->prev = i;`	
	3%	licm	shl hoisted	link_msps
	3%	licm	and hoisted	link_msps
855			`}`	
856			`}`	
857			`}`	

SIBsim4

846			`for (j = i + 1; j < stop; j++) {`	
	100%	loop-vectorize	loop not vectorized: loop control flow is not understood by vectorizer	link_msps
	100%	loop-vectorize	loop not vectorized: use -Rpass-analysis=loop-vectorize for more info	link_msps
847			`exon_p_t n = mCol->e.exon[j];`	
	99%	licm	failed to hoist load with loop-invariant address	link_msps
	100%	gvn	load eliminated by PRE	link_msps
	99%	licm	failed to hoist load with loop-invariant address	link_msps
848			`if (lies_after_p(m, n) && m->Score >= n->Score) {`	
	97%	inline	lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)	link_msps
	97%	inline	lies_after_p inlined into link_msps	link_msps
	55%	licm	failed to hoist load with loop-invariant address	link_msps
	55%	licm	failed to hoist load with loop-invariant address	link_msps
849			`unsigned int penalty;`	
850			`penalty = abs(n->from1 - m->from1) >> 15;`	
	3%	licm	getelementptr hoisted	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
	3%	gvn	load of type i32 eliminated	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
851			`penalty += abs(n->from2 - m->from2) >> 15;`	
	3%	licm	getelementptr hoisted	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
	3%	gvn	load of type i32 eliminated	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
852			`if (penalty < m->Score) {`	
853			`n->Score = m->Score - penalty;`	
854			`n->prev = i;`	
	3%	licm	shl hoisted	link_msps
	3%	licm	and hoisted	link_msps
855			`}`	
856			`}`	
857			`}`	

SIBsim4


712 static inline int
713 lies_after_p(exon_p_t a, exon_p_t b)
714 {
715 /* When we have some overlap, make sure it is only a small part. */
716 /* -----------------
717 ---------------------
718 | p1 | p2 | p3 | */
719
720 if (b->from1 > a->to1) {
99% licm getelementptr hoisted link msps
99% licm failed to hoist load with loop-invariant address link msps
100% licm failed to hoist load with loop-invariant address link msps
721 unsigned int p1;
722 unsigned int p2;
723 unsigned int p3;
724 if (b->from2 > a->to2)
99% licm getelementptr hoisted link msps
99% licm failed to hoist load with loop-invariant address link msps
100% licm failed to hoist load with loop-invariant address link msps
725 return 1;
726 if (b->from2 < a->from2 || b->to2 < a->to2)
41% licm getelementptr hoisted link msps
41% licm failed to hoist load with loop-invariant address link msps
41% licm failed to hoist load with loop-invariant address link msps
727 return 0;
728 p1 = b->from2 - a->from2;
729 p2 = a->to2 - b->from2;
730 p3 = b->to2 - a->to2;
731 if (p1 > p2 && p3 > p2 && p1 > options.K && p3 > options.K)
732 return 1;
733 } else if (b->from2 > a->to2) {
734 unsigned int p1;
735 unsigned int p2;
736 unsigned int p3;
737 if (b->from1 < a->from1 || b->to1 < a->to1)
2% licm getelementptr hoisted link msps
2% licm failed to hoist load with loop-invariant address link msps
2% licm failed to hoist load with loop-invariant address link msps
738 return 0;
739 p1 = b->from1 - a->from1;
740 p2 = a->to1 - b->from1;
741 p3 = b->to1 - a->to1;
742 if (p1 > p2 && p3 > p2 && p1 > options.K && p3 > options.K)
743 return 1;
744 }
745 return 0;
746 }

