

THE ATTRIBUTOR: A VERSATILE INTER-PROCEDURAL FIXPOINT ITERATION FRAMEWORK

LLVM-Dev'19 — October 22, 2019 — San Jose, CA, USA

Johannes Doerfert*, Hideto Ueno, Stefan Stipanovic

*Leadership Computing Facility
Argonne National Laboratory
<https://www.alcf.anl.gov/>



ACKNOWLEDGMENT

Two of the authors were supported by Google Summer of Code (GSoC)!

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.



I. BACKGROUND

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```

What is the alignment of:

(1) the **return type**?



```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```

What is the alignment of:

(1) the `return type`? (2) the `returned value`?



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```

What is the alignment of:

(1) the `return type`? (2) the `returned value`? (3) the `argument`?



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```

What is the alignment of:

- (1) the return type? (2) the returned value? (3) the argument?
(4) the returned value?



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```

What is the alignment of:

- (1) the return type? (2) the returned value? (3) the argument?
(4) the returned value? (5) the return type?



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```

What is the alignment of:

(1, ∞)

- (1) the return type? (2) the returned value? (3) the argument?
(4) the returned value? (5) the return type?



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```

What is the alignment of:

(1, ∞)

(1, ∞)

- (1) the return type? (2) the returned value? (3) the argument?
(4) the returned value? (5) the return type?



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```

What is the alignment of:

(1, ∞)

(1, ∞)

(16, 16)

(1) the return type? (2) the returned value? (3) the argument?

(4) the returned value? (5) the return type?



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```

What is the alignment of:

(1, ∞)

(16, 16) ← ----- (16, 16)

- (1) the return type? (2) the returned value? (3) the argument?
(4) the returned value? (5) the return type?



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4);  
    return p;  
}
```

What is the alignment of:

(1, ∞)

(16, 16) ← ----- (16, 16)

(1) the return type? (2) the returned value? (3) the argument?

(4) the returned value? (5) the return type?

(1, ∞)



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4);  
    return p;  
}
```

What is the alignment of:

$(1, \infty)$

$(16, 16) \leftarrow \text{-----} (16, 16)$

(1) the `return type`? (2) the `returned value`? (3) the `argument`?

(4) the `returned value`? (5) the `return type`?

$(1, \infty)$

$(1, \infty)$



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4);  
    return p;  
}
```

What is the alignment of:

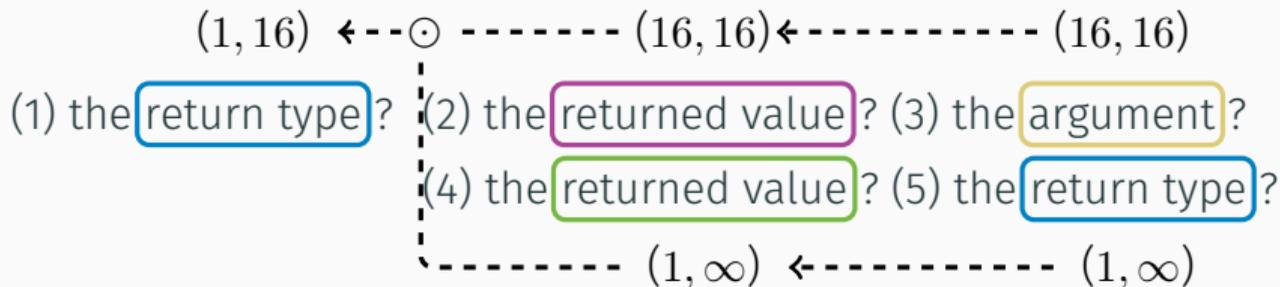
- (1) the `return type`? (2) the `returned value`? (3) the `argument`?
(4) the `returned value`? (5) the `return type`?
- $(1, \infty)$ $(16, 16) \leftarrow \text{-----} (16, 16)$
 $(1, \infty) \leftarrow \text{-----} (1, \infty)$



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4);  
    return p;  
}
```

What is the alignment of:



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4);  
    return p;  
}
```

What is the alignment of:

(1, 16)

(16, 16) ← ----- (16, 16)

(1) the return type? (2) the returned value? (3) the argument?

(4) the returned value? (5) the return type?

(1, ∞)

(1, 16)



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```

What is the alignment of:

(1, 16)

(16, 16) ←----- (16, 16)

(1) the return type? (2) the returned value? (3) the argument?

(4) the returned value? (5) the return type?

(1, 16) ←----- (1, 16)



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```

What is the alignment of:

(1, 16) ← --- ⊙ ----- (16, 16) ← ----- (16, 16)
(1) the return type? (2) the returned value? (3) the argument?
(4) the returned value? (5) the return type?
----- (1, 16) ← ----- (1, 16)



