clang-reforge
Automatic whole-codebase source code rewriting tool for security hardening

Jan Korous, Static Security Tools, Apple

EuroLLVM, Apr 15, 2026

The Team

- Aviral Goel, @aviralgoel
- Balázs Benics, @steakhal
- Jan Korous, @jkorous
- Rashmi Mudduluru, @t-rasmud
- Ziqing Luo, @ziqingluo-90

Agenda

Automated Adoption of C++ Safe Buffers

New Wide-Pointer: bounded_ptr<T>

Retrofitting Bounds-Safety to Existing C++ Code

Summary-Based Pointer Classification

Cross-TU Reasoning by SSAF

Status

Agenda

Automated Adoption of C++ Safe Buffers

New Wide-Pointer: bounded_ptr<T>

Retrofitting Bounds-Safety to Existing C++ Code

Summary-Based Pointer Classification

Cross-TU Reasoning by SSAF

Status

Agenda

Automated Adoption of C++ Safe Buffers

New Wide-Pointer: bounded_ptr<T>

Retrofitting Bounds-Safety to Existing C++ Code

Summary-Based Pointer Classification

Cross-TU Reasoning by SSAF

Status

Agenda

Automated Adoption of C++ Safe Buffers

New Wide-Pointer: bounded_ptr<T>

Retrofitting Bounds-Safety to Existing C++ Code

Summary-Based Pointer Classification

Cross-TU Reasoning by SSAF

Status

Agenda

Automated Adoption of C++ Safe Buffers

New Wide-Pointer: bounded_ptr<T>

Retrofitting Bounds-Safety to Existing C++ Code

Summary-Based Pointer Classification

Cross-TU Reasoning by SSAF

Status

Agenda

Automated Adoption of C++ Safe Buffers

New Wide-Pointer: bounded_ptr<T>

Retrofitting Bounds-Safety to Existing C++ Code

Summary-Based Pointer Classification

Cross-TU Reasoning by SSAF

Status

Automated Adoption of C++ Safe Buffers

C++ Safe Buffers

- OOB memory access is a leading cause of software vulnerabilities

C++ Safe Buffers

- OOB memory access is a leading cause of software vulnerabilities
- C++ Safe Buffers prohibits unsafe pointer operations

```
ptr += step;          warning: [-Wunsafe-buffer-usage]
buf[i] = res;         warning: [-Wunsafe-buffer-usage]
```

C++ Safe Buffers

- OOB memory access is a leading cause of software vulnerabilities
- C++ Safe Buffers prohibits unsafe pointer operations

```
ptr += step;            warning: [-Wunsafe-buffer-usage]
buf[i] = res;           warning: [-Wunsafe-buffer-usage]
```

- Libc++ hardening provides bounds-safe alternatives

  `std::span`, `std::array`, `std::vector`, …

C++ Safe Buffers

- OOB memory access is a leading cause of software vulnerabilities
- C++ Safe Buffers prohibits unsafe pointer operations

```
ptr += step;          warning: [-Wunsafe-buffer-usage]
buf[i] = res;         warning: [-Wunsafe-buffer-usage]
```

- Libc++ hardening provides bounds-safe alternatives

  `std::span`, `std::array`, `std::vector`, …

- Adoption in existing code is expensive

C++ Safe Buffers

- OOB memory access is a leading cause of software vulnerabilities
- C++ Safe Buffers prohibits unsafe pointer operations

```
ptr += step;          warning: [-Wunsafe-buffer-usage]
buf[i] = res;         warning: [-Wunsafe-buffer-usage]
```

- Libc++ hardening provides bounds-safe alternatives
  `std::span`, `std::array`, `std::vector`, …

- Adoption in existing code is expensive

AUTOMATED with clang-reforge!

clang-reforge

- Automatically rewrites C++ codebase to be bounds-safe
  - Replaces unsafe pointers and arrays with bounds-safe types
- Uses static analysis; requires cross-TU reasoning
  - Based on Scalable Static Analysis Framework in clang
- In development in mainline LLVM

Prototype

- Built a prototype
- Hardened a security relevant C++ codebase
  - 50 kLoC++
  - 2000 line diff
  - Manual adoption would've taken months
- Need for a new pointer replacement type