SIBsim4

846			`for (j = i + 1; j < stop; j++) {`	
	100%	loop-vectorize	loop not vectorized: loop control flow is not understood by vectorizer	link_msps
	100%	loop-vectorize	loop not vectorized: use -Rpass-analysis=loop-vectorize for more info	link_msps
847			`exon_p_t n = mCol->e.exon[j];`	
	99%	licm	failed to hoist load with loop-invariant address	link_msps
	100%	gvn	load eliminated by PRE	link_msps
	99%	licm	failed to hoist load with loop-invariant address	link_msps
848			`if (lies_after_p(m, n) && m->Score >= n->Score) {`	
	97%	inline	lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)	link_msps
	97%	inline	lies_after_p inlined into link_msps	link_msps
	55%	licm	failed to hoist load with loop-invariant address	link_msps
	55%	licm	failed to hoist load with loop-invariant address	link_msps
849			`unsigned int penalty;`	
850			`penalty = abs(n->from1 - m->from1) >> 15;`	
	3%	licm	getelementptr hoisted	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
	3%	gvn	load of type i32 eliminated	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
851			`penalty += abs(n->from2 - m->from2) >> 15;`	
	3%	licm	getelementptr hoisted	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
	3%	gvn	load of type i32 eliminated	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
852			`if (penalty < m->Score) {`	
853			`n->Score = m->Score - penalty;`	
854			`n->prev = i;`	
	3%	licm	shl hoisted	link_msps
	3%	licm	and hoisted	link_msps
855			`}`	
856			`}`	
857			`}`	

SIBsim4

```
846                                  for (j = i + 1; j < stop; j++) {
      100%  loop-vectorize           loop not vectorized: loop control flow is not understood by vectorizer   link_msps
      100%  loop-vectorize           loop not vectorized: use -Rpass-analysis=loop-vectorize for more info    link_msps
847                                    exon_p_t n = mCol->e.exon[j];
       99%            licm                              failed to hoist load with loop-invariant address    link_msps
      100%             gvn                              load eliminated by PRE                               link_msps
       99%            licm                              failed to hoist load with loop-invariant address    link_msps
848                                    if (lies_after_p(m, n) && m->Score >= n->Score) {
       97%          inline             lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)   link_msps
       97%          inline             lies_after_p inlined into link_msps                                  link_msps
       55%            licm                                failed to hoist load with loop-invariant address  link_msps
       55%            licm                                failed to hoist load with loop-invariant address  link_msps
849                                      unsigned int penalty;
850                                      penalty = abs(n->from1 - m->from1) >> 15;
        3%            licm                                getelementptr hoisted                              link_msps
        3%            licm                                failed to hoist load with loop-invariant address   link_msps
        3%             gvn                          load of type i32 eliminated                               link_msps
        3%            licm                                failed to hoist load with loop-invariant address   link_msps
851                                      penalty += abs(n->from2 - m->from2) >> 15;
        3%            licm                                 getelementptr hoisted                             link_msps
        3%            licm                                 failed to hoist load with loop-invariant address  link_msps
        3%             gvn                           load of type i32 eliminated                              link_msps
        3%            licm                                 failed to hoist load with loop-invariant address  link_msps
852                                      if (penalty < m->Score) {
853                                        n->Score = m->Score - penalty;
854                                        n->prev = i;
        3%            licm                          shl hoisted                                              link_msps
        3%            licm                          and hoisted                                              link_msps
855                                      }
856                                    }
857                                  }
```