FIXPOINT DATA FLOW ANALYSIS — ALIGNMENT EXAMPLE

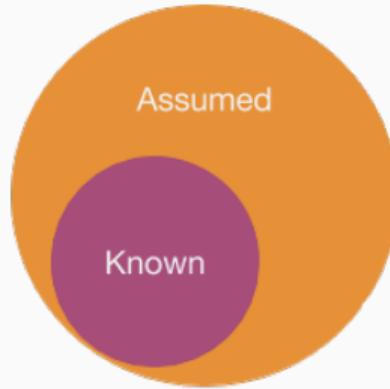
```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```

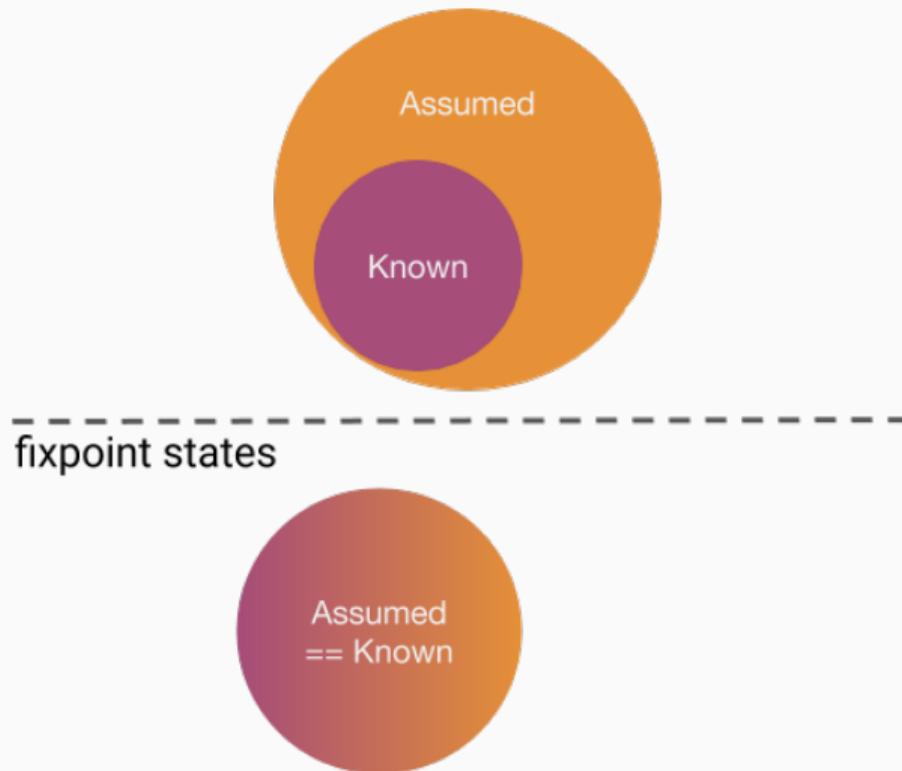
What is the alignment of:

(1) the `return type`? (2) the `returned value`? (3) the `argument`?
(4) the `returned value`? (5) the `return type`?

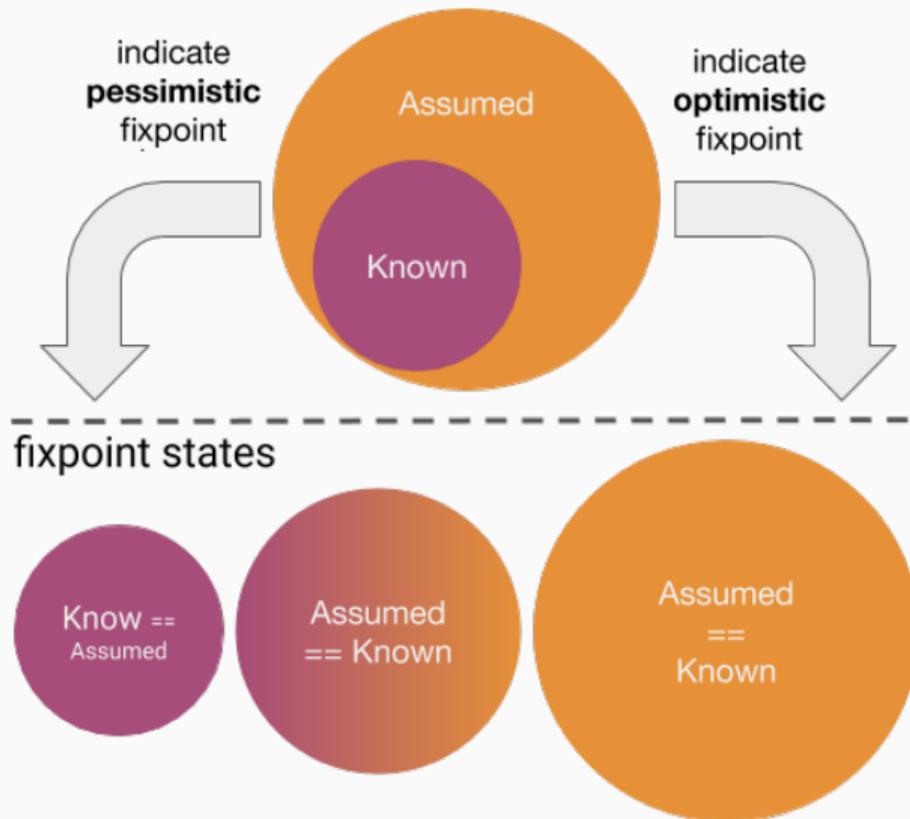








ABSTRACT STATES



```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {  
    if (*p == 0)  
        return checkAndAdvance(p + 4) ;  
    return p ;  
}
```



```
Attributor A;
```

```
// Select what information is to be deduced.
```

```
IRPosition IRPRet = IRPosition::returned(Fn) ;
```

```
const auto &AA = A.getOrCreateAAFor< AAAlign >(IRPRet);
```

```
// Deduce information and manifest it in the IR.
```

```
auto Changed = A.run(*Fn->getParent());
```



```
Attributor A;
```

```
// Select what information is to be deduced.
```

```
IRPosition IRPRet = IRPosition::returned(Fn);
```

```
const auto &AA = A.getOrCreateAAFor< AAAlign >(IRPRet);
```

```
// Deduce information and manifest it in the IR.
```

```
auto Changed = A.run(*Fn->getParent());
```



```
Attributor A;
```

```
// Select what information is to be deduced.
```

```
IRPosition IRPRet = IRPosition::returned(Fn) ;
```

```
const auto &AA = A.getOrCreateAAFor< AAAlign >(IRPRet);
```

```
// Deduce information and manifest it in the IR.
```

```
auto Changed = A.run(*Fn->getParent());
```



```
// Restrict deduction to specific abstract attributes.
```

```
auto Whitelist = {&AAAlign::ID};
```

```
Attributor A(Whitelist);
```

```
// Select what information is to be deduced.
```

```
IRPosition IRPRet = IRPosition::returned(Fn) ;
```

```
const auto &AA = A.getOrCreateAAFor< AAAlign >(IRPRet);
```

```
// Deduce information and manifest it in the IR.
```

```
auto Changed = A.run(*Fn->getParent());
```



```
// Restrict deduction to specific abstract attributes.  
auto Whitelist = {&AAAlign::ID,  
    /* Think IP-SCCP */ &AAIsDead::ID, &AAValueSimplify::ID };  
  
