SW 재난 방지를 위한 안전한 프로그래밍 시스템

허기홍
KAIST 전산학부

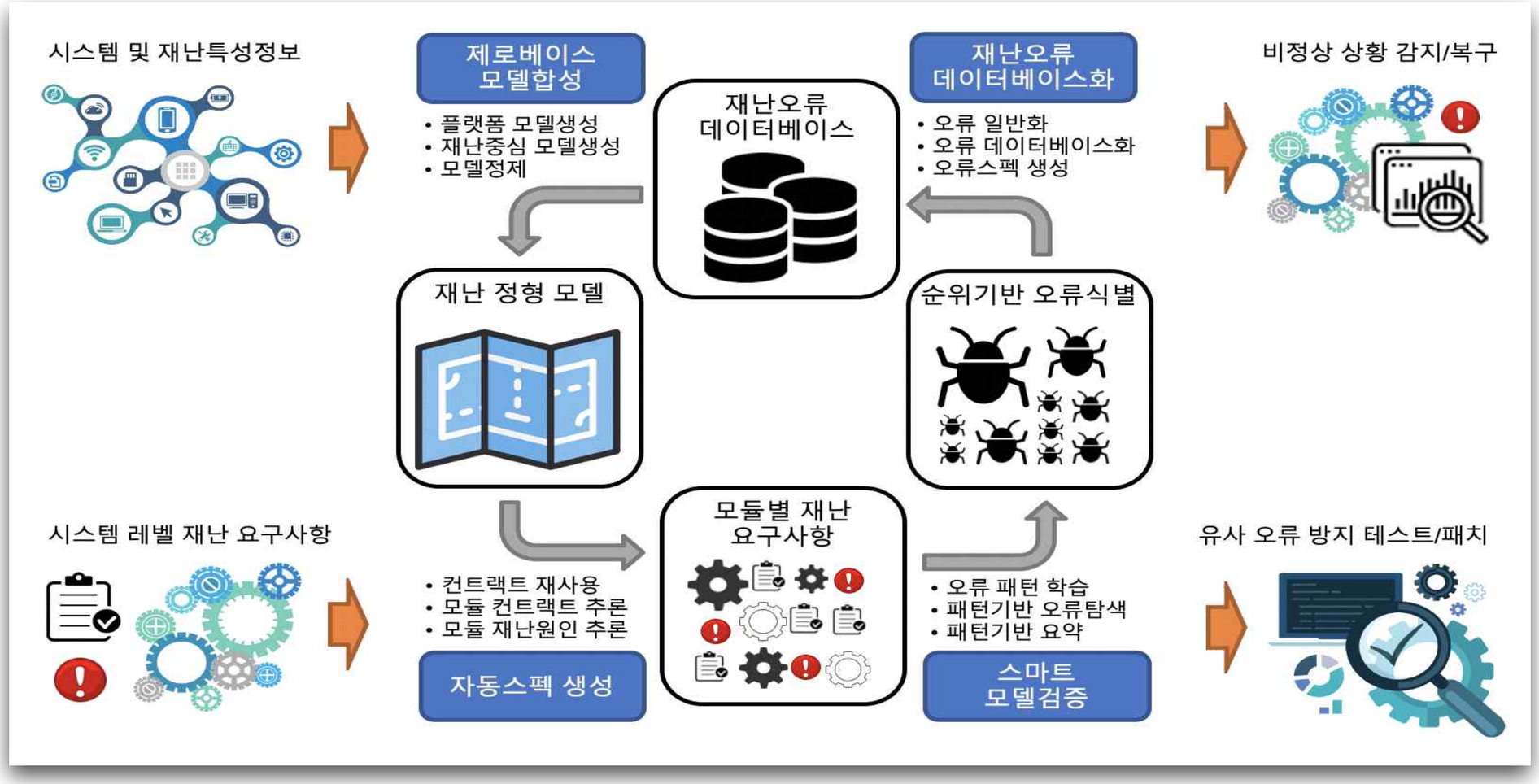
2024 겨울 SW 재난연구센터 워크샵

KAIST

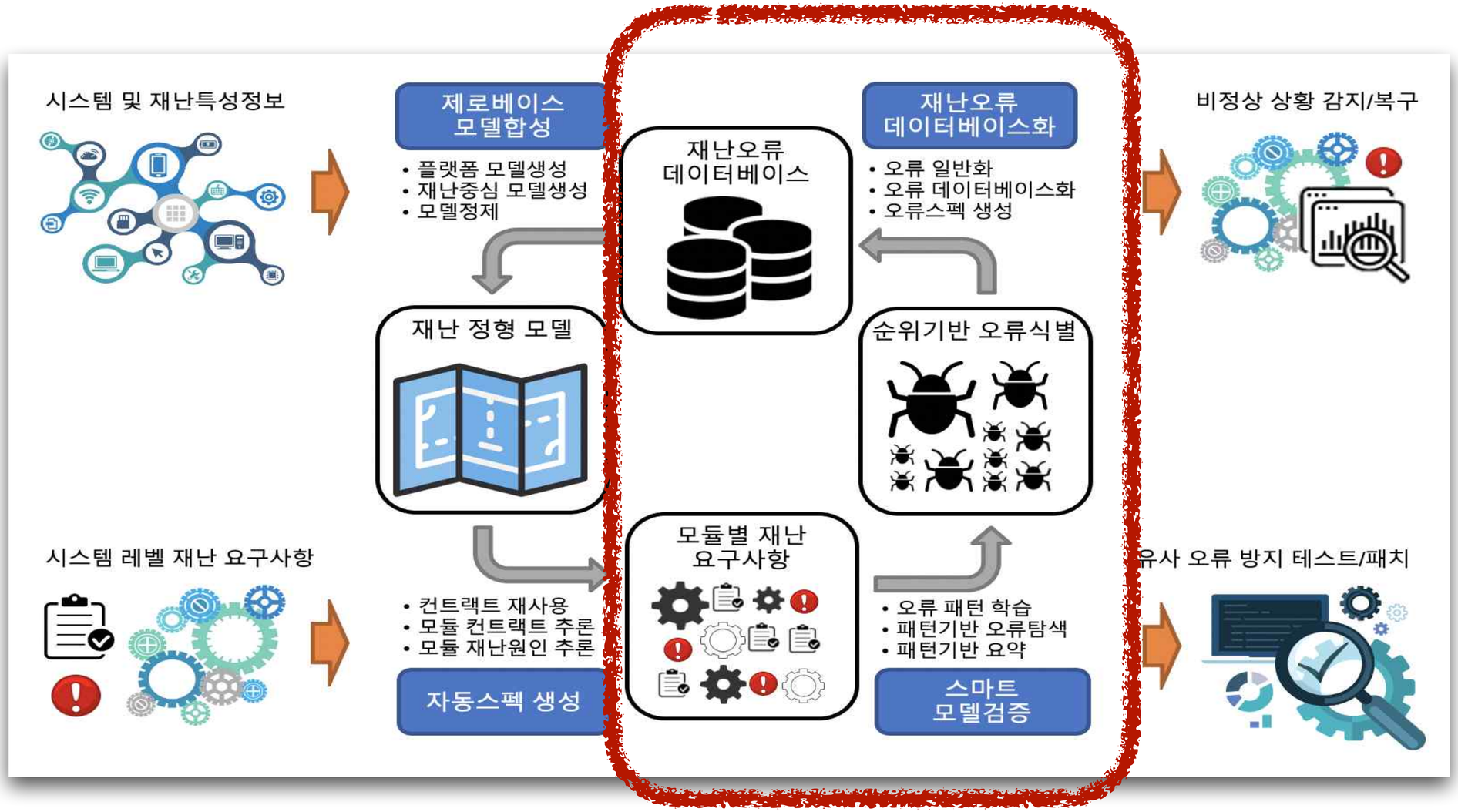
Programming Systems Laboratory

3그룹의 목표

3그룹의 목표


시스템 및 재난특성정보
제로베이스
모델합성
• 플랫폼 모델생성
• 재난중심 모델생성
• 모델정제
재난오류
데이터베이스
재난오류
데이터베이스화
• 오류 일반화
• 오류 데이터베이스화
• 오류스펙 생성
비정상 상황 감지/복구
재난 정형 모델
순위기반 오류식별
시스템 레벨 재난 요구사항
• 컨트랙트 재사용
• 모듈 컨트랙트 추론
• 모듈 재난원인 추론
자동스펙 생성
모듈별 재난
요구사항
• 오류 패턴 학습
• 패턴기반 오류탐색
• 패턴기반 요약
