# LLVM IR: Past, Present and Future

Nikita Popov

EuroLLVM 2025

Red Hat

# LLVM 1.0

```
implementation

int "testfunction"(long %i0, long %j0)
begin
   %array0 = malloc [4 x ubyte]
   %size   = add uint 2, 2
   %array1 = malloc ubyte, uint 4
   %array2 = malloc ubyte, uint %size

   %idx = getelementptr [4 x ubyte]* %array0, long 0, long 2
   store ubyte 123, ubyte* %idx
   free [4 x ubyte]* %array0
   free ubyte* %array1
   free ubyte* %array2
   ; ...
   ret int 3
end
```

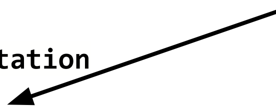
https://github.com/llvm/llvm-project/blob/release/1.0.x/llvm/test/Feature/testmemory.ll



Red Hat

# LLVM 1.0

Pascal-style syntax



```
implementation

int "testfunction"(long %i0, long %j0)
begin
   %array0 = malloc [4 x ubyte]
   %size   = add uint 2, 2
   %array1 = malloc ubyte, uint 4
   %array2 = malloc ubyte, uint %size

   %idx = getelementptr [4 x ubyte]* %array0, long 0, long 2
   store ubyte 123, ubyte* %idx
   free [4 x ubyte]* %array0
   free ubyte* %array1
   free ubyte* %array2
   ; ...
   ret int 3
end
```

https://github.com/llvm/llvm-project/blob/release/1.0.x/llvm/test/Feature/testmemory.ll



Red Hat

# LLVM 1.0

Pascal-style syntax

C-style integer types

```
implementation

int "testfunction"(long %i0, long %j0)
begin
   %array0 = malloc [4 x ubyte]
   %size   = add uint 2, 2
   %array1 = malloc ubyte, uint 4
   %array2 = malloc ubyte, uint %size

   %idx = getelementptr [4 x ubyte]* %array0, long 0, long 2
   store ubyte 123, ubyte* %idx
   free [4 x ubyte]* %array0
   free ubyte* %array1
   free ubyte* %array2
   ; ...
   ret int 3
end
```

https://github.com/llvm/llvm-project/blob/release/1.0.x/llvm/test/Feature/testmemory.ll



Red Hat

# LLVM 1.0

Pascal-style syntax

C-style integer types

First-class malloc/free

```
implementation

int "testfunction"(long %i0, long %j0)
begin
   %array0 = malloc [4 x ubyte]
   %size   = add uint 2, 2
   %array1 = malloc ubyte, uint 4
   %array2 = malloc ubyte, uint %size

   %idx = getelementptr [4 x ubyte]* %array0, long 0, long 2
   store ubyte 123, ubyte* %idx
   free [4 x ubyte]* %array0
   free ubyte* %array1
   free ubyte* %array2
   ; ...
   ret int 3
end
```

https://github.com/llvm/llvm-project/blob/release/1.0.x/llvm/test/Feature/testmemory.ll

# LLVM 2.0

Sign-less integers

```
define i32 @testfunction(i64 %i0, i64 %j0)
{
   %array0 = malloc [4 x i8]
   %size   = add i32 2, 2
   %array1 = malloc i8, i32 4
   %array2 = malloc i8, i32 %size

   %idx = getelementptr [4 x i8]* %array0, i64 0, i64 2
   store i8 123, i8* %idx
   free [4 x i8]* %array0
   free i8* %array1
   free i8* %array2
   ; ...
   ret i32 3
}
```