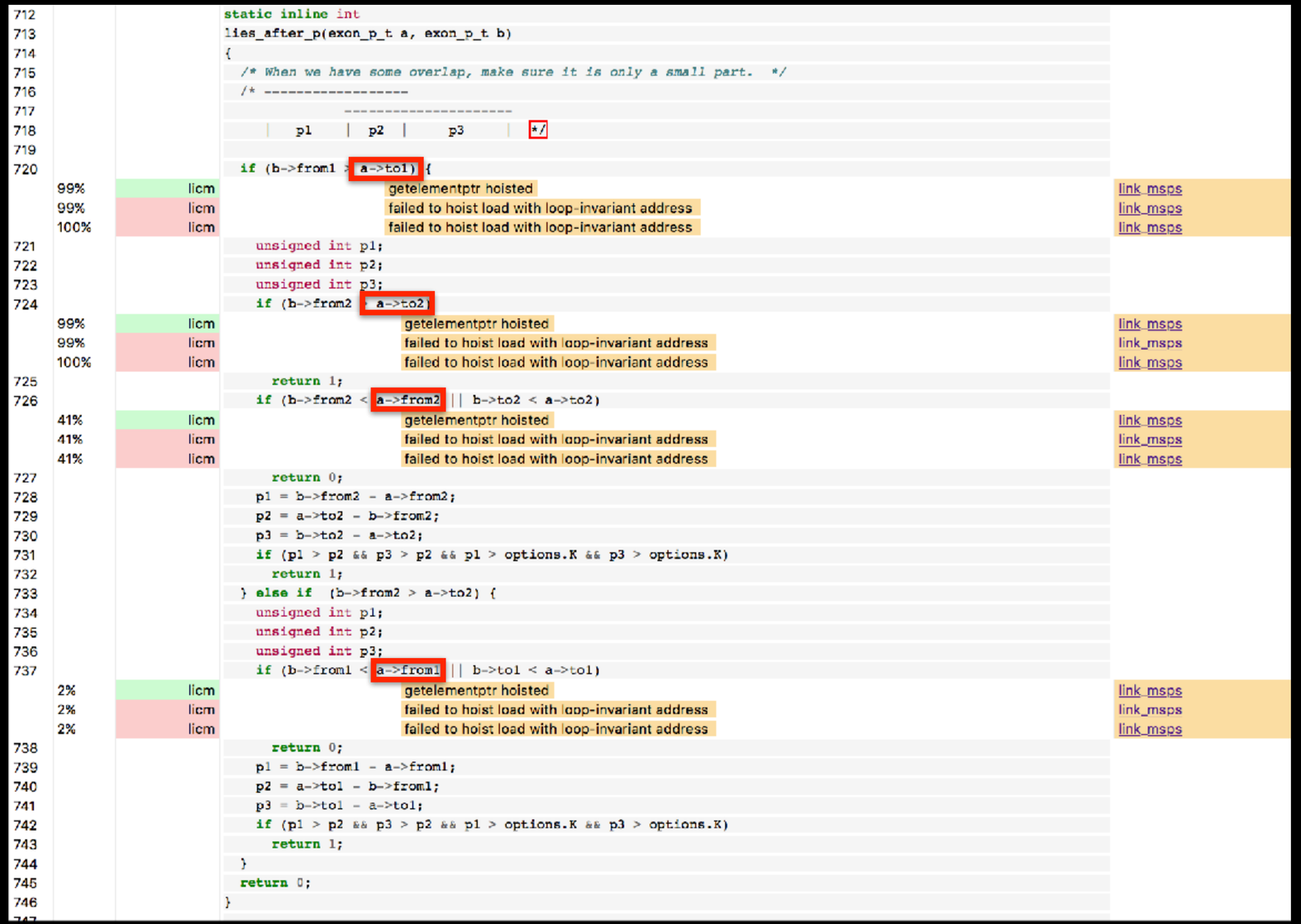
Look at the loads

SIBsim4

846			`for (j = i + 1; j < stop; j++) {`	
	100%	loop-vectorize	loop not vectorized: loop control flow is not understood by vectorizer	link_msps
	100%	loop-vectorize	loop not vectorized: use -Rpass-analysis=loop-vectorize for more info	link_msps
847			`exon_p_t n = mCol->e.exon[j];`	
	99%	licm	failed to hoist load with loop-invariant address	link_msps
	100%	gvn	load eliminated by PRE	link_msps
	99%	licm	failed to hoist load with loop-invariant address	link_msps
848			`if (lies_after_p(m, n) && m->Score >= n->Score) {`	
	97%	inline	lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)	link_msps
	97%	inline	lies_after_p inlined into link_msps	link_msps
	55%	licm	failed to hoist load with loop-invariant address	link_msps
	55%	licm	failed to hoist load with loop-invariant address	link_msps
849			`unsigned int penalty;`	
850			`penalty = abs(n->from1 - m->from1) >> 15;`	
	3%	licm	getelementptr hoisted	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
	3%	gvn	load of type i32 eliminated	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
851			`penalty += abs(n->from2 - m->from2) >> 15;`	
	3%	licm	getelementptr hoisted	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
	3%	gvn	load of type i32 eliminated	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
852			`if (penalty < m->Score) {`	
853			`n->Score = m->Score - penalty;`	
854			`n->prev = i;`	
	3%	licm	shl hoisted	link_msps
	3%	licm	and hoisted	link_msps
855			`}`	
856			`}`	
857			`}`	

