Type Based Iteration

Stephan Roslen

Motivation

Recursion?

```cpp
template <typename T>
decltype(auto) deref(T &ptr) {
  if constexpr(std::is_pointer_v<T>) {
    return deref(*ptr);
  } else {
    return ptr;
  }
}
```

Insights

```
…
#ifdef INSIGHTS_USE_TEMPLATE
template<>
long & deref<long *>(long *& ptr)
{
  if constexpr(std::is_pointer_v<long *>) {
    return deref(*ptr);
  }

}
#endif


#ifdef INSIGHTS_USE_TEMPLATE
template<>
long & deref<long>(long & ptr)
{
  if constexpr(std::is_pointer_v<long>) ;
  else /* constexpr */ {
    return ptr;
  }

}
#endif
```

Disassembly O0

```
decltype(auto) deref<long>(long&): # @decltype(auto) deref<long>(long&)
  push    rbp
  mov     rbp, rsp
  mov     qword ptr [rbp - 8], rdi
  mov     rax, qword ptr [rbp - 8]
  pop     rbp
  ret
decltype(auto) deref<long*>(long*&): # @decltype(auto) deref<long*>(long*&)
  push    rbp
  mov     rbp, rsp
  sub     rsp, 16
  mov     qword ptr [rbp - 8], rdi
  mov     rax, qword ptr [rbp - 8]
  mov     rdi, qword ptr [rax]
  call    decltype(auto) deref<long>(long&)
  add     rsp, 16
  pop     rbp
  ret
decltype(auto) deref<long**>(long**&): # @decltype(auto) deref<long**>(long**&)
 …
```

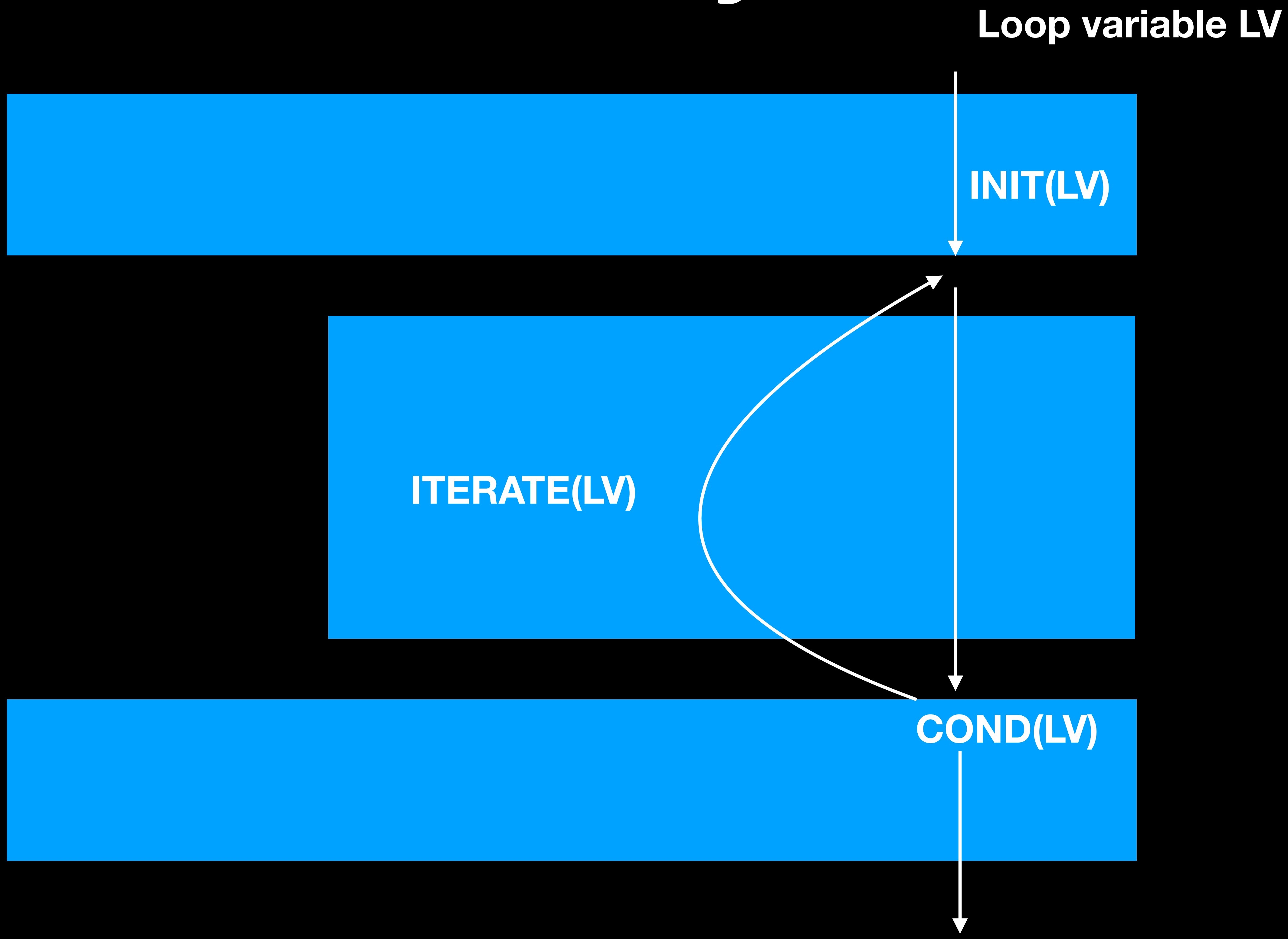
Disassembly O1,OS,O3

```cpp
decltype(auto) test(long ***********v) {
    return deref(v);
}
```