New Wide-Pointer: bounded_ptr<T>

Bounds-Safe Pointer Replacement

- ✅ Has analogy of pointer operations
- ✅ Tracks bounds
- ✅ No ownership semantics

Bounds-Safe Pointer Replacement

```
int* find_last_nonzero(int* first, int* last) {
    while (last >= first) {
        if (*last != 0)
            return last;
        --last; // warning: [-Wunsafe-buffer-usage]
    }
    return nullptr;
}
```

New Wide-Pointer for Automatic Adoption

```
template<typename T>
class bounded_ptr {
public:
  T& operator*();      // bounds-checked
  T& operator[](...); // bounds-checked
  bounded_ptr operator+(...);
  bounded_ptr operator-(...);
  bounded_ptr& operator++(...);
  bounded_ptr& operator--(...);

  static bounded_ptr _new(size_t sz);
  void _delete();
  // ...
};
```

Retrofitting Bounds-Safety to Existing C++ Code

Hardening Unsafe C++ Code

```cpp
struct RGBABitMap {
  uint8_t* pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    uint8_t* pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Hardening Unsafe C++ Code

```
struct RGBABitMap {
  uint8_t* pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    uint8_t* pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;              warning: [-Wunsafe-buffer-usage]
  }                          warning: [-Wunsafe-buffer-usage]
  // ...
```

Hardening Unsafe C++ Code

```
struct RGBABitMap {
  uint8_t* pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    uint8_t* pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Hardening Unsafe C++ Code

```
template<typename T>
class bounded_ptr

 T& operator[](...); // safe
```

```
struct RGBABitMap {
  uint8_t* pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    bounded_ptr<uint8_t> pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Hardening Unsafe C++ Code

```
struct RGBABitMap {
  uint8_t* pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    bounded_ptr<uint8_t> pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Hardening Unsafe C++ Code

```
struct RGBABitMap {
  uint8_t* pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    bounded_ptr<uint8_t> pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Hardening Unsafe C++ Code

```
struct RGBABitMap {
  bounded_ptr<uint8_t> pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    bounded_ptr<uint8_t> pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

```
template<typename T>
class bounded_ptr

 bounded_ptr operator+(...);
```

Hardening Unsafe C++ Code

```
struct RGBABitMap {
  bounded_ptr<uint8_t> pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    bounded_ptr<uint8_t> pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Hardening Unsafe C++ Code

```
struct RGBABitMap {
  bounded_ptr<uint8_t> pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    bounded_ptr<uint8_t> pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Hardening Unsafe C++ Code

```
struct RGBABitMap {
  bounded_ptr<uint8_t> pixels;
  long width, height;

  void load(long width, long height) {
    pixels = bounded_ptr<uint8_t>::_new(width * height * 4);
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    bounded_ptr<uint8_t> pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

```
template<typename T>
class bounded_ptr

 static bounded_ptr _new(...);
```

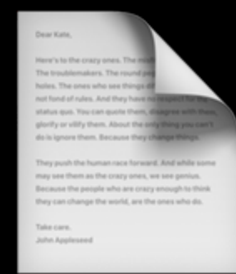
Hardening Unsafe C++ Code

```
struct RGBABitMap {
  bounded_ptr<uint8_t> pixels;
  long width, height;

  void load(long width, long height) {
    pixels = bounded_ptr<uint8_t>::_new(width * height * 4);
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    bounded_ptr<uint8_t> pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Summary-Based Pointer Classification