Look at the loads

SIBsim4



```
712                 static inline int
713                 lies_after_p(exon_p_t a, exon_p_t b)
714                 {
715                   /* When we have some overlap, make sure it is only a small part.  */
716                   /* -----------------
717                                      ---------------------
718                          p1     |  p2  |      p3       |  */
719
720                   if (b->from1 > a->to1) {
       99%  licm                            getelementptr hoisted                          link msps
       99%  licm                            failed to hoist load with loop-invariant address  link msps
       100% licm                            failed to hoist load with loop-invariant address  link msps
721                     unsigned int p1;
722                     unsigned int p2;
723                     unsigned int p3;
724                     if (b->from2 > a->to2)
       99%  licm                              getelementptr hoisted                        link msps
       99%  licm                              failed to hoist load with loop-invariant address  link msps
       100% licm                              failed to hoist load with loop-invariant address  link msps
725                       return 1;
726                     if (b->from2 < a->from2 || b->to2 < a->to2)
       41%  licm                              getelementptr hoisted                        link msps
       41%  licm                              failed to hoist load with loop-invariant address  link msps
       41%  licm                              failed to hoist load with loop-invariant address  link msps
727                       return 0;
728                     p1 = b->from2 - a->from2;
729                     p2 = a->to2 - b->from2;
730                     p3 = b->to2 - a->to2;
731                     if (p1 > p2 && p3 > p2 && p1 > options.K && p3 > options.K)
732                       return 1;
733                   } else if  (b->from2 > a->to2) {
734                     unsigned int p1;
735                     unsigned int p2;
736                     unsigned int p3;
737                     if (b->from1 < a->from1 || b->to1 < a->to1)
       2%   licm                              getelementptr hoisted                        link msps
       2%   licm                              failed to hoist load with loop-invariant address  link msps
       2%   licm                              failed to hoist load with loop-invariant address  link msps
738                       return 0;
739                     p1 = b->from1 - a->from1;
740                     p2 = a->to1 - b->from1;
741                     p3 = b->to1 - a->to1;
742                     if (p1 > p2 && p3 > p2 && p1 > options.K && p3 > options.K)
743                       return 1;
744                   }
745                   return 0;
746                 }
```

SIBsim4

```
846                                   for (j = i + 1; j < stop; j++) {
      100%  loop-vectorize              loop not vectorized: loop control flow is not understood by vectorizer   link_msps
      100%  loop-vectorize              loop not vectorized: use -Rpass-analysis=loop-vectorize for more info   link_msps
847                                     exon_p_t n = mCol->e.exon[j];
       99%            licm                                failed to hoist load with loop-invariant address   link_msps
      100%             gvn                                load eliminated by PRE                             link_msps
       99%            licm                                failed to hoist load with loop-invariant address   link_msps
848                                     if (lies_after_p(m, n) && m->Score >= n->Score) {
       97%          inline                lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)   link_msps
       97%          inline                lies_after_p inlined into link_msps                                   link_msps
       55%            licm                                  failed to hoist load with loop-invariant address   link_msps
       55%            licm                                  failed to hoist load with loop-invariant address   link_msps
849                                       unsigned int penalty;
850                                       penalty = abs(n->from1 - m->from1) >> 15;
        3%            licm                                  getelementptr hoisted                              link_msps
        3%            licm                                  failed to hoist load with loop-invariant address   link_msps
        3%             gvn                              load of type i32 eliminated                             link_msps
        3%            licm                                  failed to hoist load with loop-invariant address   link_msps
851                                       penalty += abs(n->from2 - m->from2) >> 15;
        3%            licm                                   getelementptr hoisted                             link_msps
        3%            licm                                   failed to hoist load with loop-invariant address  link_msps
        3%             gvn                               load of type i32 eliminated                            link_msps
        3%            licm                                   failed to hoist load with loop-invariant address  link_msps
852                                       if (penalty < m->Score) {
853                                         n->Score = m->Score - penalty;
854                                         n->prev = i;
        3%            licm                          shl hoisted                                                 link_msps
        3%            licm                          and hoisted                                                 link_msps
855                                       }
856                                     }
857                                   }
```