Attributor A(Whitelist);  
  
// Select what information is to be deduced.  
IRPosition IRPRet = IRPosition::returned(Fn) ;  
const auto &AA = A.getOrCreateAAFor< AAAlign >(IRPRet);  
  
// Deduce information and manifest it in the IR.  
auto Changed = A.run(*Fn->getParent());
```



```
// Restrict deduction to specific abstract attributes.  
auto Whitelist = {&AAAlign::ID,  
    /* Think IP-SCCP */ &AAIsDead::ID, &AAValueSimplify::ID };
```

Att **AAAlign** is *unaware* of **AAIsDead** and **AAValueSimplify**!

```
// Select what information is to be deduced.  
IRPosition IRPRet = IRPosition::returned(Fn) ;  
const auto &AA = A.getOrCreateAAFor< AAAlign >(IRPRet);
```

```
// Deduce information and manifest it in the IR.  
auto Changed = A.run(*Fn->getParent());
```



THE ATTRIBUTOR — WHAT IT IS



- easy way to perform fixpoint analyses
 - dependence tracking, work list algorithm, timeouts, ...



- easy way to perform fixpoint analyses
 - dependence tracking, work list algorithm, timeouts, ...
- powerful way to perform fixpoint analyses
 - utilize concurrently deduced information, e.g., liveness



- easy way to perform fixpoint analyses
 - dependence tracking, work list algorithm, timeouts, ...
- powerful way to perform fixpoint analyses
 - utilize concurrently deduced information, e.g., liveness
- alternative to inlining
 - IPO + internalization + function rewriting, e.g., argument promotion



- easy way to perform fixpoint analyses
 - dependence tracking, work list algorithm, timeouts, ...
- powerful way to analyze code
 - utilize concurrency, e.g., parallelism, ...
- alternative to inlining
 - IPO + internalization + function rewriting, e.g., argument promotion

All good, but *why*?

II. MOTIVATION

THE ATTRIBUTOR — THE WHY IPO?



inlining has limits:



inlining has limits:

- recursion



inlining has limits:

- recursion \equiv loops



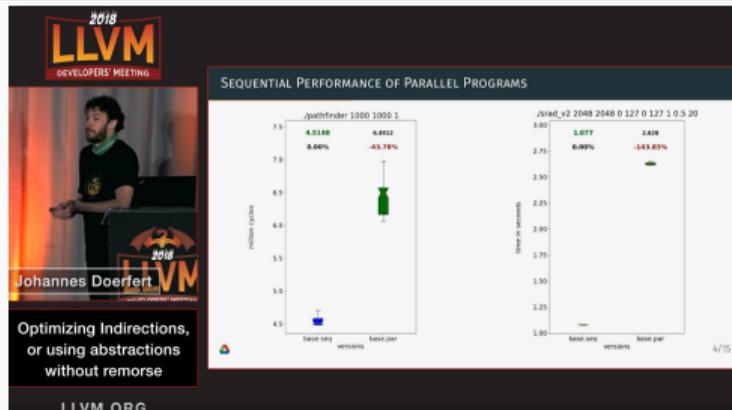
inlining has limits:

- recursion \equiv loops
- code size



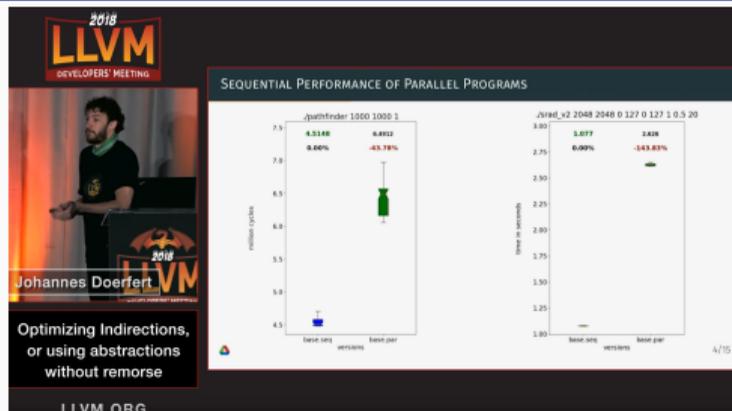
inlining has limits:

- recursion \equiv loops
- code size
- *parallelism* (think `pthread_create`) \uparrow



inlining has limits:

- recursion \equiv loops
- code size
- parallelism (think `pthread_create`) \uparrow
- (declarations) \Rightarrow



"Header Time Optimization": Cross-Translation Unit Optimization via Annotated Headers

William S. Moses, lamson@cs.cmu.edu, Johannes Doerfert, jdoerfert@ant.uni-kl.de, jdoerf@cs.uni-kl.de

4/15/2018, 10:00 AM

Writing Optimizable Code in Header

```
#ifndef HEADER_H
#define HEADER_H

void foo(int x, int y);