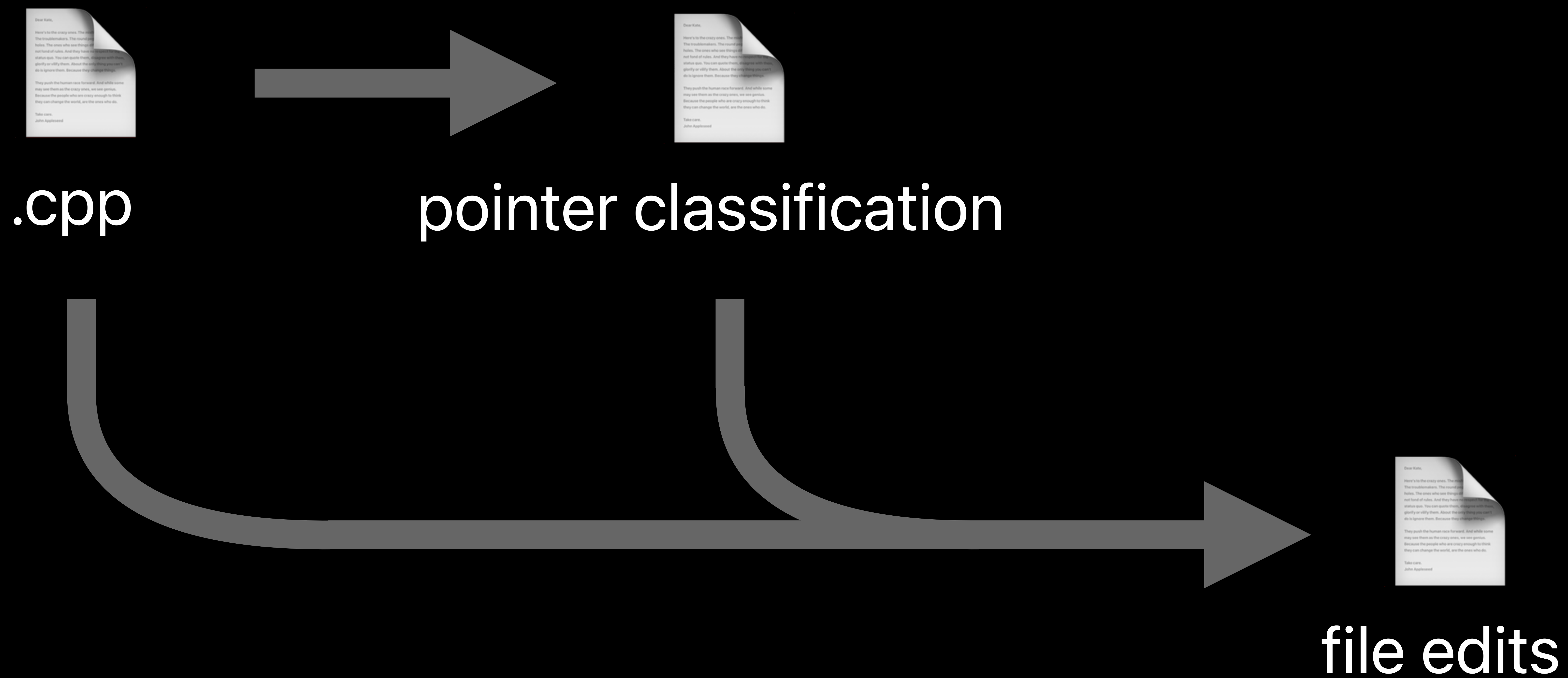
Hardening Unsafe C++ Code



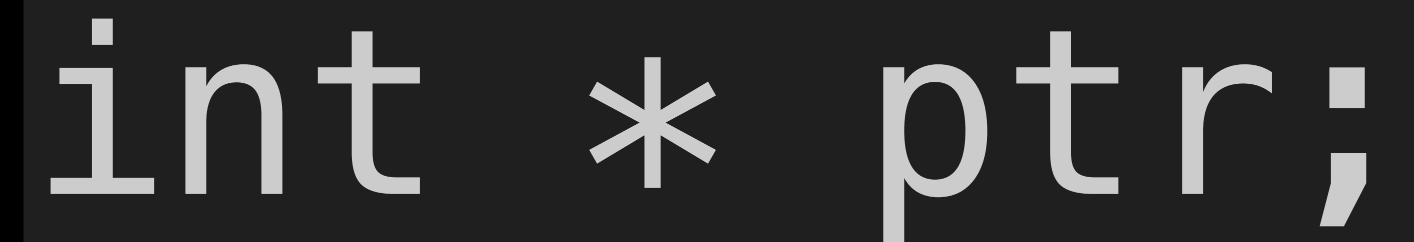
.cpp

pointer classification

Hardening Unsafe C++ Code



Pointer Classification



`BUFFER`

- either used in arithmetic
- or flows to a `BUFFER`

`SINGLE`

- neither used in arithmetic
- nor flows to a `BUFFER`

Pointer Classification

```
struct RGBABitMap {
  uint8_t* pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    uint8_t* pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Pointer Classification