Look at the stores

SIBsim4

```
846                                  for (j = i + 1; j < stop; j++) {
      100%  loop-vectorize             loop not vectorized: loop control flow is not understood by vectorizer   link_msps
      100%  loop-vectorize             loop not vectorized: use -Rpass-analysis=loop-vectorize for more info   link_msps
847                                    exon_p_t n = mCol->e.exon[j];
       99%           licm                                failed to hoist load with loop-invariant address   link_msps
      100%            gvn                                load eliminated by PRE                            link_msps
       99%           licm                                failed to hoist load with loop-invariant address   link_msps
848                                    if (lies_after_p(m, n) && m->Score >= n->Score) {
       97%         inline                 lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)   link_msps
       97%         inline                 lies_after_p inlined into link_msps                                   link_msps
       55%           licm                                  failed to hoist load with loop-invariant address   link_msps
       55%           licm                                  failed to hoist load with loop-invariant address   link_msps
849                                      unsigned int penalty;
850                                      penalty = abs(n->from1 - m->from1) >> 15;
        3%           licm                                  getelementptr hoisted                             link_msps
        3%           licm                                  failed to hoist load with loop-invariant address  link_msps
        3%            gvn                              load of type i32 eliminated                            link_msps
        3%           licm                                  failed to hoist load with loop-invariant address  link_msps
851                                      penalty += abs(n->from2 - m->from2) >> 15;
        3%           licm                                   getelementptr hoisted                            link_msps
        3%           licm                                   failed to hoist load with loop-invariant address link_msps
        3%            gvn                               load of type i32 eliminated                           link_msps
        3%           licm                                   failed to hoist load with loop-invariant address link_msps
852                                      if (penalty < m->Score) {
853                                        n->Score = m->Score - penalty;
854                                        n->prev = i;
        3%           licm                          shl hoisted                                               link_msps
        3%           licm                          and hoisted                                               link_msps
855                                      }
856                                    }
857                                  }
```

Look at the stores

SIBsim4

846			`for (j = i + 1; j < stop; j++) {`	
	100%	loop-vectorize	loop not vectorized: loop control flow is not understood by vectorizer	link_msps
	100%	loop-vectorize	loop not vectorized: use -Rpass-analysis=loop-vectorize for more info	link_msps
847			`exon_p_t n = mCol->e.exon[j];`	
	99%	licm	failed to hoist load with loop-invariant address	link_msps
	100%	gvn	load eliminated by PRE	link_msps
	99%	licm	failed to hoist load with loop-invariant address	link_msps
848			`if (lies_after_p(m, n) && m->Score >= n->Score) {`	
	97%	inline	lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)	link_msps
	97%	inline	lies_after_p inlined into link_msps	link_msps
	55%	licm	failed to hoist load with loop-invariant address	link_msps
	55%	licm	failed to hoist load with loop-invariant address	link_msps
849			`unsigned int penalty;`	
850			`penalty = abs(n->from1 - m->from1) >> 15;`	
	3%	licm	getelementptr hoisted	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
	3%	gvn	load of type i32 eliminated	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
851			`penalty += abs(n->from2 - m->from2) >> 15;`	
	3%	licm	getelementptr hoisted	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
	3%	gvn	load of type i32 eliminated	link_msps
	3%	licm	failed to hoist load with loop-invariant address	link_msps
852			`if (penalty < m->Score) {`	
853			`n->Score = m->Score - penalty;`	
854			`n->prev = i;`	
	3%	licm	shl hoisted	link_msps
	3%	licm	and hoisted	link_msps
855			`}`	
856			`}`	
857			`}`	

Can ‘m’ and ’n’ really alias?