스마트
모델검증
유사 오류 방지 테스트/패치

3그룹의 목표


시스템 및 재난특성정보
제로베이스 모델합성
• 플랫폼 모델생성
• 재난중심 모델생성
• 모델정제
재난 정형 모델
시스템 레벨 재난 요구사항
• 컨트랙트 재사용
• 모듈 컨트랙트 추론
• 모듈 재난원인 추론
자동스펙 생성
재난오류 데이터베이스
재난오류 데이터베이스화
• 오류 일반화
• 오류 데이터베이스화
• 오류스펙 생성
순위기반 오류식별
모듈별 재난 요구사항
• 오류 패턴 학습
• 패턴기반 오류탐색
• 패턴기반 요약
스마트 모델검증
비정상 상황 감지/복구
유사 오류 방지 테스트/패치

우리가 꿈꾸는 프로그래밍

우리가 꿈꾸는 프로그래밍

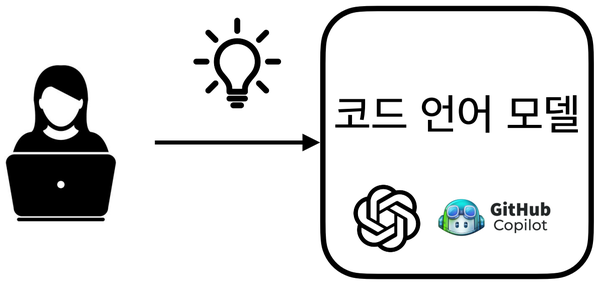
- 목표: 손쉽고 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습