```
struct RGBABitMap {
  uint8_t* pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    uint8_t* pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Pointer Classification

```
struct RGBABitMap {
  uint8_t* pixels;
  long width, height;

  void load(long width, ...
    pixels = new uint8_t[...
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, ...
    uint8_t* pixel = pixels + ...
    pixel[3] = alpha;
  }
  // ...
```

Summary

Pointer Classification

```
uint8_t* pixels;


  pixels = new uint8_t[ ];



  uint8_t* pixel = pixels + ...
  pixel[ ]
```

```
Summary
  Unsafe Buffers
```

Pointer Classification

```
uint8_t* pixels;

  pixels = new uint8_t[ ];

  uint8_t* pixel = pixels + ...
  pixel[ ]
```

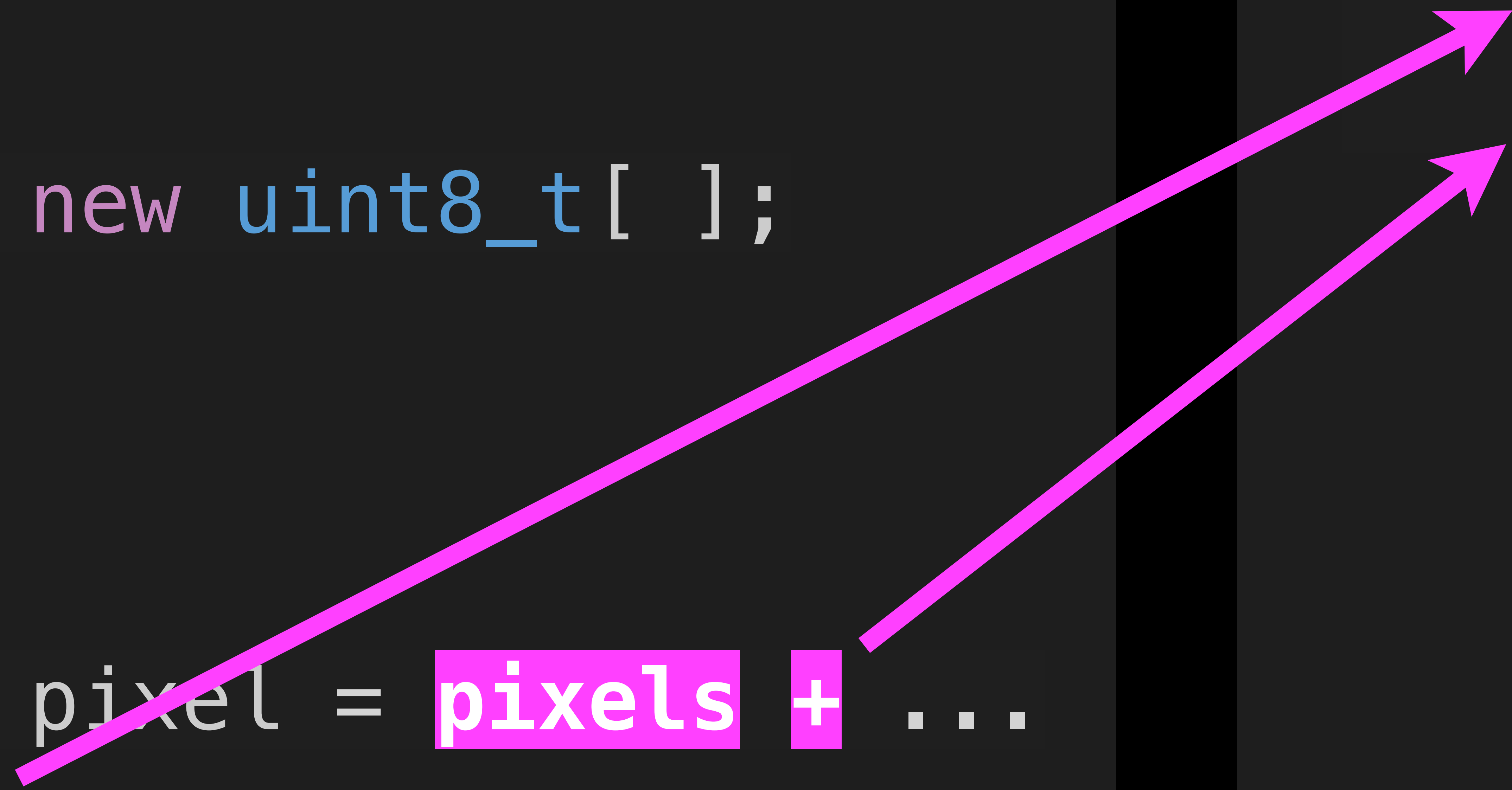
```
Summary
  Unsafe Buffers
```

Pointer Classification

```
uint8_t* pixels;

  pixels = new uint8_t[ ];

  uint8_t* pixel = pixels + ...
  pixel[ ]
```



```
Summary
  Unsafe Buffers
    pixel
    pixels
```

Pointer Classification

```
uint8_t* pixels;


  pixels = new uint8_t[ ];



  uint8_t* pixel = pixels + ...
  pixel[ ]
```

```
Summary
  Unsafe Buffers
    pixel
    pixels
  Pointer Flow
```

Pointer Classification

```
uint8_t* pixels;

  pixels = new uint8_t[ ];

  uint8_t* pixel = pixels + ...
  pixel[ ]
```

```
Summary
  Unsafe Buffers
    pixel
    pixels
  Pointer Flow
```

Pointer Classification

```
uint8_t* pixels;

  pixels = new uint8_t[ ];

  uint8_t* pixel = pixels + ...
  pixel[ ]
```

```
Summary
  Unsafe Buffers
    pixel
    pixels
  Pointer Flow
    pixels -> pixel
```

Pointer Classification

```
struct RGBABitMap {
  uint8_t* pixels;
  long width, height;

  void load(long width, ...
    pixels = new uint8_t[...
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, ...
    uint8_t* pixel = pixels + …
    pixel[3] = alpha;
  }