Red Hat

# LLVM 2.7

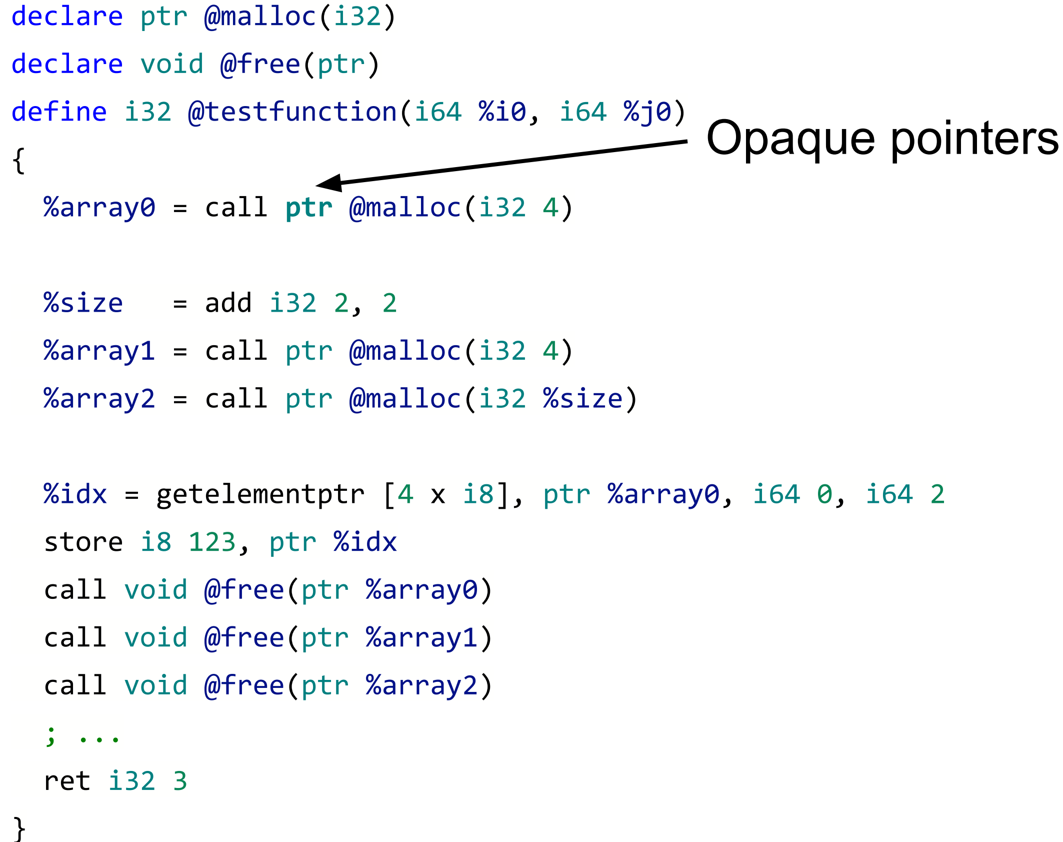
```
declare i8* @malloc(i32)
declare void @free(i8*)
define i32 @testfunction(i64 %i0, i64 %j0)
{
  %array0 = call i8* @malloc(i32 4)
  %array0.cast = bitcast i8* %array0 to [4 x i8]*
  %size   = add i32 2, 2
  %array1 = call i8* @malloc(i32 4)
  %array2 = call i8* @malloc(i32 %size)

  %idx = getelementptr [4 x i8]* %array0.cast, i64 0, i64 2
  store i8 123, i8* %idx
  call void @free(i8* %array0)
  call void @free(i8* %array1)
  call void @free(i8* %array2)
  ; ...
  ret i32 3
}
```

No first class malloc/free



Red Hat

# LLVM 15




```
declare ptr @malloc(i32)
declare void @free(ptr)
define i32 @testfunction(i64 %i0, i64 %j0)
{
  %array0 = call ptr @malloc(i32 4)

  %size   = add i32 2, 2
  %array1 = call ptr @malloc(i32 4)
  %array2 = call ptr @malloc(i32 %size)

  %idx = getelementptr [4 x i8], ptr %array0, i64 0, i64 2
  store i8 123, ptr %idx
  call void @free(ptr %array0)
  call void @free(ptr %array1)
  call void @free(ptr %array2)
  ; ...
  ret i32 3
}
```



Red Hat

# De-type-ification: Remove redundant type information



Red Hat

# De-type-ification: Remove redundant type information

Sign-less integers: `int`, `uint` -> `i32`

Opaque pointers: `i32*`, `%struct*` -> `ptr`



Red Hat

# De-type-ification: Remove redundant type information

Sign-less integers: `int, uint -> i32`

Opaque pointers: `i32*, %struct* -> ptr`

- Avoid unnecessary bitcasts
- Encourages more general optimizations
  - For example: Better redundancy elimination (CSE/GVN)
- Makes invalid assumptions impossible



Red Hat

# De-type-ification: ptradd

```
%gep = getelementptr {i32, i8}, ptr %ptr, i64 0, i32 1
%gep = getelementptr [2 x i32], ptr %ptr, i64 0, i32 1
%gep = getelementptr i16, ptr %ptr, i64 2
```