우리가 꿈꾸는 프로그래밍

- 목표: 손쉽고 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습

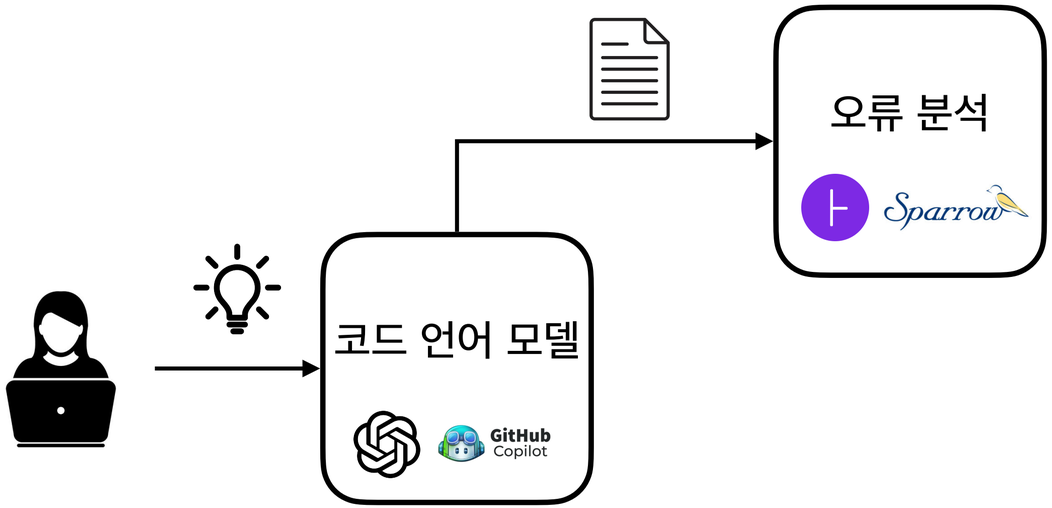
우리가 꿈꾸는 프로그래밍

- 목표: 손쉽고 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



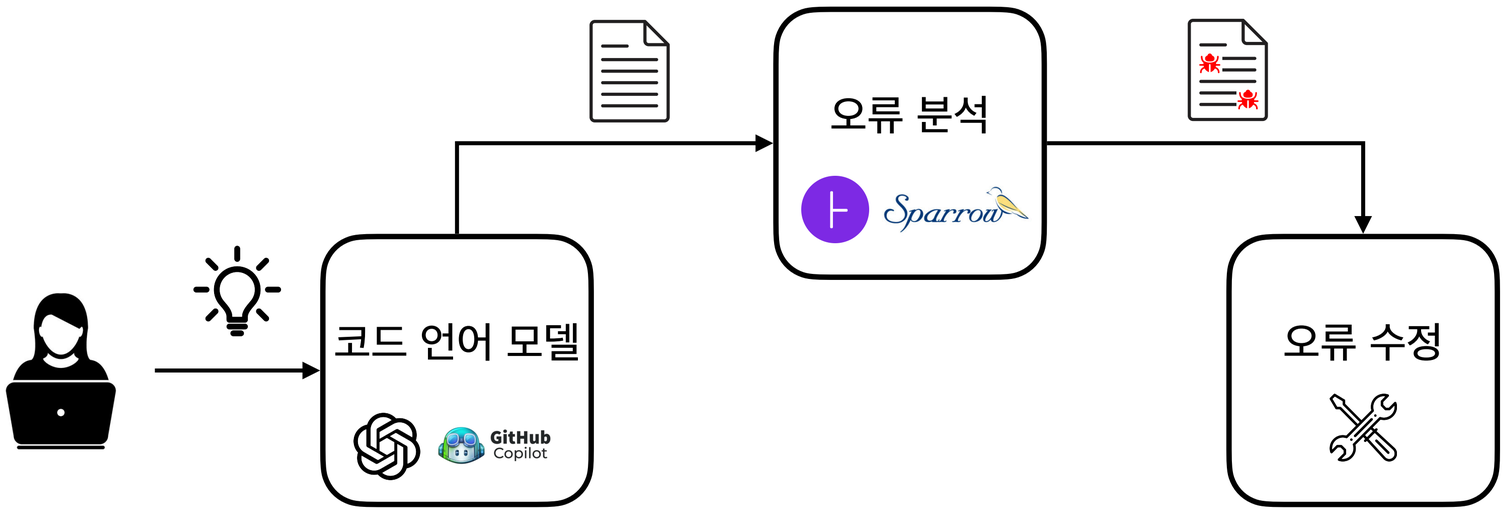
우리가 꿈꾸는 프로그래밍

- 목표: 손쉽고 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



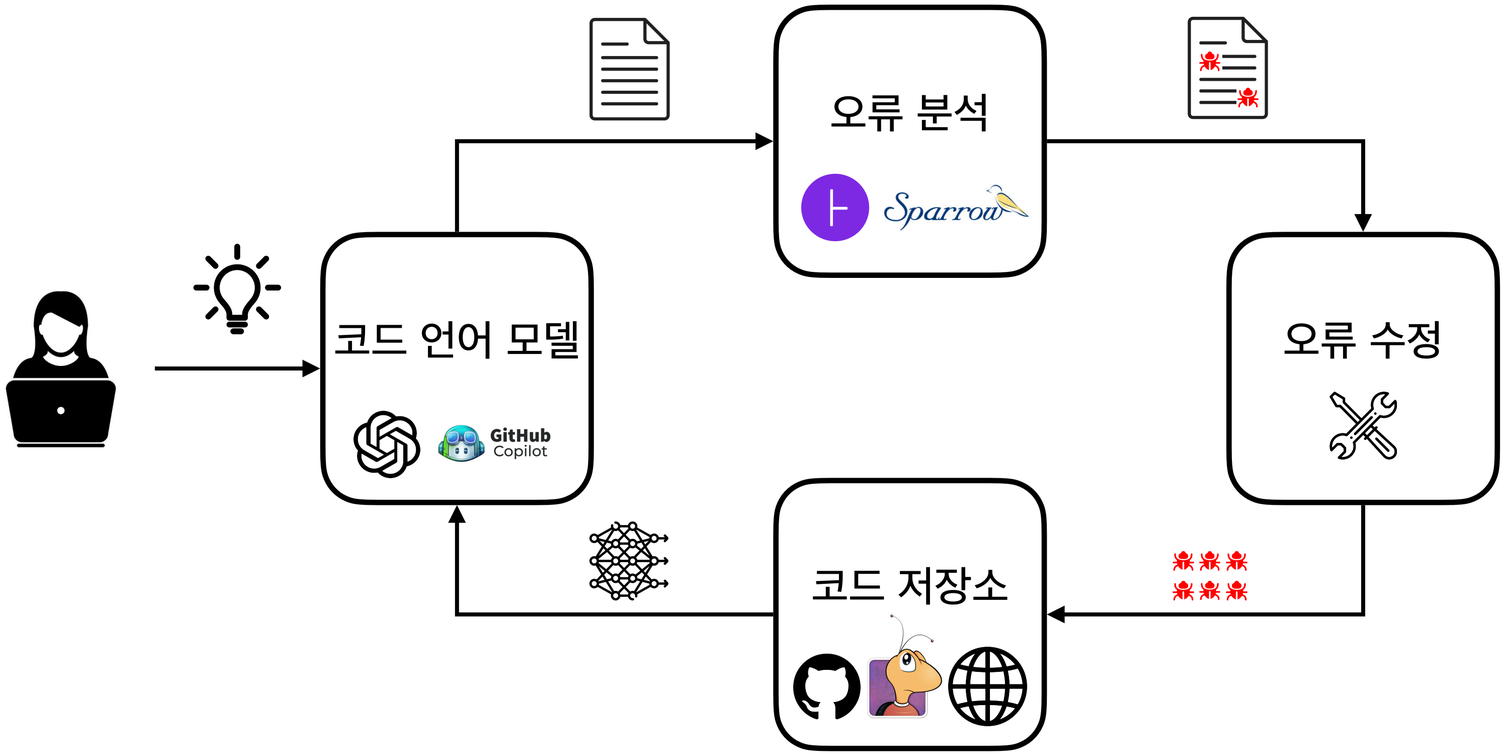
우리가 꿈꾸는 프로그래밍

- 목표: 손쉽고 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



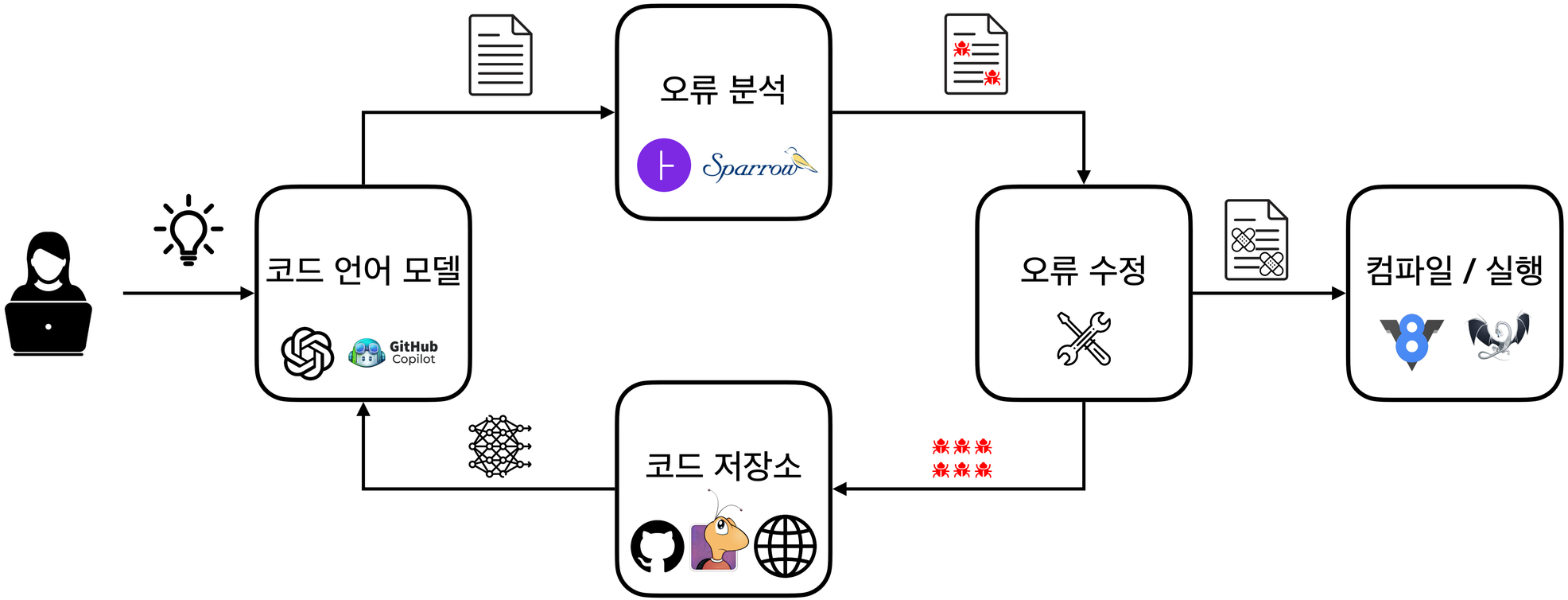
우리가 꿈꾸는 프로그래밍

- 목표: 손쉽고 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



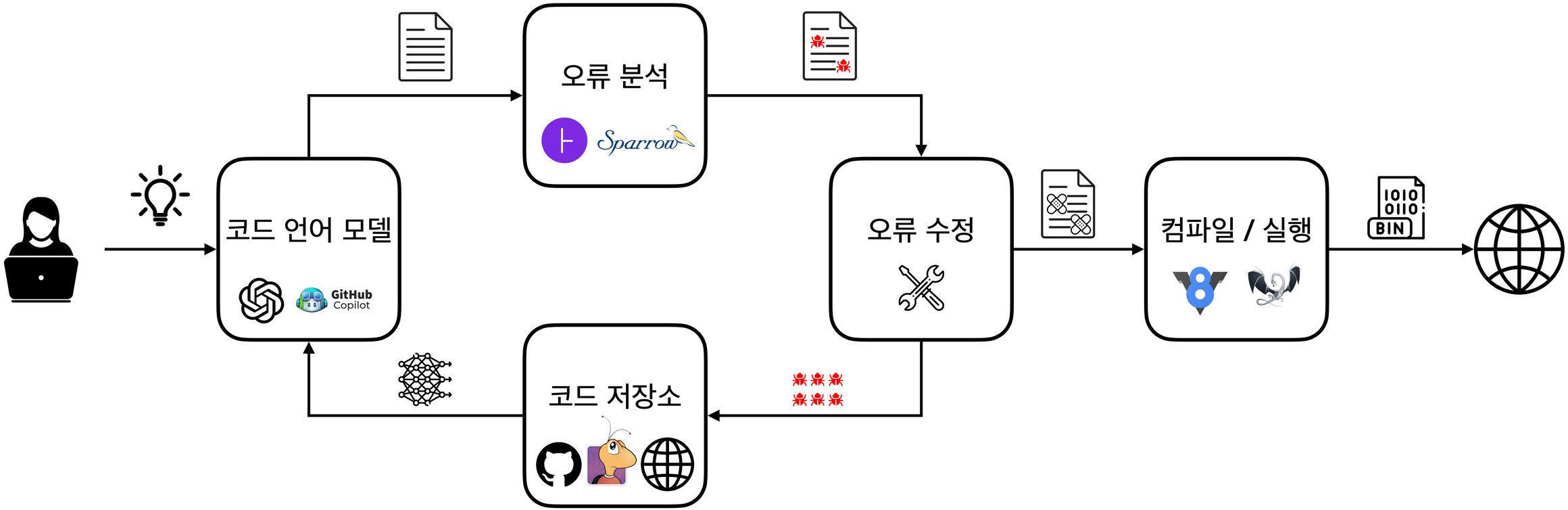
우리가 꿈꾸는 프로그래밍

- 목표: 손쉽고 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



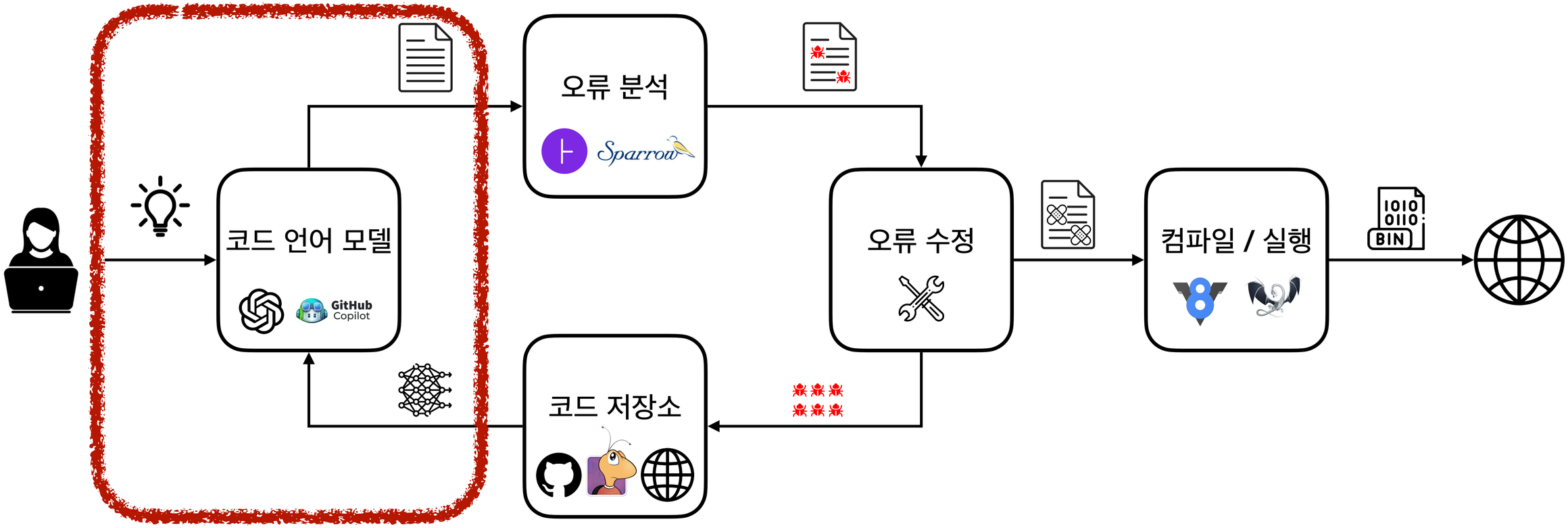
우리가 꿈꾸는 프로그래밍

- 목표: 손쉽고 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