#endif
```

This is the header file. It defines a function `foo`. The header is included in the source file. The compiler will generate code for `foo` in the source file. The header is included in the source file. The compiler will generate code for `foo` in the source file.

Inlining "Header Time Optimizations"

The header file is included in the source file. The compiler will generate code for `foo` in the source file. The header is included in the source file. The compiler will generate code for `foo` in the source file.

Experiments

The header file is included in the source file. The compiler will generate code for `foo` in the source file. The header is included in the source file. The compiler will generate code for `foo` in the source file.

Header Files

The header file is included in the source file. The compiler will generate code for `foo` in the source file. The header is included in the source file. The compiler will generate code for `foo` in the source file.

Practical Limitations & Future Work

The header file is included in the source file. The compiler will generate code for `foo` in the source file. The header is included in the source file. The compiler will generate code for `foo` in the source file.

Acknowledgments & References

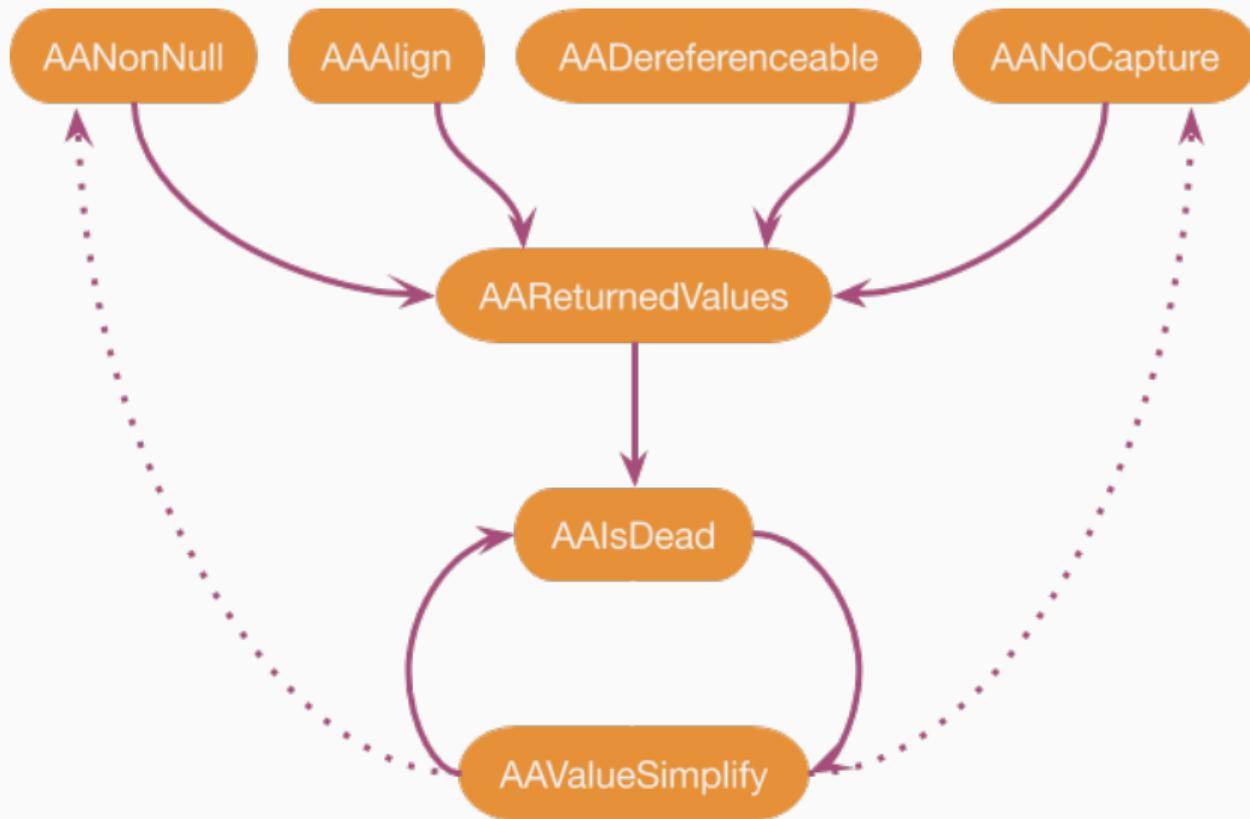
The header file is included in the source file. The compiler will generate code for `foo` in the source file. The header is included in the source file. The compiler will generate code for `foo` in the source file.



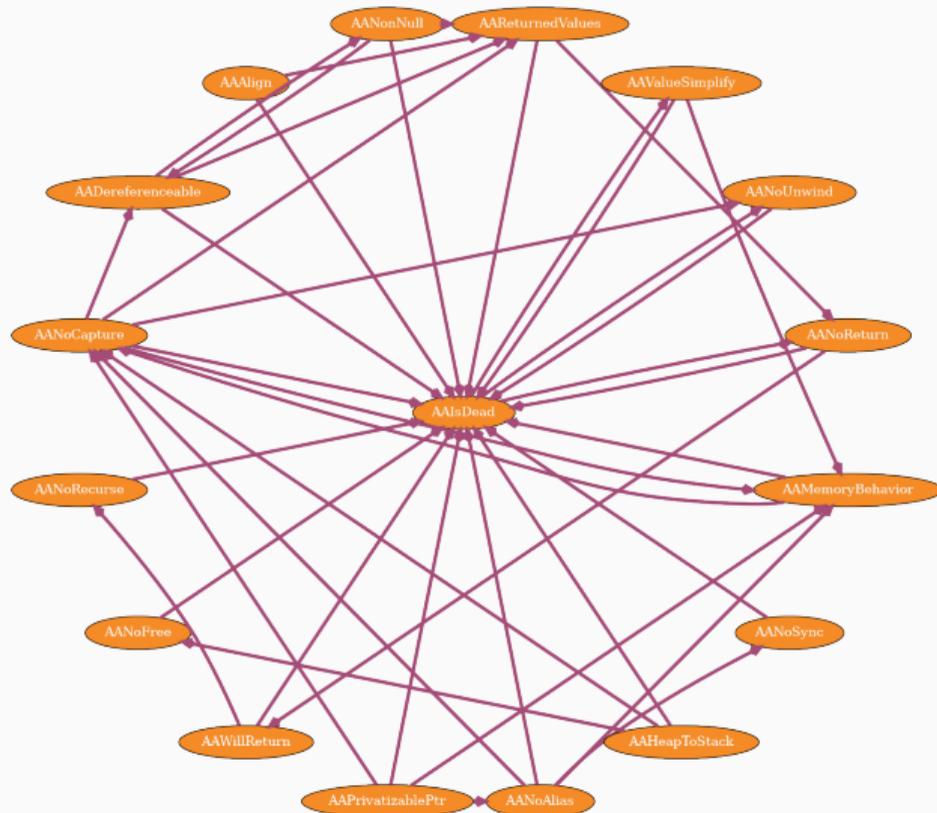
THE ATTRIBUTOR — WHY A FRAMEWORK?



THE ATTRIBUTOR — WHY A FRAMEWORK?



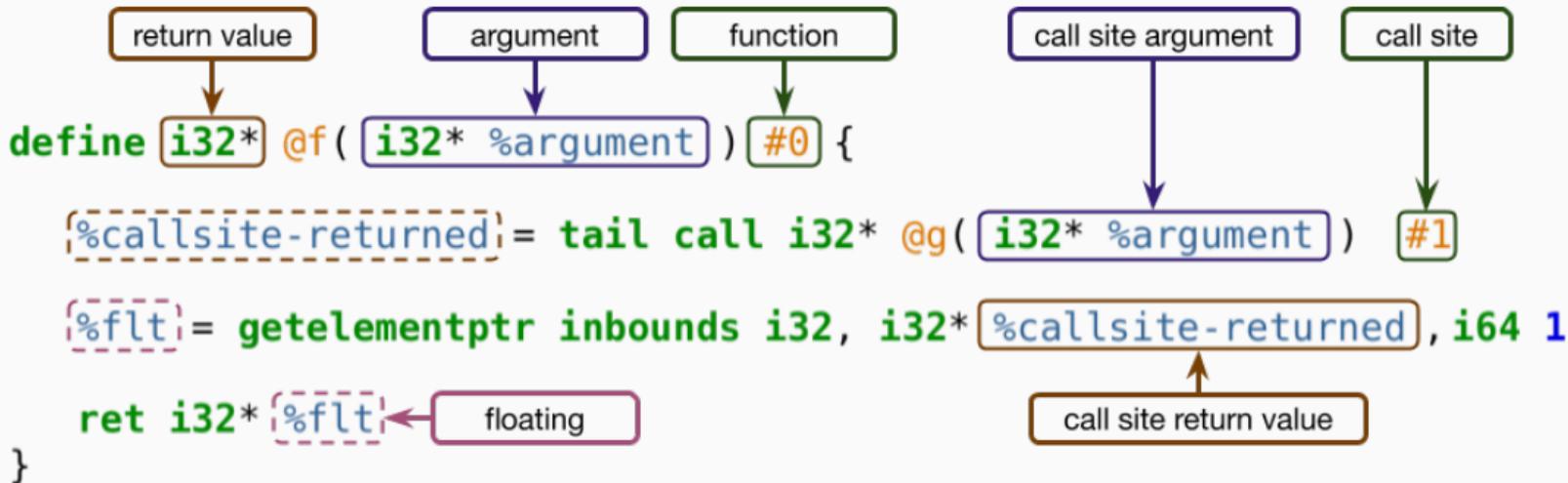
THE ATTRIBUTOR — WHY A FRAMEWORK?



III. DESIGN



LLVM-IR POSITIONS