  // ...
```

```
Summary
  Unsafe Buffers
    pixel
    pixels
  Pointer Flow
    pixels -> pixel
```

Pointer Classification

```
Summary
  Unsafe Buffers
    pixel
    pixels
  Pointer Flow
    pixels -> pixel
```

Pointer Classification

```
Summary
  Unsafe Buffers
    pixel
    pixels
  Pointer Flow
    pixels -> pixel
```

```
Analysis Result
  Buffer pointers
    pixel
    pixels
```

Source Edit Generation

```
struct RGBABitMap {
  uint8_t* pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    uint8_t* pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Source Edit Generation

```
struct RGBABitMap {
  bounded_ptr<uint8_t> pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    bounded_ptr<uint8_t> pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Source Edit Generation

```
struct RGBABitMap {
  bounded_ptr<uint8_t> pixels;
  long width, height;

  void load(long width, long height) {
    pixels = new uint8_t[width * height * 4];
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    bounded_ptr<uint8_t> pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Source Edit Generation

```
struct RGBABitMap {
  bounded_ptr<uint8_t> pixels;
  long width, height;

  void load(long width, long height) {
    pixels = bounded_ptr<uint8_t>::_new(width * height * 4);
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    bounded_ptr<uint8_t> pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Source Edit Generation

```
struct RGBABitMap {
  bounded_ptr<uint8_t> pixels;
  long width, height;

  void load(long width, long height) {
    pixels = bounded_ptr<uint8_t>::_new(width * height * 4);
    this->width = width;
    this->height = height;
  }
  void setAlpha(long X, long Y, uint8_t alpha) {
    bounded_ptr<uint8_t> pixel = pixels + (Y * width + X) * 4;
    pixel[3] = alpha;
  }
  // ...
```