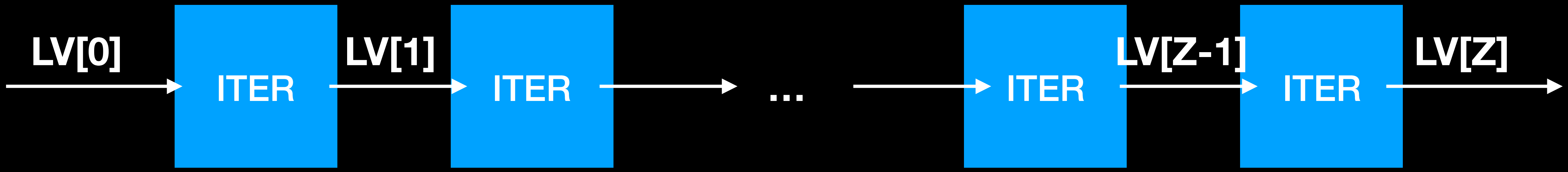
```asm
test(long***********): # @test(long***********)
  mov rax, qword ptr [rdi]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  ret
```

Idea

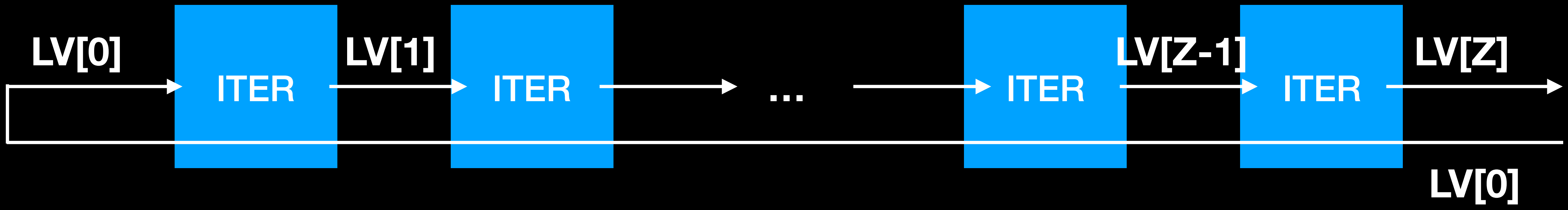
Theory



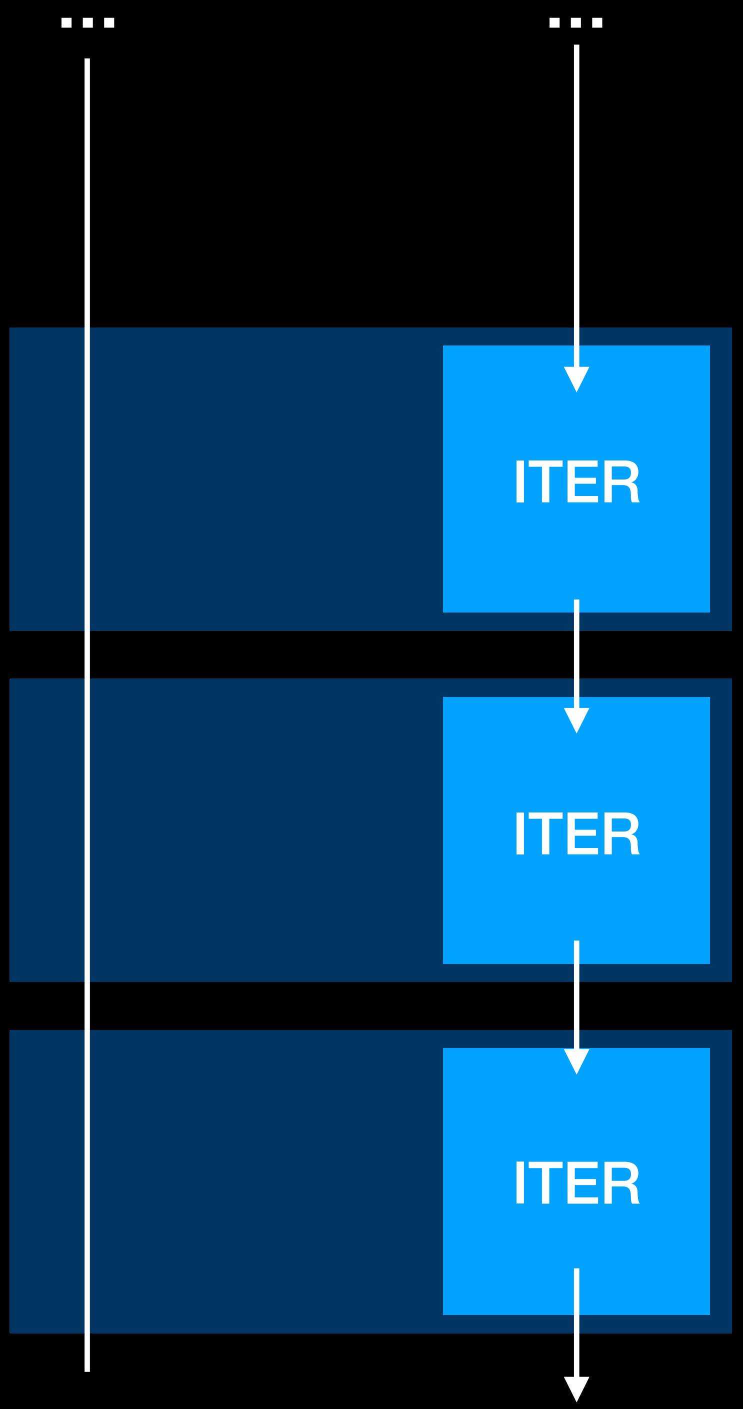
Theory



Theory



Theory



```
decltype(LV[Z-2]) Iter_(decltype(LV[0]))
decltype(LV[Z-1]) Iter_(decltype(LV[0]))
 decltype(LV[Z]) Iter_(decltype(LV[0]))
```

TMP Basics

Typelist

```cpp
template <typename ... Ts>
struct typelist;

template <typename T1, typename T2>
struct concat;

template <typename ... Ts1, typename ... Ts2>
struct concat<typelist<Ts1 ... >, typelist<Ts2 ... >> {
  using type = typelist<Ts1 ... , Ts2 ... >;
};

template <typename T1, typename T2>
using concat_t = typename concat<T1, T2>::type;

template <typename T>
struct headtail;

template <typename THead, typename ... TTail>
struct headtail<typelist<THead, TTail ... >> {
  using head = THead;
  using tail = typelist<TTail ... >;
};

template <typename T>
using head_t = typename headtail<T>::head;

template <typename T>
using tail_t = typename headtail<T>::tail;
```