```
ChangeStatus updateImpl(Attributor &A) override {
```

```
}
```



AAVALUESIMPLIFYRETURNED::UPDATEIMPL(ATTRIBUTOR &A)

```
ChangeStatus updateImpl(Attributor &A) override {  
    Optional<Value *> Before = getAssumedSimplifiedValue();
```

```
    Optional<Value *> After = getAssumedSimplifiedValue();  
    if (Before == After)  
        return ChangeStatus::UNCHANGED;  
    return ChangeStatus::CHANGED;  
}
```



AAVALUESIMPLIFYRETURNED::UPDATEIMPL(ATTRIBUTOR &A)

```
ChangeStatus updateImpl(Attributor &A) override {
    Optional<Value *> Before = getAssumedSimplifiedValue();

    auto Pred = [&](Instruction &I) {
};
    if (!A.checkForAllInstructions(Pred, this, {Instruction::Ret}))
        return indicatePessimisticFixpoint();

    Optional<Value *> After = getAssumedSimplifiedValue();
    if (Before == After)
        return ChangeStatus::UNCHANGED;
    return ChangeStatus::CHANGED;
}
```



AAVALUESIMPLIFYRETURNED::UPDATEIMPL(ATTRIBUTOR &A)

```
ChangeStatus updateImpl(Attributor &A) override {
    Optional<Value *> Before = getAssumedSimplifiedValue();

    auto Pred = [&](Instruction &I) {
        A.getAAFor<AAValueSimplify>(this, I.getOperand(0));
    };
    if (!A.checkForAllInstructions(Pred, this, {Instruction::Ret}))
        return indicatePessimisticFixpoint();