우리가 꿈꾸는 프로그래밍

- 목표: 손쉽고 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



반복되는 SW 오류

반복되는 SW 오류

```
long ToL (char *pbuffer) { return (puffer[0] | puffer[1]<<8 | puffer[2]<<16 | puffer[3]<<24); }

short ToS (char *pbuffer) { return ((short)(puffer[0] | puffer[1]<<8)); }

gint32 ReadBMP (gchar *name, GError **error) {
    if (fread(buffer, Bitmap_File_Head.biSize - 4, fd) != 0)
        FATALP ("BMP: Error reading BMP file header #3");
    Bitmap_Head.biWidth = ToL (&buffer[0x00]);
    Bitmap_Head.biBitCnt = ToS (&buffer[0x0A]);

    rowbytes = ((Bitmap_Head.biWidth * Bitmap_Head.biBitCnt - 1) / 32) * 4 + 4;
    image_ID = ReadImage (rowbytes);
    ...
}

gint32 ReadImage (int rowbytes) {
    buffer = malloc(rowbytes);      // malloc with overflowed size
    ...
}
```

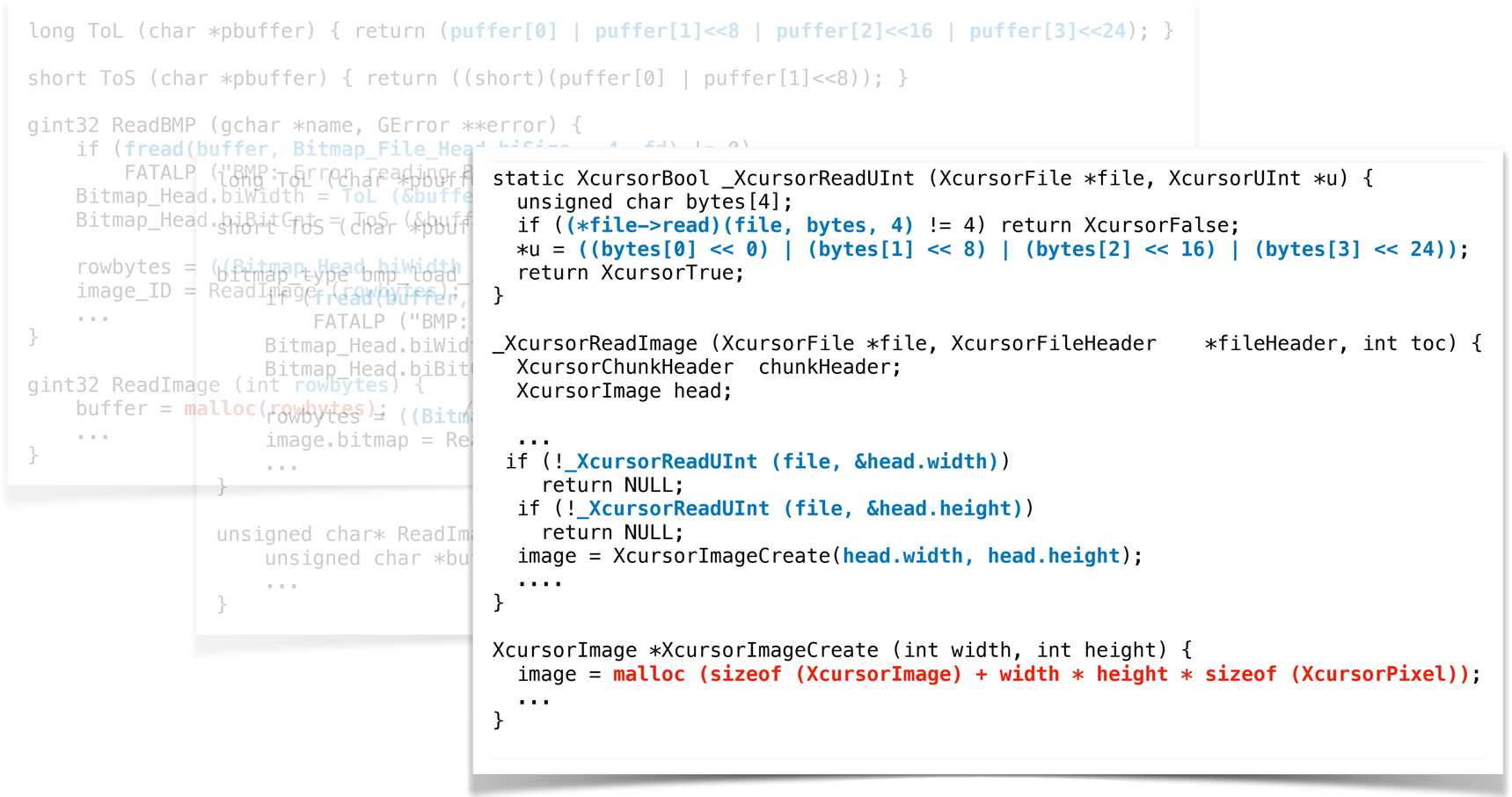
gimp-2.6.7 (CVE-2009-1570)

반복되는 SW 오류

```
long ToL (char *pbuffer) { return (puffer[0] | puffer[1]<<8 | puffer[2]<<16 | puffer[3]<<24); }

short ToS (char *pbuffer) { return ((short)(puffer[0] | puffer[1]<<8)); }

bitmap_type bmp_load_image (FILE* filename) {
    if (fread(buffer, Bitmap_File_Head.biSize - 4, fd) != 0)
        FATALP ("BMP: Error reading BMP file header #3");
    Bitmap_Head.biWidth = ToL (&buffer[0x00]);
    Bitmap_Head.biBitCnt = ToS (&buffer[0x0A]);

    rowbytes = ((Bitmap_Head.biWidth * Bitmap_Head.biBitCnt - 1) / 32) * 4 + 4;
    image.bitmap = ReadImage (rowbytes);
    ...
}

unsigned char* ReadImage (int rowbytes) {
    unsigned char *buffer = (unsigned char*) new char[rowbytes];    // malloc with overflowed size
    ...
}
```

sam2p-0.49.4 (CVE-2017-16663)