SIBsim4

```
exon_p_t m = mCol->e.exon[i];
```

Probably not!

```
846                                   for (j = i + 1; j < stop; j++) {
       100%  loop-vectorize             loop not vectorized: loop control flow is not understood by vectorizer   link_msps
       100%  loop-vectorize             loop not vectorized: use -Rpass-analysis=loop-vectorize for more info   link_msps
847                                       exon_p_t n = mCol->e.exon[j];
        99%           licm                          failed to hoist load with loop-invariant address            link_msps
       100%            gvn                          load eliminated by PRE                                      link_msps
        99%           licm                          failed to hoist load with loop-invariant address            link_msps
848                                     if (lies_after_p(m, n) && m->Score >= n->Score) {
        97%         inline                lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)   link_msps
        97%         inline                lies_after_p inlined into link_msps                                       link_msps
        55%           licm                              failed to hoist load with loop-invariant address          link_msps
        55%           licm                              failed to hoist load with loop-invariant address          link_msps
849                                       unsigned int penalty;
850                                       penalty = abs(n->from1 - m->from1) >> 15;
         3%           licm                              getelementptr hoisted                                     link_msps
         3%           licm                              failed to hoist load with loop-invariant address          link_msps
         3%            gvn                          load of type i32 eliminated                                 link_msps
         3%           licm                              failed to hoist load with loop-invariant address          link_msps
851                                       penalty += abs(n->from2 - m->from2) >> 15;
         3%           licm                               getelementptr hoisted                                    link_msps
         3%           licm                               failed to hoist load with loop-invariant address         link_msps
         3%            gvn                           load of type i32 eliminated                                 link_msps
         3%           licm                               failed to hoist load with loop-invariant address         link_msps
852                                       if (penalty < m->Score) {
853                                         n->Score = m->Score - penalty;
854                                         n->prev = i;
         3%           licm                                shl hoisted                                             link_msps
         3%           licm                                and hoisted                                             link_msps
855                                       }
856                                     }
857                                   }
```

SIBsim4

```
exon_p_t m = mCol->e.exon[i];
```

We need to use ‘restrict’ or hoist manually

```
846                                   for (j = i + 1; j < stop; j++) {
       100%  loop-vectorize             loop not vectorized: loop control flow is not ur                    link_msps
       100%  loop-vectorize             loop not vectorized: use -Rpass-analysis=loop                       link_msps
847                                     exon_p_t n = mCol->e.exon[j];
       99%           licm                          failed to hoist load with loop-invariant address        link_msps
       100%          gvn                           load eliminated by PRE                                  link_msps
       99%           licm                          failed to hoist load with loop-invariant address        link_msps
848                                   if (lies_after_p(m, n) && m->Score >= n->Score) {
       97%         inline              lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)  link_msps
       97%         inline              lies_after_p inlined into link_msps                                     link_msps
       55%           licm                              failed to hoist load with loop-invariant address      link_msps
       55%           licm                              failed to hoist load with loop-invariant address      link_msps
849                                     unsigned int penalty;
850                                     penalty = abs(n->from1 - m->from1) >> 15;
       3%            licm                              getelementptr hoisted                                 link_msps
       3%            licm                              failed to hoist load with loop-invariant address      link_msps
       3%            gvn                       load of type i32 eliminated                                   link_msps
       3%            licm                              failed to hoist load with loop-invariant address      link_msps
851                                     penalty += abs(n->from2 - m->from2) >> 15;
       3%            licm                              getelementptr hoisted                                 link_msps
       3%            licm                              failed to hoist load with loop-invariant address      link_msps
       3%            gvn                       load of type i32 eliminated                                   link_msps
       3%            licm                              failed to hoist load with loop-invariant address      link_msps
852                                     if (penalty < m->Score) {
853                                       n->Score = m->Score - penalty;
854                                       n->prev = i;
       3%            licm                        shl hoisted                                                 link_msps
       3%            licm                        and hoisted                                                 link_msps
855                                     }
856                                   }
857                                 }
```