    Optional<Value *> After = getAssumedSimplifiedValue();
    if (Before == After)
        return ChangeStatus::UNCHANGED;
    return ChangeStatus::CHANGED;
}
```



AAVALUESIMPLIFYRETURNED::UPDATEIMPL(ATTRIBUTOR &A)

```
ChangeStatus updateImpl(Attributor &A) override {
    Optional<Value *> Before = getAssumedSimplifiedValue();

    auto Pred = [&](Instruction &I) {
        return combine(A.getAAFor<AAValueSimplify>(this, I.getOperand(0)));
    };
    if (!A.checkForAllInstructions(Pred, this, {Instruction::Ret}))
        return indicatePessimisticFixpoint();

    Optional<Value *> After = getAssumedSimplifiedValue();
    if (Before == After)
        return ChangeStatus::UNCHANGED;
    return ChangeStatus::CHANGED;
}
```



NEW ATTRIBUTES



nofree



`nosync`



`willreturn`



dereferenceable_globally



NON-ATTRIBUTE DEDUCTIONS



liveness



returned values



value simplify



heap-2-stack



pointer privatization





when to specialize for call sites
(\equiv “inlining + outlining”)



how to seed abstract attributes
(heuristics, pgo-based, ...)



reduce overheads



combine deduction schemes, e.g.,
context-based & def-use-based



...



EVALUATION — FUNCTIONATTRS (LATE) VS. ATTRIBUTOR (EARLY)

loc.	attribute	# w/o A.	# w/ A.	A. Δ	tot. w/o A.	tot. w/ A.
fn.	nosync	0	7612		0.0%	4.36%



EVALUATION — FUNCTIONATTRS (LATE) VS. ATTRIBUTOR (EARLY)

loc.	attribute	# w/o A.	# w/ A.	A. Δ	tot. w/o A.	tot. w/ A.
fn.	nosync	0	7612		0.0%	4.36%
arg.	dereferenceable	61825	66317	+7.27%	35.4%	38.0%



EVALUATION — FUNCTIONATTRS (LATE) VS. ATTRIBUTOR (EARLY)

loc.	attribute	# w/o A.	# w/ A.	A. Δ	tot. w/o A.	tot. w/ A.
fn.	nosync	0	7612		0.0%	4.36%
arg.	dereferenceable	61825	66317	+7.27%	35.4%	38.0%
fn.	nofree	5762	10188	+76.81%	3.3%	5.83%
fn.	willreturn	0	4146		0.0%	2.37%
arg.	writeonly	0	3562		0.0%	2.04%
arg.	readnone	5377	6040	+12.33%	3.08%	3.46%
fn.	noreturn	965	1611	+66.94%	0.553%	0.923%
arg.	align	419	900	+114.80%	0.24%	0.515%
ret.	dereferenceable	19041	19479	+2.30%	11.2%	11.4%