반복되는 SW 오류



```
static XcursorBool _XcursorReadUInt (XcursorFile *file, XcursorUInt *u) {
  unsigned char bytes[4];
  if ((*file->read)(file, bytes, 4) != 4) return XcursorFalse;
  *u = ((bytes[0] << 0) | (bytes[1] << 8) | (bytes[2] << 16) | (bytes[3] << 24));
  return XcursorTrue;
}

_XcursorReadImage (XcursorFile *file, XcursorFileHeader    *fileHeader, int toc) {
  XcursorChunkHeader  chunkHeader;
  XcursorImage head;

  ...
 if (!_XcursorReadUInt (file, &head.width))
    return NULL;
  if (!_XcursorReadUInt (file, &head.height))
    return NULL;
  image = XcursorImageCreate(head.width, head.height);
  ....
}

XcursorImage *XcursorImageCreate (int width, int height) {
  image = malloc (sizeof (XcursorImage) + width * height * sizeof (XcursorPixel));
  ...
}
```

libXcursor-1.1.14 (CVE-2017-16612)

반복되는 SW 오류

```
long ToL (char *pbuffer) { return (puffer[0] | puffer[1]<<8 | puffer[2]<<16 | puffer[3]<<24); }

short ToS (char *pbuffer) { return ((short)(puffer[0] | puffer[1]<<8)); }

gint32 ReadBMP (gchar *name, GError **error) {
    if (fread(buffer, Bitmap_File_Head.biSize, 4, fd) != 0)
        FATALP ("BMP: Error reading B
    Bitmap_Head.biWidth = ToL (&buffe
    Bitmap_Head.biBitCnt = ToS (&buff

    rowbytes = ((Bitmap_Head.biWidth
    image_ID = ReadImage (rowbytes);
    ...
}

gint32 ReadImage (int rowbytes) {
    buffer = malloc(rowbytes);
    ...
}
```

```
static XcursorBool _XcursorReadUInt (XcursorFile *file, XcursorUInt *u) {
  unsigned char bytes[4];
  if ((*file->read)(file, bytes, 4) != 4) return XcursorFalse;
  *u = ((bytes[0] << 0) | (bytes[1] << 8) | (bytes[2] << 16) | (bytes[3] << 24));
  return XcursorTrue;
}

_XcursorReadImage (XcursorFile *file, XcursorFileHeader    *fileHeader, int toc) {
  XcursorChunkHeader  chunkHeader;
  XcursorImage head;

  ...
 if (!_XcursorReadUInt (file, &head.width))
    return NULL;
  if (!_XcursorReadUInt (file, &head.height))
    return NULL;
  image = XcursorImageCreate(head.width, head.height);
  ....
}

XcursorImage *XcursorImageCreate (int width, int height) {
  image = malloc (sizeof (XcursorImage) + width * height * sizeof (XcursorPixel));
  ...
}
```

Tracer: 반복되는 오류 탐지를 위한 정적 분석 시스템 [CCS'22]

libXcursor-1.1.14 (CVE-2017-16612)

반복되는 SW 오류

```
int toLong(char *buffer) {
  return (buffer[0]) | (buffer[1] << 8) | (buffer[2] << 16) | (buffer[3] << 24);
}

int f(char *name) {
  int width, height, area;
  char buffer[10];
  FILE *fd = fopen(name, "rb");
  fread(buffer, 10, 1, fd);
  fclose(fd);

  // Copilot, fill it!
```

GitHub Copilot

반복되는 SW 오류

```
int toLong(char *buffer) {
  return (buffer[0]) | (buffer[1] << 8) | (buffer[2] << 16) | (buffer[3] << 24);
}

int f(char *name) {
  int width, height, area;
  char buffer[10];
  FILE *fd = fopen(name, "rb");
  fread(buffer, 10, 1, fd);
  fclose(fd);

  // Copilot, fill it!
  width = toLong(buffer + 18);
  height = toLong(buffer + 22);
  area = width * height;
```

GitHub Copilot

반복되는 SW 오류

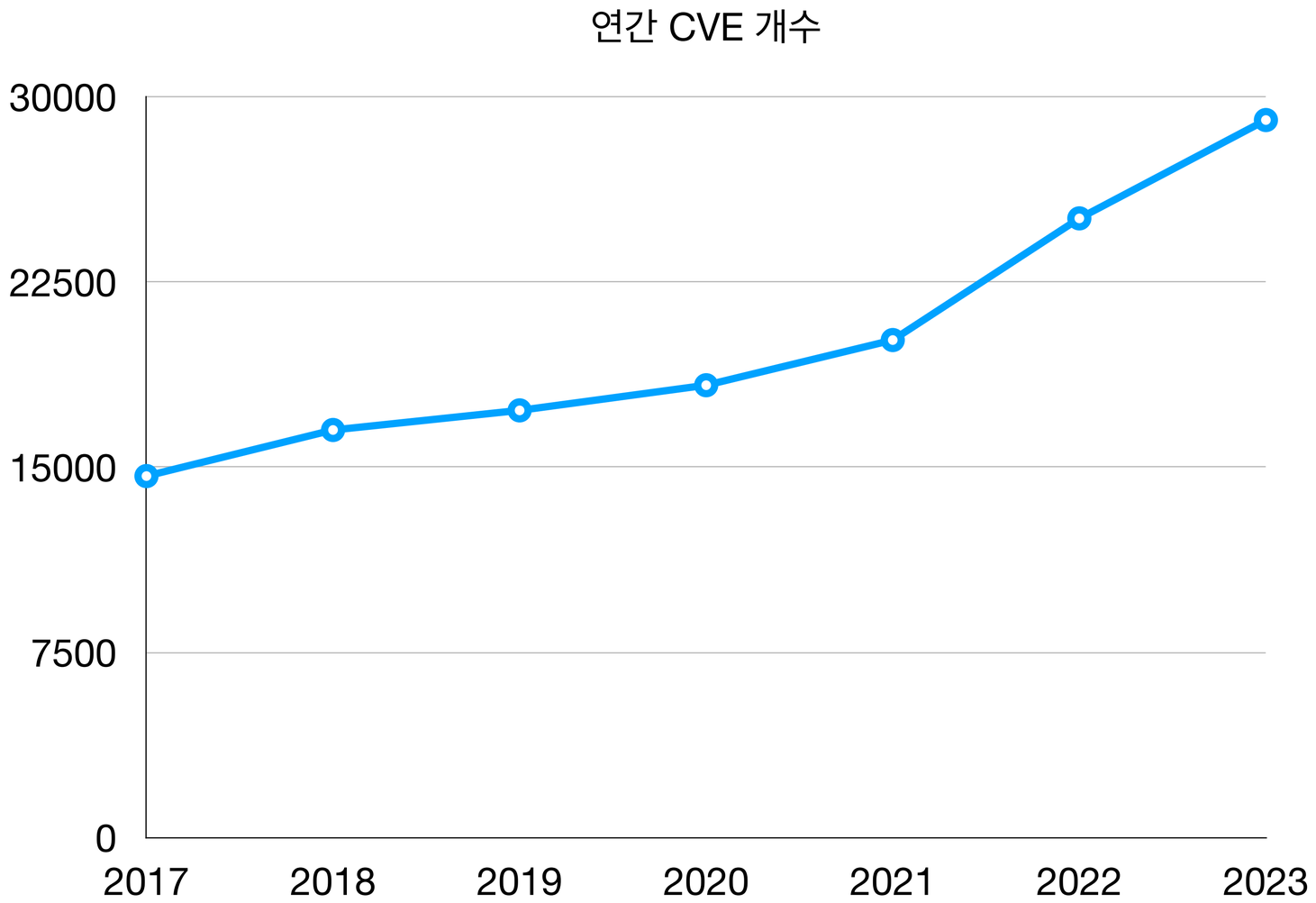
```
int toLong(char *buffer) {
  return (buffer[0]) | (buffer[1] << 8) | (buffer[2] << 16) | (buffer[3] << 24);
}

int f(char *name) {
  int width, height, area;
  char buffer[10];
  FILE *fd = fopen(name, "rb");
  fread(buffer, 10, 1, fd);
  fclose(fd);

  // Copilot, fill it!
  width = toLong(buffer + 18);
  height = toLong(buffer + 22);
  area = width * height;
```