RFC: https://discourse.llvm.org/t/rfc-replacing-getelementptr-with-ptradd/68699



Red Hat

# De-type-ification: ptradd

```
%gep = getelementptr {i32, i8}, ptr %ptr, i64 0, i32 1
%gep = getelementptr [2 x i32], ptr %ptr, i64 0, i32 1
%gep = getelementptr i16, ptr %ptr, i64 2
```

```
%gep = getelementptr i8, ptr %ptr, i64 4
```

RFC: https://discourse.llvm.org/t/rfc-replacing-getelementptr-with-ptradd/68699



Red Hat

# De-type-ification: ptradd

```
%gep = getelementptr {i32, i8}, ptr %ptr, i64 0, i32 1
%gep = getelementptr [2 x i32], ptr %ptr, i64 0, i32 1
%gep = getelementptr i16, ptr %ptr, i64 2
```

```
%gep = getelementptr i8, ptr %ptr, i64 4
```

```
%gep = ptradd ptr %ptr, i64 4
```

RFC: https://discourse.llvm.org/t/rfc-replacing-getelementptr-with-ptradd/68699



Red Hat

# De-type-ification: ptradd

```
%gep = getelementptr {i32, i8}, ptr %ptr, i64 0, i32 1
%gep = getelementptr [2 x i32], ptr %ptr, i64 0, i32 1
%gep = getelementptr i16, ptr %ptr, i64 2
```

```
%gep = getelementptr i8, ptr %ptr, i64 4
```

← Present

```
%gep = ptradd ptr %ptr, i64 4
```

RFC: https://discourse.llvm.org/t/rfc-replacing-getelementptr-with-ptradd/68699



Red Hat

# De-type-ification: ptradd

```
%gep = getelementptr i32, ptr %ptr, i64 %n
```

RFC: https://discourse.llvm.org/t/rfc-replacing-getelementptr-with-ptradd/68699



Red Hat

# De-type-ification: ptradd

```
%gep = getelementptr i32, ptr %ptr, i64 %n
```

```
%scaled = shl i32 %n, 2
%gep = ptradd ptr %ptr, i64 %scaled
```

RFC: https://discourse.llvm.org/t/rfc-replacing-getelementptr-with-ptradd/68699



Red Hat

# De-type-ification: ptradd

```
%gep = getelementptr i32, ptr %ptr, i64 %n
```

```
%scaled = shl i32 %n, 2
%gep = ptradd ptr %ptr, i64 %scaled
```

or

```
%gep = ptradd ptr %ptr, i64 4 * %n
```

RFC: https://discourse.llvm.org/t/rfc-replacing-getelementptr-with-ptradd/68699

# De-type-ification: alloca, byval, etc

```
%a = alloca { i32, i64 }
```

```
%a = alloca 16, align 8
```



Red Hat

# Instruction flags

- LLVM 18: or disjoint
- LLVM 18: zext nneg
- LLVM 19: uitofp nneg
- LLVM 19: trunc nuw/nsw
- LLVM 19: getelementptr nuw
- LLVM 20: icmp samesign

# Instruction flags: Undo canonicalization

- or disjoint -> add
- zext nneg -> sext
- uitofp nneg -> sitofp
- icmp samesign ult -> icmp slt

# Instruction flags: Undo canonicalization

```
%y = shl i32 %x, 2
%z = add i32 %y, 1
```

↓ Canonicalize

```
%y = shl i32 %x, 2
%z = or disjoint i32 %y, 1
```

↓ Undo

```
lea eax, [4*rdi + 1]
```



Red Hat

# Instruction flags

- Undo canonicalization
- Manifest analysis results
  - Example: IPSCCP infers inter-procedurally, InstCombine uses locally
- Convey frontend guarantees



Red Hat

# Manifesting constraints and analysis results

- Attributes, metadata
  - Precise
  - Only at call/load boundaries
  - Often get lost (SROA, inlining)



Red Hat

# Manifesting constraints and analysis results

- Attributes, metadata
  - Precise
  - Only at call/load boundaries
  - Often get lost (SROA, inlining)
- Flags:
  - Imprecise
  - Only certain instructions



Red Hat

# Manifesting constraints and analysis results

- Attributes, metadata
  - Precise
  - Only at call/load boundaries
  - Often get lost (SROA, inlining)
- Flags:
  - Imprecise
  - Only certain instructions
- Assumes:
  - Precise
  - Undefined behavior rather than poison semantics
  - Often negative optimization impact

