daw::string_view

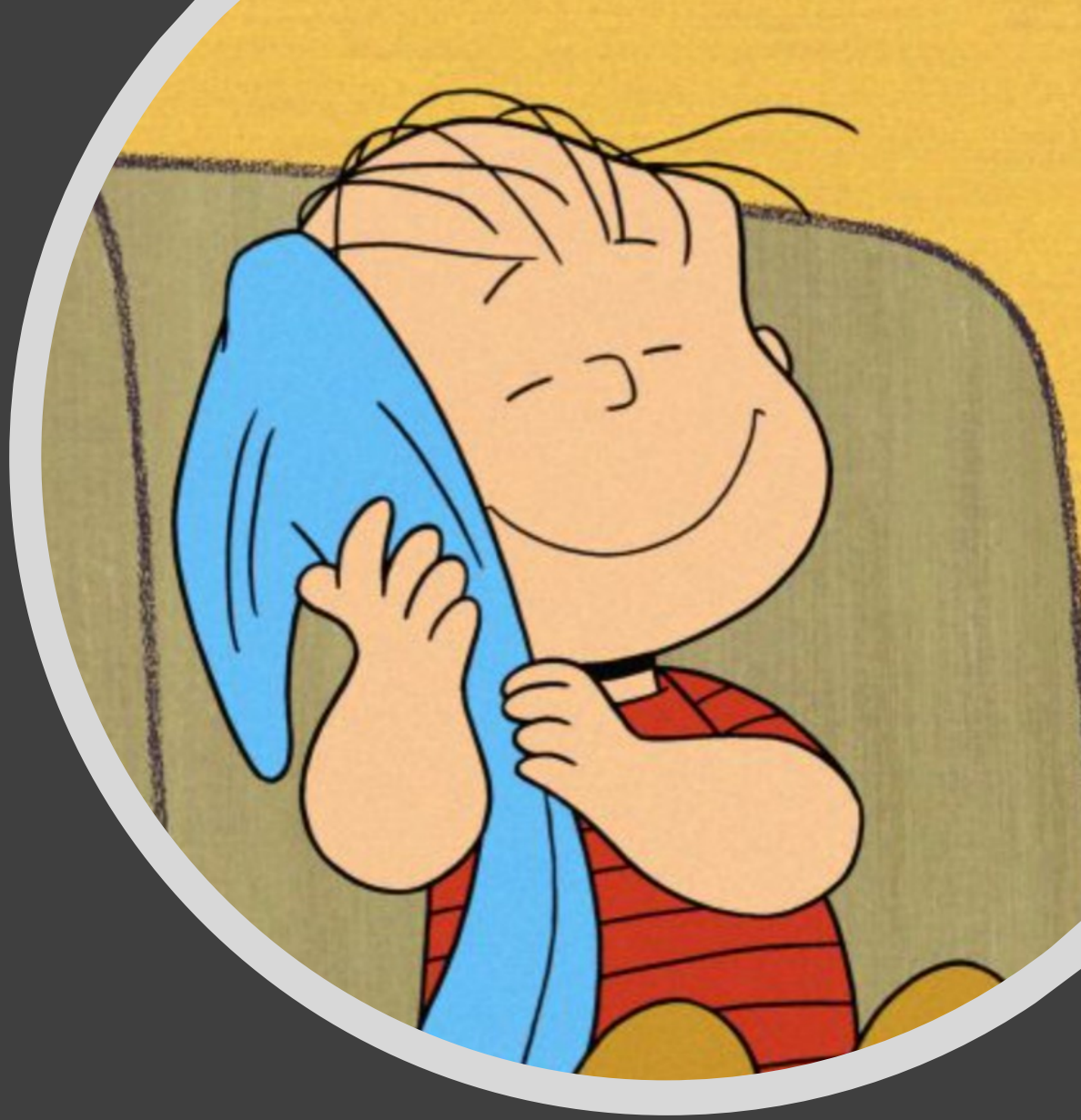
Making string processing explicit and safer

Danger Will Robinson

- std::string_view has several areas that can easily lead to UB and security issues.
- remove_prefix/suffix require that the parameter be <= size( ). This forces code to clip the value and makes find/remove_prefix code more verbose and more likely to result in UB
- Char const * constructor needs pre checking for nullptr at runtime. This makes using null returning API’s more likely to result in UB

Safer by default

- Most routines will clip to bounds, can opt into less safe versions
- Nullptr is defined as an empty range, not UB
- Less code. Code around find/substr/remove_prefix is a source of off by one errors