arg.	nocapture	28991	29413	+1.46%	16.6%	16.8%
arg.	readonly	14946	15281	+2.24%	8.56%	8.75%
arg.	returned	512	599	+16.99%	0.293%	0.343%
arg.	noalias	4098	4158	+1.46%	2.35%	2.38%
ret.	noalias	1150	1194	+3.83%	0.676%	0.701% ^{14/16}



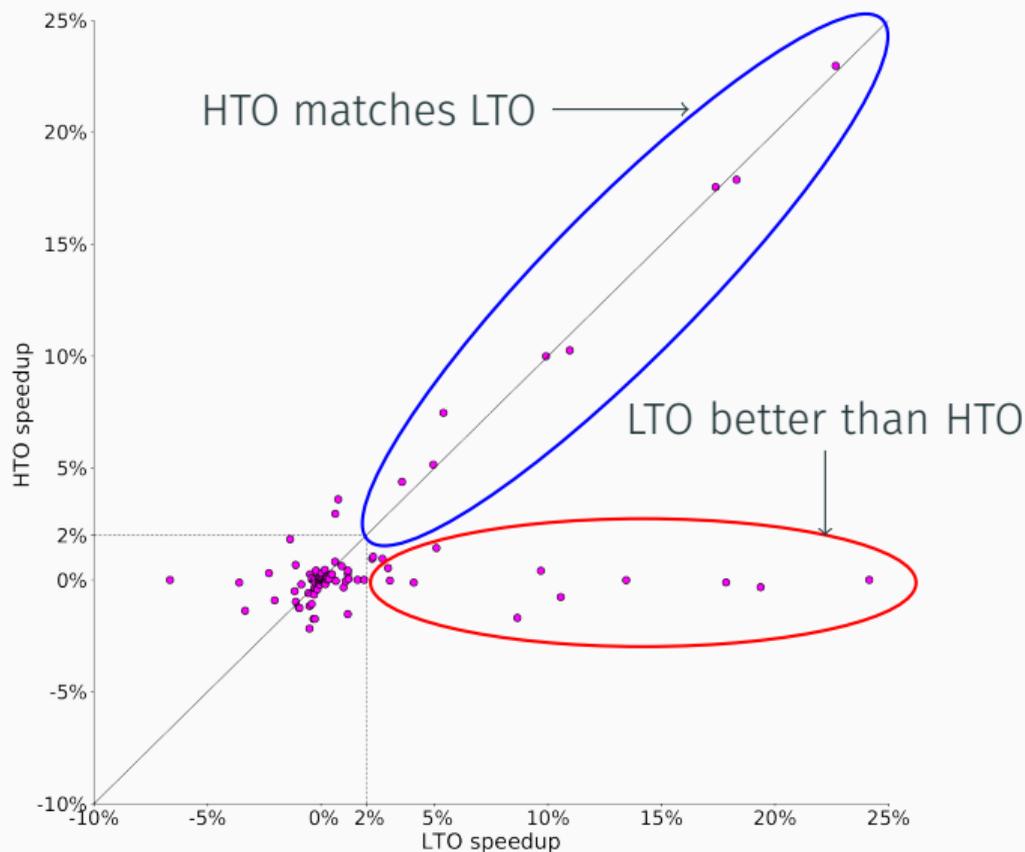
EVALUATION — FUNCTIONATTRS (LATE) VS. ATTRIBUTOR (EARLY)

loc.	attribute	# w/o A.	# w/ A.	A. Δ	tot. w/o A.	tot. w/ A.
fn.	nosync	0	7612		0.0%	4.36%
arg.	dereferenceable	61825	66317	+7.27%	35.4%	38.0%
fn.	nofree	5762	10188	+76.81%	3.3%	5.83%
fn.	willreturn	0	416		0.0%	2.37%
arg.	writable				0.0%	2.04%
arg.	readonly				3.08%	3.46%
fn.	noreturn	965	1611	+66.94%	0.553%	0.923%
arg.	align	419	900	+114.80%	0.24%	0.515%
ret.	dereferenceable	19041	19479	+2.30%	11.2%	11.4%
arg.	nocapture	28991	29413	+1.46%	16.6%	16.8%
arg.	readonly	14946	15281	+2.24%	8.56%	8.75%
arg.	returned	512	599	+16.99%	0.293%	0.343%
arg.	noalias	4098	4158	+1.46%	2.35%	2.38%
ret.	noalias	1150	1194	+3.83%	0.676%	0.701% ^{14/16}

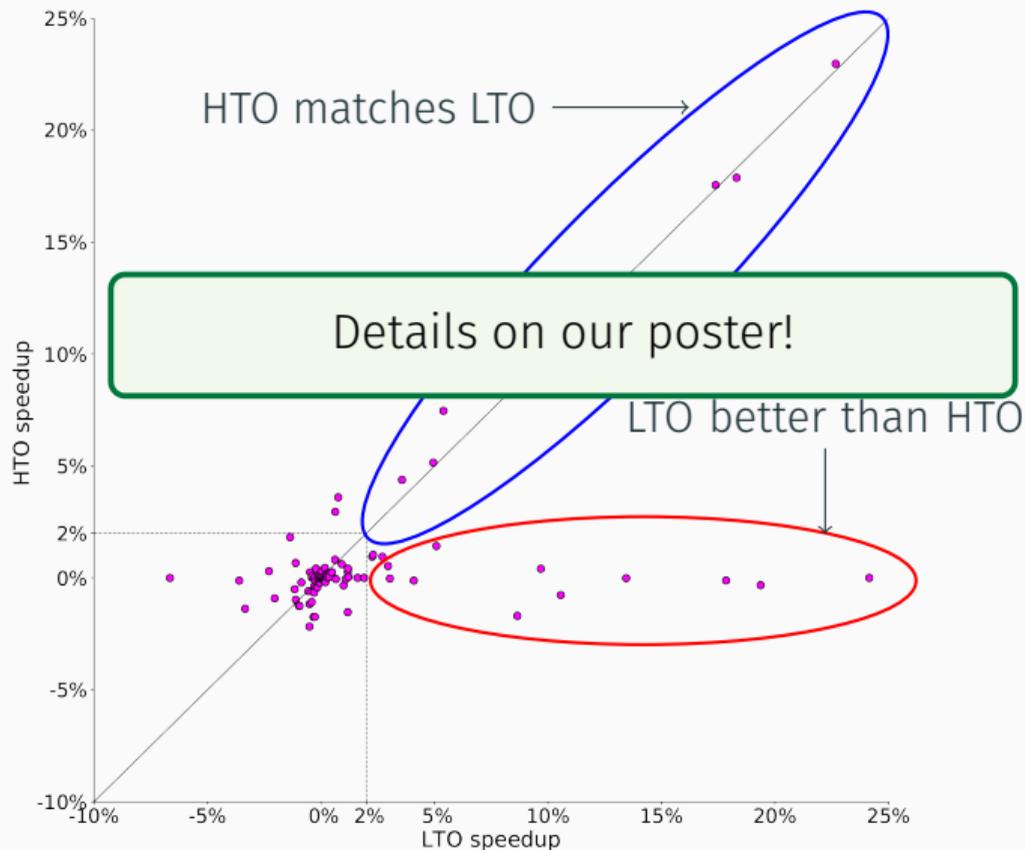
Details on our poster!



EVALUATION — (ATTRIBUTOR AIDED) “HEADER TIME OPTIMIZATION” (HTO)



EVALUATION — (ATTRIBUTOR AIDED) “HEADER TIME OPTIMIZATION” (HTO)



Tutorial: tomorrow 1:45pm - 2:55pm



Tutorial: tomorrow 1:45pm - 2:55pm

1) introduce a new `llvm::Attribute`



Tutorial: tomorrow 1:45pm - 2:55pm

- 1) introduce a new `llvm::Attribute`
- 2) derive the new `llvm::Attribute` with the *Attributor*



Tutorial: tomorrow 1:45pm - 2:55pm

- 1) introduce a new `llvm::Attribute`
- 2) derive the new `llvm::Attribute` with the *Attributor*
- 3) use the new `llvm::Attribute` to improve *alias analysis*



Tutorial: tomorrow 1:45pm - 2:55pm

Posters: tomorrow 4:00pm - 5:00pm



Tutorial: tomorrow 1:45pm - 2:55pm

Posters: tomorrow 4:00pm - 5:00pm

ATTRIBUTOR, A FRAMEWORK FOR INTER-PROCEDURAL INFORMATION DEDUCTION
Johannes Doerfler, Heiko Ueno, Stefan Sigler
Austrian National University of Applied Sciences, University of Innsbruck

Writing Optimizable Code in Rust