Speculative

# At-use flags?

```
%ext = zext nneg i32 %x to i64
```

```
%ext = zext i32 nneg %x to i64
```



Red Hat

# At-use flags?

```
%ext = zext nneg i32 %x to i64
```

```
%ext = zext i32 nneg %x to i64
```

```
%fti = uitofp i32 nneg %x to float
%cmp = icmp i32 nneg %x, nneg %y
%shr = lshr i32 nneg %x, %shamt
```

# At-use flags?

- Generalization: At-use range attribute

# At-use flags?

- Generalization: At-use range attribute
- Problem: Miscompiles due to missing flag invalidation
  - Prefer creating new instructions over in-place modification



Red Hat

Speculative

# At-use flags?

- Generalization: At-use range attribute
- Problem: Miscompiles due to missing flag invalidation
  - Prefer creating new instructions over in-place modification
- Problem: Memory overhead of per-use information
  - ConstantRange is huge
  - Number of leading zeros/sign-bits probably best memory/usefulness tradeoff



Red Hat

# Attributes

- LLVM 16: memory
- LLVM 17: nofpclass
- LLVM 18: writable, dead_on_unwind
- LLVM 19: range
- LLVM 19: initializes
- LLVM 21: captures


Red Hat

# captures(...) attribute

```
captures(none)                  ; == nocapture
captures(address, provenance)   ; == no attribute

captures(address)
captures(ret: address, provenance)
```

RFC: https://discourse.llvm.org/t/rfc-improvements-to-capture-tracking/81420



Red Hat

# captures(...) attribute

```
captures(none)                  ; == nocapture
captures(address, provenance)   ; == no attribute

captures(address)
captures(ret: address, provenance)
```

- address: Information about integral value of pointer
- provenance: Permission to perform memory accesses through pointer

RFC: https://discourse.llvm.org/t/rfc-improvements-to-capture-tracking/81420



Red Hat

# captures(...) attribute

```
captures(none)                     ; == nocapture
captures(address, provenance)      ; == no attribute

captures(address)
captures(ret: address, provenance)
```

- address: Information about integral value of pointer
- provenance: Permission to perform memory accesses through pointer
- Alias analysis only cares about provenance
  - Pointer icmps no longer interfere with alias analysis

RFC: https://discourse.llvm.org/t/rfc-improvements-to-capture-tracking/81420



Red Hat

# ptrtoint

- ptrtoint exposes provenance, can be recovered via inttoptr
- Technically a side-effect, cannot DCE – but we do it anyway

Related RFC: https://discourse.llvm.org/t/clarifiying-the-semantics-of-ptrtoint/83987

# ptrtoint

- ptrtoint exposes provenance, can be recovered via inttoptr
- Technically a side-effect, cannot DCE – but we do it anyway
- Need: ptrtoint that does not expose provenance
  - Could be: ptrtoint flag
  - Could be: Separate ptrtoaddr instruction

Related RFC: https://discourse.llvm.org/t/clarifiying-the-semantics-of-ptrtoint/83987



Red Hat

# ptrsub

```
%a.int = ptrtoint ptr %a to i64   ; captures(address, provenance)
%b.int = ptrtoint ptr %b to i64
%sub = sub i64 %a.int, %b.int
```



Red Hat

# ptrsub

```
%a.int = ptrtoint ptr %a to i64    ; captures(address, provenance)
%b.int = ptrtoint ptr %b to i64
%sub = sub i64 %a.int, %b.int
```

```
%sub = ptrsub ptr %a, %b           ; captures(address)
```



Red Hat

# ptrsub

```
%a.int = ptrtoint ptr %a to i64   ; captures(address, provenance)
%b.int = ptrtoint ptr %b to i64
%sub = sub i64 %a.int, %b.int
```

```
%sub = ptrsub ptr %a, %b          ; captures(address)
%sub = ptrsub nuw ptr %a, %b
%sub = ptrsub inbounds ptr %a, %b
```



Red Hat

# Summary



Red Hat

# Summary

This presentation:

- De-type-ification
- Constraint and analysis manifestation
- Reducing pointer provenance exposure

# Summary

This presentation:

- De-type-ification
- Constraint and analysis manifestation
- Reducing pointer provenance exposure

Other directions:

- Improving debuginfo representation/quality
- Improving floating point environment support (constrained FP)
- [...]



Red Hat

# Thank You!

Questions?