SIBsim4

```
846                 exon_t mm = *m;
847                 for (j = i + 1; j < stop; j++) {
848                   exon_p_t n = mCol->e.exon[j];
849                   if (lies_after_p(&mm, n) && mm.Score >= n->Score) {
850                     unsigned int penalty;
851                     penalty = abs(n->from1 - mm.from1) >> 15;
852                     penalty += abs(n->from2 - mm.from2) >> 15;
853                     if (penalty < mm.Score) {
854                       n->Score = mm.Score - penalty;
855                       n->prev = i;
856                     }
857                   }
858                 }
```

SIBsim4

```
846                 exon_t mm = *m;
847                 for (j = i + 1; j < stop; j++) {
848                   exon_p_t n = mCol->e.exon[j];
849                   if (lies_after_p(&mm, n) && mm.Score >= n->Score) {
850                     unsigned int penalty;
851                     penalty = abs(n->from1 - mm.from1) >> 15;
852                     penalty += abs(n->from2 - mm.from2) >> 15;
853                     if (penalty < mm.Score) {
854                       n->Score = mm.Score - penalty;
855                       n->prev = i;
856                     }
857                   }
858                 }
```

SIBsim4

```
846                                  exon_t mm = *m;
847                                  for (j = i + 1; j < stop; j++) {
     100%  loop-vectorize              loop not vectorized: loop control flow is not understood by vectorizer   link_msps
     100%  loop-vectorize              loop not vectorized: use -Rpass-analysis=loop-vectorize for more info    link_msps
848                                    exon_p_t n = mCol->e.exon[j];
      99%  licm                                    failed to hoist load with loop-invariant address because the loop may invalidate its value   link_msps
     100%  gvn                                     load of type %struct._exon_t** not eliminated in favor of load because it is clobbered by store   link_msps
     100%  gvn                                     load eliminated by PRE   link_msps
       0%  gvn                                     load of type %struct._exon_t** not eliminated in favor of load because it is clobbered by store   link_msps
      99%  licm                                    failed to hoist load with loop-invariant address because the loop may invalidate its value   link_msps
849                                    if (lies_after_p(&mm, n) && mm.Score >= n->Score) {
      97%  inline                        lies_after_p can be inlined into link_msps with cost=-14805 (threshold=325)   link_msps
      97%  inline                        lies_after_p inlined into link_msps   link_msps
850                                      unsigned int penalty;
851                                      penalty = abs(n->from1 - mm.from1) >> 15;
       3%  gvn                                       load of type i32 eliminated   link_msps
852                                      penalty += abs(n->from2 - mm.from2) >> 15;
       3%  gvn                                       load of type i32 eliminated   link_msps
853                                      if (penalty < mm.Score) {
854                                        n->Score = mm.Score - penalty;
855                                        n->prev = i;
       3%  licm                                      shl hoisted   link_msps
       3%  licm                                      and hoisted   link_msps
856                                      }
857                                    }
858                                  }
```

SIBsim4

```
712  static inline int
713  lies_after_p(exon_p_t a, exon_p_t b)
714  {
715    /* When we have some overlap, make sure it is only a small part.  */
716    /* ------------------
717                  --------------------
718       |   p1      |   p2   |     p3     |  */
719
720    if (b->from1 > a->to1) {
721      unsigned int p1;
722      unsigned int p2;
723      unsigned int p3;
724      if (b->from2 > a->to2)
725        return 1;
726      if (b->from2 < a->from2 || b->to2 < a->to2)
727        return 0;
728      p1 = b->from2 - a->from2;
729      p2 = a->to2 - b->from2;
730      p3 = b->to2 - a->to2;
731      if (p1 > p2 && p3 > p2 && p1 > options.K && p3 > options.K)
732        return 1;
733    } else if  (b->from2 > a->to2) {
734      unsigned int p1;
735      unsigned int p2;
736      unsigned int p3;
737      if (b->from1 < a->from1 || b->to1 < a->to1)
738        return 0;
739      p1 = b->from1 - a->from1;
740      p2 = a->to1 - b->from1;
741      p3 = b->to1 - a->to1;
742      if (p1 > p2 && p3 > p2 && p1 > options.K && p3 > options.K)
743        return 1;
744    }
745    return 0;
746  }
```