Successor

```cpp
template <typename Tlist, typename T>
struct successor;

template <typename Tlist, typename T>
using successor_t = typename successor<Tlist, T>::type;

template <typename T1, typename T2, typename ...Ts>
struct successor<typelist<T1, T2, Ts...>, T1> {
  using type = T2;
};

template <typename T, typename ...Ts>
struct successor<typelist<Ts...>, T> {
  using type = successor_t<tail_t<typelist<Ts...>>, T>;
};
```

Implementation

Type Based Loop

```
struct CondTraitPtr {
  template <typename T>
  static constexpr bool value = std::is_pointer_v<std::remove_reference_t<T>>;
};

template <typename T>
decltype(auto) deref(T &ptr) {
  return TypeBasedLoop(
      ptr,
      CondTraitPtr{},
      [](const auto &input) -> auto & {
        return *input;
      });
}
```

Type Based Loop

```cpp
template <typename Initial_, typename WhileCondTrait, typename Morphism>
auto TypeBasedLoop(
    Initial_ &&initial,
    WhileCondTrait &&,
    Morphism &&morphism) -> result_t<
        std::remove_reference_t<Initial_>,
        WhileCondTrait,
        Morphism>;
```

```cpp
template <typename Initial, typename WhileCondTrait, typename Morphism>
using result_t =
    typelist::head_t<generate_typecascade_t<Initial, WhileCondTrait, Morphism>>;
```

```cpp
  /* First instantiated from: insights.cpp:70 */
#ifdef INSIGHTS_USE_TEMPLATE
template<>
struct generate_typecascade<long ****&, CondTraitPtr, __lambda_138_7, void>
{
  using type = typelist::concat_t<
      typelist::typelist<long &, long *&, long **&, long ***&>,
      typelist::typelist<long ****&>>;
};

#endif
```

Type Based Loop

```cpp
template <
    typename WhileCondTrait,
    typename Initial,
    typename Morphism,
    typename SFINAE = void>
struct generate_typecascade;
```

```cpp
  /* First instantiated from: insights.cpp:70 */
#ifdef INSIGHTS_USE_TEMPLATE
template<>
struct generate_typecascade<long ****&, CondTraitPtr, __lambda_138_7, void>
{
  using type = typelist::concat_t<
      typelist::typelist<long &, long *&, long **&, long ***&>,
      typelist::typelist<long ****&>>;
};

#endif
```

Type Based Loop

```cpp
template <typename Initial, typename WhileCondTrait, typename Morphism>
struct generate_typecascade<
    Initial,
    WhileCondTrait,
    Morphism,
    std::enable_if_t<!WhileCondTrait::template value<Initial>>> {
  using type = typelist::typelist<Initial>;
};
```

```cpp
/* First instantiated from: insights.cpp:70 */
#ifdef INSIGHTS_USE_TEMPLATE
template<>
struct generate_typecascade<long &, CondTraitPtr, __lambda_138_7, void>
{
  using type = typelist::typelist<long &>;
};

#endif
```

Type Based Loop

```
template <typename Initial, typename WhileCondTrait, typename Morphism>
struct generate_typecascade<
    Initial,
    WhileCondTrait,
    Morphism,
    std::enable_if_t<WhileCondTrait::template value<Initial>>> {
  using type = typelist::concat_t<
      generate_typecascade_t<
          application_result_t<Morphism, Initial>,
          WhileCondTrait,
          Morphism>,
      typelist::typelist<Initial>>;
};
```

```
  /* First instantiated from: insights.cpp:70 */
#ifdef INSIGHTS_USE_TEMPLATE
template<>
struct generate_typecascade<long ****&, CondTraitPtr, __lambda_138_7, void>
{
  using type = typelist::concat_t<
      typelist::typelist<long &, long *&, long **&, long ***&>,
      typelist::typelist<long ****&>>;
};

#endif
```