Cross-TU Reasoning by SSAF

Scalable Static Analysis Framework

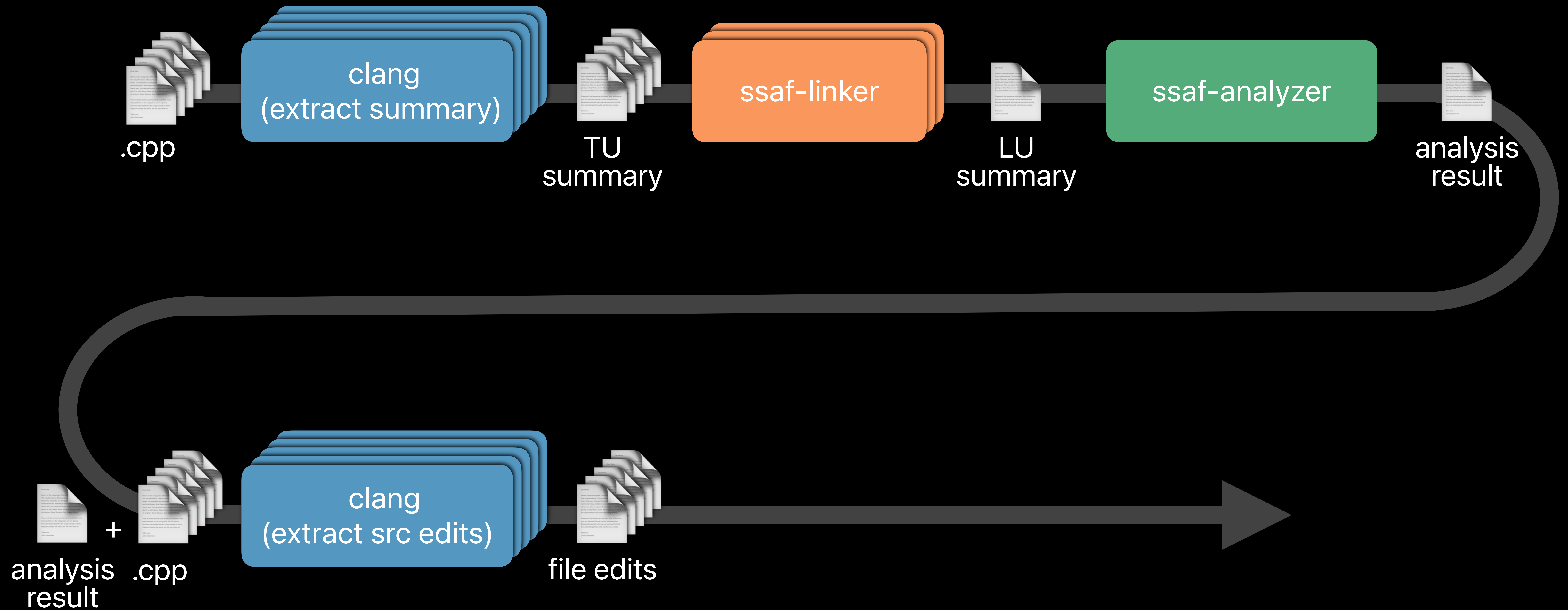
- Program entities abstract away declarations in individual TUs
  - Simplifies analysis implementation
- Model of code: summaries
  - Sparse abstract (analysis-specific) representation
  - Decoupled from the AST; scalable
    - Accurate model of linking
- General-purpose framework