A few good routines

- Adding find_if as a member to provide balance in the force
- Hot take: nullptr forms the range [nullptr, nullptr + 0)
  - constructor forms an empty string_view when passed null
- remove_prefix( ) removes 1.
- Combining find/find_if and remove_prefix/substr leads to super powers
  - string_view string_view::pop_front_until(…) is the basis of a cleaner parsing. This includes the back variant too.
  - Argument types include char/string_view/unary predicates
  - Returns the skipped over part, and by default discards the delimiter.
  - try_pop_front_until is similar but returns an empty string_view and leaves the object alone if not found
  - pop_front_while similarly returns the substring formed while condition is true
- These routines lead to much less complex code

Split shouldn’t be like the Kumite

- npos and remove_prefix/substr do not mix. When they do, that’s a UB
- Easy off by one errors

```
std::string_view sv = "This is a test  of the test system   ";
while( not sv.empty( ) ) {
    auto pos = sv.find_first_of( ' ' );
    // Need to check for npos when not found
    if( pos == std::string_view::npos ) {
        std::cout << '"' << sv << '"' << '\n';
        break;
    }
    std::cout << '"' << sv.substr( 0, pos ) << '"' << '\n';
    // Easy to go off by 1 here
    sv.remove_prefix( pos + 1 );
}
```

```
daw::string_view sv = "This is a test  of the test system   ";
while( not sv.empty( ) ) {
  auto part = sv.pop_front_until( ' ' );
  std::cout << '"' << part << '"' << '\n';
}
```

Shouldn’t need Kelly Clarkson to trim strings

- Trimming of strings is painful

```
std::string_view sv = "   This is a test  of the test system   ";
auto pos = sv.find_first_not_of( ' ' );
// npos checking
if( pos != std::string_view::npos ) {
  sv.remove_prefix( pos );
}
pos = sv.find_last_not_of( ' ' );
if( pos != std::string_view::npos ) {
  // Easy off by 1
  sv.remove_suffix( sv.size( ) - pos - 1 );
}
std::cout << '"' << sv << '"' << '\n';
```

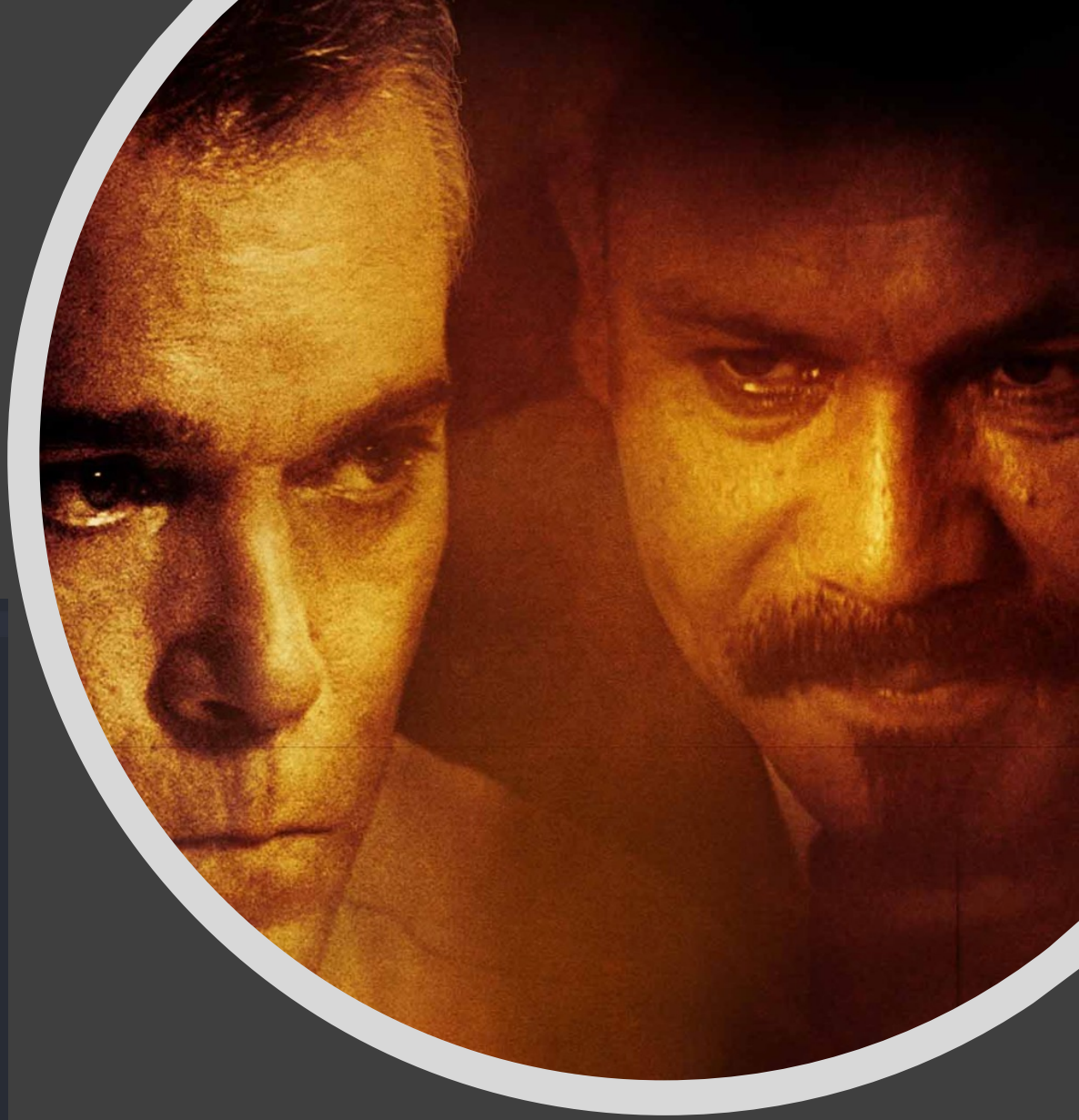
```
daw::string_view sv = "   This is a test  of the test system   ";
sv.trim( );
std::cout << '"' << sv << '"' << '\n';
```