```
fn main() { let mut x = 0; for _ in 0..1000 { x = x + 1; } }
```

Automatically Making Code Optimizable

```
#[no_mangle] pub extern "C" fn main() { let mut x = 0; for _ in 0..1000 { x = x + 1; } }
```

Header Files

```
extern "C" { pub fn main(); }
```

Configuration	Time (s)	Size (KB)
Baseline	1.2	100
With Clang	1.1	95
With Clang + Clang	1.0	90
With Clang + Clang + Clang	0.9	85

'Header Time Optimization': Cross-Translation Unit Optimization via Annotated Headers
William S. Moses, Johannes Doerfler, Stefan Sigler
MIT CSAIL, Austrian National UoA

Writing Optimizable Code in Rust

```
fn main() { let mut x = 0; for _ in 0..1000 { x = x + 1; } }
```

Introducing Header Time Optimization

```
#[no_mangle] pub extern "C" fn main() { let mut x = 0; for _ in 0..1000 { x = x + 1; } }
```

Experiments

Figure 1: Scatter plot of Time vs. Size for different configurations. The plot shows a clear trend where smaller headers result in faster compilation times.

Figure 2: Bar chart showing the distribution of compilation times for different header sizes. The x-axis represents header size and the y-axis represents the number of compilations.

Acknowledgements & References

This work was supported by the Austrian Science Foundation FWF (SFB F4012-N13). We thank the reviewers for their helpful comments.



Tutorial: tomorrow 1:45pm - 2:55pm

Posters: tomorrow 4:00pm - 5:00pm

The image shows a screenshot of the ATTRIBUTOR framework paper with several handwritten annotations in yellow and orange. The paper title is "ATTRIBUTOR, A FRAMEWORK FOR INTER-PROCEDURAL INFORMATION DEDUCTION" by Johannes Doerfler, Heiko Ueno, Stefan Sigler, and Armin Heine. The annotations highlight various parts of the paper, including the abstract, introduction, and technical details. A green box is overlaid on the image with the text "Visit our posters and tutorial!".

Visit our posters and tutorial!

The image shows a poster titled "Header Time Optimization: Cross-Translation Unit Optimization via Annotated Headers" by William S. Moses (wsmoses@mit.edu) and Johannes Doerfler (jdoerfler@mit.edu). The poster is divided into several sections: Introduction, Experiments, and Acknowledgements & References. It includes code snippets, diagrams, and graphs. The poster discusses the challenges of header time optimization and presents a solution using annotated headers. The Experiments section shows a graph of compilation time vs. number of headers, and the Acknowledgements section lists the authors and their affiliations.





THE ATTRIBUTOR — EVALUATION — ASSUMING EXACT DEFINITIONS

loc.	attribute	# w/o A.	# w/ A.	A. Δ	tot. w/o A.	tot. w/ A.
fn.	nosync	0	78491		0.0%	45.90%
arg.	dereferenceable	59578	64214	+7.78%	34.8%	37.50%
fn.	nofree	25649	76719	+199.11%	15.0%	44.90%
fn.	willreturn	0	64748		0.0%	37.90%
arg.	writeonly	0	4229		0.0%	2.47%
arg.	readnone	40505	38414	-5.16%	23.7%	22.50%
fn.	noreturn	879	2394	+172.36%	0.514%	1.40%
arg.	align	449	1028	+128.95%	0.263%	0.60%
ret.	dereferenceable	18064	19419	+7.50%	10.8%	11.60%
arg.	nocapture	153523	155294	+1.15%	89.8%	90.80%
arg.	returned	9418	13937	+47.98%	5.51%	8.15%
arg.	noalias	4113	4189	+1.85%	2.41%	2.45%
ret.	noalias	3015	3310	+9.78%	1.81%	1.98%
fn.	writeonly	8089	9877	+22.10%	4.73%	5.78%
fn.	nounwind	123516	125480	+1.59%	72.2%	73.40%





MUST-BE-EXECUTED-CONTEXT

```
define internal void @f( [i32* %P], i1 %cmp ) {
```

```
  [store i32 1, i32* %P] [dereferenceable(4) %P]  
  br i1 %cmp, label %then, label %else
```

then:

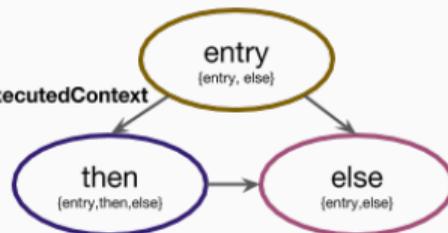
```
  call void @g( [i32* %P] )  
  call void @g( [i32* dereferenceable(8) %P] )  
  br label %else
```

else:

```
  call void @g( [i32* %P] )  
  ret void
```

```
}
```

MustBeExecutedContext



Always comes back to the callsite !!

```
declare void @g(i32*) [willreturn nounwind]
```



INLINING VS. IPO



The “*inline-first*” approach:

- I: aggressive inlining, e.g., all N call sites
- II: perform intra-procedural analyses + transformations (N times)
- III: derive information + transformation opportunities inter-procedurally



INLINING VS. IPO

The “*inline-first*” approach:

- I: aggressive inlining, e.g., all N call sites
- II: perform intra-procedural analyses + transformations (N times)
- III: derive information + transformation opportunities inter-procedurally

The “*IPO-first*” approach:

- I: derive information + transformation opportunities inter-procedurally
- II: internalize & specialize functions if necessary & beneficial
- III: inline where benefit can be expected