Architecture

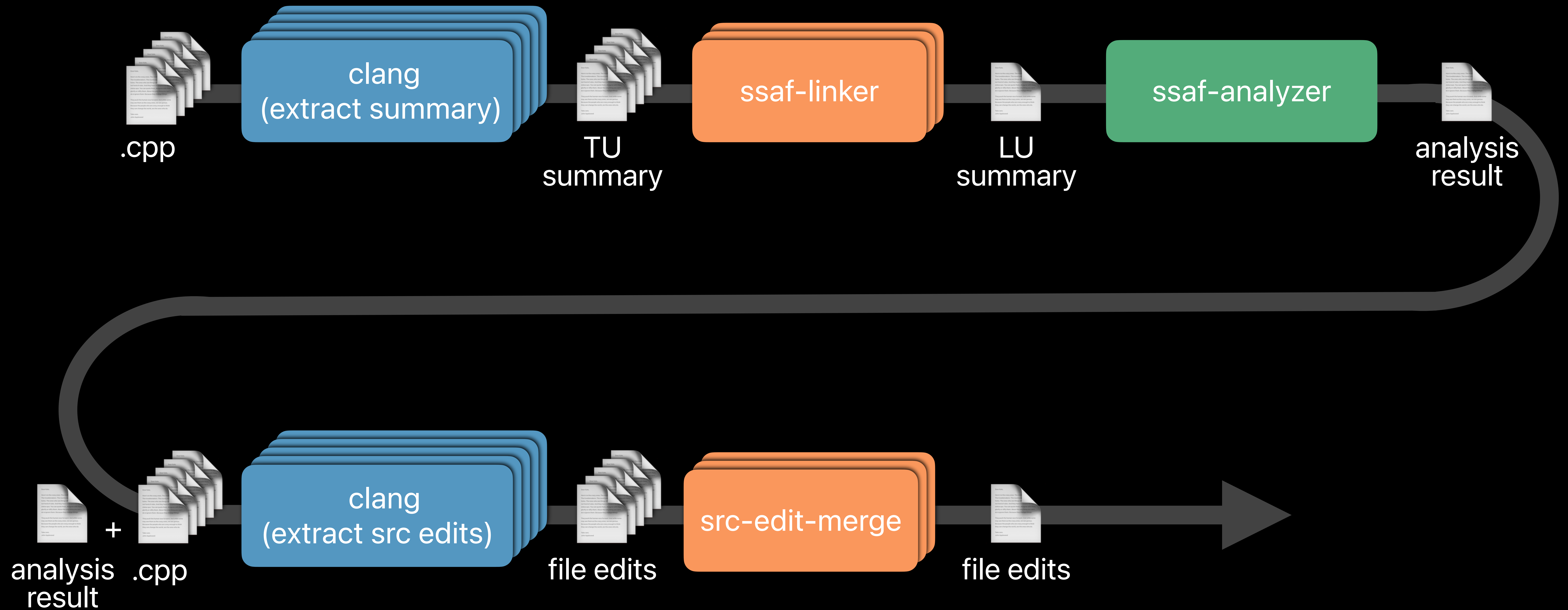
Architecture

Architecture

Architecture



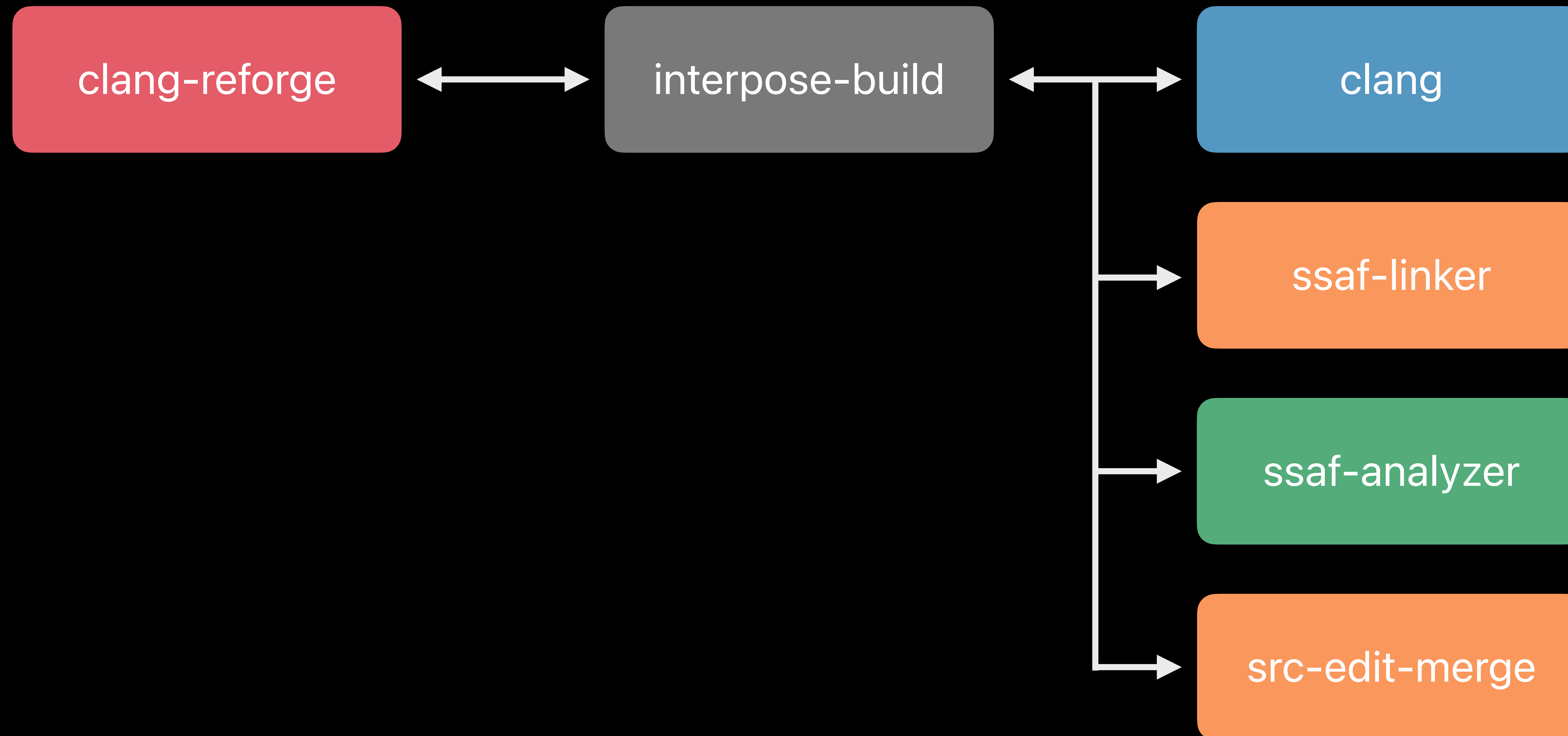
Architecture



Architecture

Architecture

Architecture



Status

Status

- SSAF
  - RFC (Oct 2025): https://discourse.llvm.org/t/88678
  - in development:
    - `clang --fextract-summaries`
    - `clang-ssaf-linker`
    - `clang-ssaf-analyzer`

Status

- clang-reforge
  - RFC (March 2026): https://discourse.llvm.org/t/90283
  - developing: `UnsafeBuffers` and `PointerFlow` summaries

Status

- `bounded_ptr<T>`, `bounded_array<T>`
  - Prototype implementation
  - Experimental adoption
  - Plan to develop all aspects of the interface and hardening
  - Aim to be added to the C++ Standard Library

Summary

Clang-reforge automates C++ Safe Buffers adoption in existing code

bounded_ptr is a purpose-designed wide-pointer for automatic adoption

Retrofitting bounds-safety to existing is a recursive process

Clang-reforge uses summaries to classify pointers; then generates source edits

SSAF allows clang-reforge to transform code across source files of entire projects

Summary

Clang-reforge automates C++ Safe Buffers adoption in existing code

bounded_ptr is a purpose-designed wide-pointer for automatic adoption

Retrofitting bounds-safety to existing is a recursive process

Clang-reforge uses summaries to classify pointers; then generates source edits

SSAF allows clang-reforge to transform code across source files of entire projects

Summary

Clang-reforge automates C++ Safe Buffers adoption in existing code

bounded_ptr is a purpose-designed wide-pointer for automatic adoption

Retrofitting bounds-safety to existing is a recursive process

Clang-reforge uses summaries to classify pointers; then generates source edits

SSAF allows clang-reforge to transform code across source files of entire projects

Summary

Clang-reforge automates C++ Safe Buffers adoption in existing code

bounded_ptr is a purpose-designed wide-pointer for automatic adoption

Retrofitting bounds-safety to existing is a recursive process

Clang-reforge uses summaries to classify pointers; then generates source edits

SSAF allows clang-reforge to transform code across source files of entire projects

Summary

Clang-reforge automates C++ Safe Buffers adoption in existing code

bounded_ptr is a purpose-designed wide-pointer for automatic adoption

Retrofitting bounds-safety to existing is a recursive process

Clang-reforge uses summaries to classify pointers; then generates source edits

SSAF allows clang-reforge to transform code across source files of entire projects

TM and © 2026 Apple Inc. All rights reserved.