The devil is in the details of parsing

- Lots of room for error/UB. Made a lot while doing examples
- The code deals with indices/sizes in a lot of places

```
constexpr uri_parts parse_url( std::string_view uri_string ) {
  auto result = uri_parts{ };
  auto pos = uri_string.find_first_of( "://" );
  if( pos == std::string_view::npos ) {
    result.scheme = uri_string;
    return result;
  }
  result.scheme = uri_string.substr( 0, pos );
  uri_string.remove_prefix( pos );
  // Need to check, UB to remove_prefix if size( ) < 3
  if( uri_string.size( ) <= 3 ) {
    return result;
  }
  uri_string.remove_prefix( 3 );
  pos = uri_string.find( '/' );
  if( pos == std::string_view::npos ) {
    result.authority = uri_string;
    return result;
  }
  result.authority = uri_string.substr( 0, pos );
  uri_string.remove_prefix( pos );
  pos = find_if( uri_string, any_of<'?', '#'> );
  if( pos == std::string_view::npos ) {
    result.path = uri_string;
    return result;
  }
  result.path = uri_string.substr( 0, pos );
```

```
bool const is_query = uri_string[pos] == '?';
uri_string.remove_prefix( pos );
if( is_query ) {
  // Need to check, UB to remove_prefix if size( ) < 1
  if( uri_string.empty( ) ) {
    return result;
  }
  uri_string.remove_prefix( 1 );
  pos = uri_string.find( '#' );
  if( pos == std::string_view::npos ) {
    result.query = uri_string;
    return result;
  }
  result.query = uri_string.substr( 0, pos );
  uri_string.remove_prefix( pos );
  // Need to check, UB to remove_prefix if size( ) < 1
  if( not uri_string.empty( ) ) {
    uri_string.remove_prefix( 1 );
  }
}
if( not uri_string.empty( ) ) {
  if( not is_query ) {
    uri_string.remove_prefix( 1 );
  }
}
result.fragment = uri_string;
return result;
```



With daw::string_view it's cleaner

- Focus on the tokens/operations
- Less/no indices to deal with

```
// scheme://authority:port/path/?query#fragment
constexpr uri_parts parse_url( daw::string_view uri_string ) {
  auto result = uri_parts{ };
  result.scheme = uri_string.pop_front_until( "://" );
  result.authority = uri_string.pop_front_until( '/', daw::nodiscard );
  result.path =
    uri_string.pop_front_until( daw::any_of<'?', '#'>, daw::nodiscard );

  if( uri_string.empty( ) ) {
    return result;
  }
  if( uri_string.front( ) == '?' ) {
    // Never UB to call remove_prefix
    uri_string.remove_prefix( );
    result.query = uri_string.pop_front_until( '#' );
  }

  if( not uri_string.empty( ) and uri_string.front( ) == '#' ) {
    // remove_prefix without dont_clip_to_bounds is guaranteed to succeed.  When
    // empty it will result in an empty string_view by default
    uri_string.remove_prefix( daw::dont_clip_to_bounds );
  }
  result.fragment = uri_string;
  return result;
}
```

We’re Finished

- Questions?