Type Based Loop

```
template <typename Initial_, typename WhileCondTrait, typename Morphism>
auto TypeBasedLoop(
    Initial_ &&initial,
    WhileCondTrait &&,
    Morphism &&morphism) -> result_t<
        std::remove_reference_t<Initial_>,
        WhileCondTrait,
        Morphism> {
  using Initial = std::decay_t<Initial_>;
  using Cascade = generate_typecascade_t<Initial, WhileCondTrait, Morphism>;
  using Result = typelist::head_t<Cascade>;

  return GetTypeIterationResult<Result, Initial, Cascade, Morphism>(
      initial,
      morphism);
}
```

Type Based Loop

```
template <typename To, typename Initial, typename Cascade, typename Morphism>
To GetTypeIterationResult(Initial &initial, Morphism &morphism) {
  if constexpr (std::is_same_v<To, Initial>) {
    return initial;
  } else {
    using Next = typelist::successor_t<Cascade, To>;
    return morphism(
        GetTypeIterationResult<Next, Initial, Cascade, Morphism>(
            initial,
            morphism));
  }
}
```

Type Based Loop

```cpp
struct CondTraitPtr {
  template <typename T>
  static constexpr bool value = std::is_pointer_v<std::remove_reference_t<T>>;
};

template <typename T>
decltype(auto) deref(T &ptr) {
  return TypeBasedLoop(
      ptr,
      CondTraitPtr{},
      [](const auto &input) -> auto & {
        return *input;
      });
}
```

Disassembly O1,OS,O3

```cpp
decltype(auto) test(long ***********v) {
    return deref(v);
}
```

```asm
test(long***********): # @test(long***********)
  mov rax, qword ptr [rdi]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  mov rax, qword ptr [rax]
  ret
```

Other algorithms?

```
template <unsigned int Iter_, unsigned int Sum_>
struct LoopSumState {
  static constexpr unsigned int Iter = Iter_;
  static constexpr unsigned int Sum = Sum_;
  std::string runtime_member;
};
```

```
template <unsigned int N>
struct CondTraitLoopCountLess {
  template <typename T>
  static constexpr bool value = T::Iter < N;
};
```

```
template <typename N>
std::tuple <int, std::string> somealgo(const N &) {
  using LoopSumStateStart = LoopSumState<0, 0>;
  auto res = TypeBasedLoop(
      LoopSumStateStart{"Golgafrinchan Arch Fleet Ship B"},
      CondTraitLoopCountLess<N::value>{},
      [](const auto &v) → auto {
          using type = std::decay_t<decltype(v)>;
          return LoopSumState<type::Iter + 1, type::Sum + type::Iter>{
              do_something(v)};
      });
  return {res.Sum, res.runtime_member};
}
```

Questions?

Thank you!

Bonus slide: True Iteration*

```cpp
template <typename T>
decltype(auto) deref(T &ptr) {
  using ptr_t = std::remove_reference_t<T>;
  if constexpr (0 == ptr_cnt_v<ptr_t>) {
    return ptr;
  } else {
    using end_t = ptr_dereftype_t<ptr_t>;
    end_t **v = std::launder(reinterpret_cast<end_t **>(ptr));
    for (size_t i{0}; i < ptr_cnt_v<ptr_t> - 1; ++i) {
      v = std::launder(reinterpret_cast<end_t **>(*v));
    }
    return *std::launder(reinterpret_cast<end_t *>(v));
  }
}
```

***don't do this, it's just a joke**

Bonus slide: Disassembly True Iteration

```cpp
decltype(auto) test(long ***********v) {
    return deref(v);
}
```

```
test(long***********): # @test(long***********)
  mov     rax, rdi
  mov     ecx, 10
.LBB0_1: # ⇒This Inner Loop Header: Depth=1
  mov     rax, qword ptr [rax]
  dec     rcx
  jne     .LBB0_1
  ret
```