GitHub Copilot

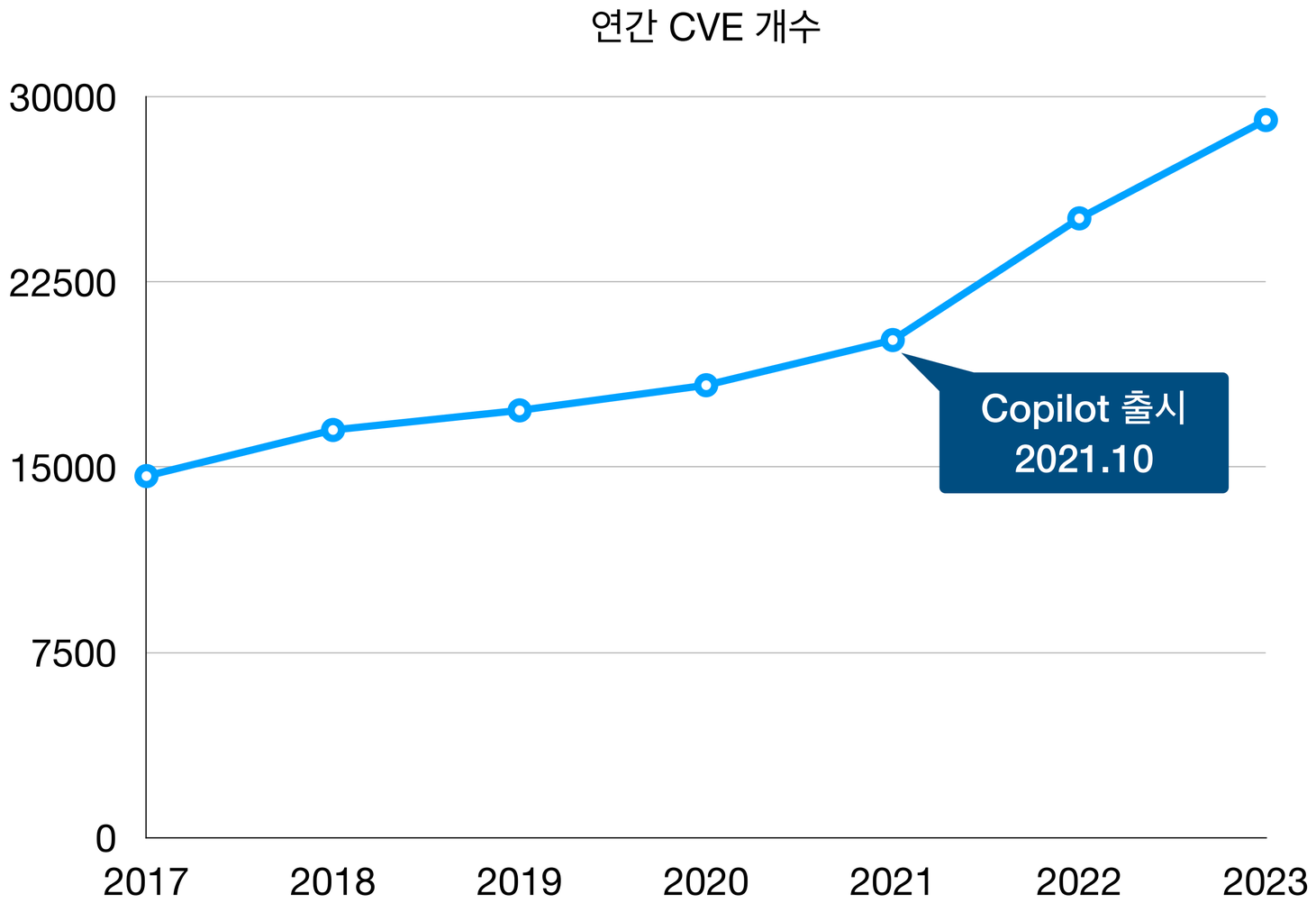
AI 코딩 시대의 SW 오류

AI 코딩 시대의 SW 오류

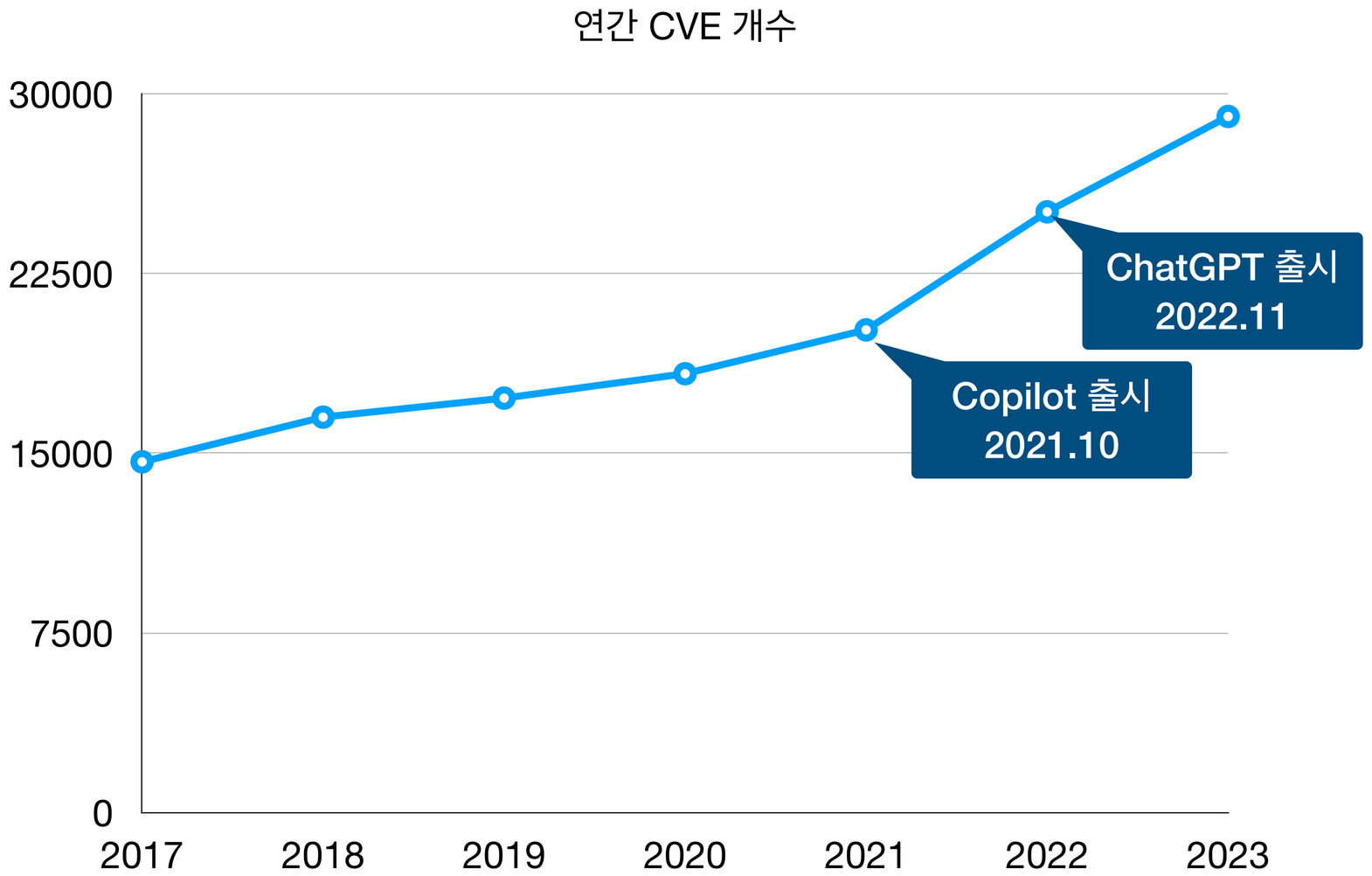
AI 코딩 시대의 SW 오류



AI 코딩 시대의 SW 오류



AI 코딩 시대의 SW 오류



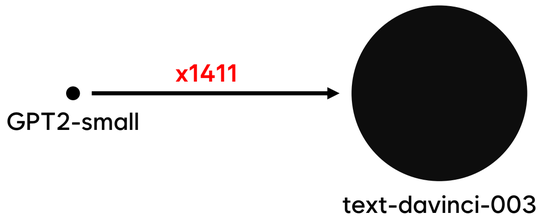
대형 생성 모델의 문제점

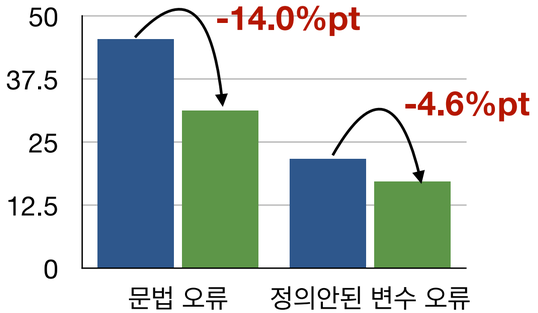
대형 생성 모델의 문제점

- 고비용: 코드 생성이 매우 비쌈
  - AlphaCode 사례: 165개 문제 푸는데 10^{22} FLOPs (5000만원짜리 H100 으로1835시간)

대형 생성 모델의 문제점

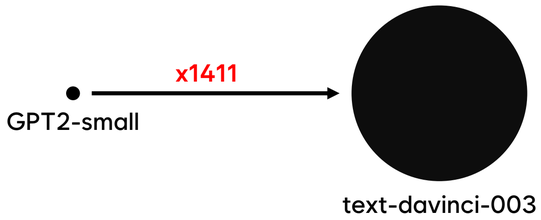
- 고비용: 코드 생성이 매우 비쌈
  - AlphaCode 사례: 165개 문제 푸는데 10^{22} FLOPs (5000만원짜리 H100 으로1835시간)
- 저신뢰: 코드의 품질이 낮음
  - GPT-2-small vs GPT-3

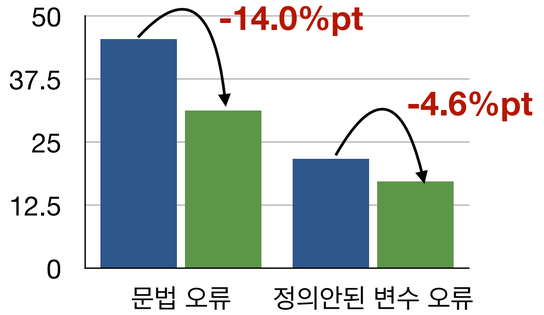




대형 생성 모델의 문제점

- 고비용: 코드 생성이 매우 비쌈
  - AlphaCode 사례: 165개 문제 푸는데 10^{22} FLOPs (5000만원짜리 H100 으로1835시간)
- 저신뢰: 코드의 품질이 낮음
  - GPT-2-small vs GPT-3
- 설명불가: 코드 이해가 어려움
  - AlphCode 사례: 왜 여기에 변수 “g” 가?



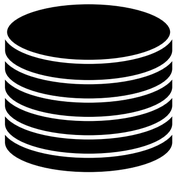



```
#include <bits/stdc++.h>
using namespace std;
const int N = 2e5 + 10;
vector<int> g[N];
int n, d[N], cnt[N];
void dfs(int x, int fa) {
  d[x] = d[fa] + 1;
  for (int i = 0; i < g[x].size(); i++) {
    int v = g[x][i];
    if (v ==
    dfs(v, x)
  }
}
```

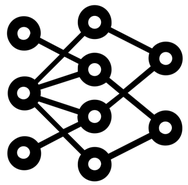
Token	Prob
g	100.0%

SW 오류 재발 방지를 위한 코드 언어 모델

SW 오류 재발 방지를 위한 코드 언어 모델



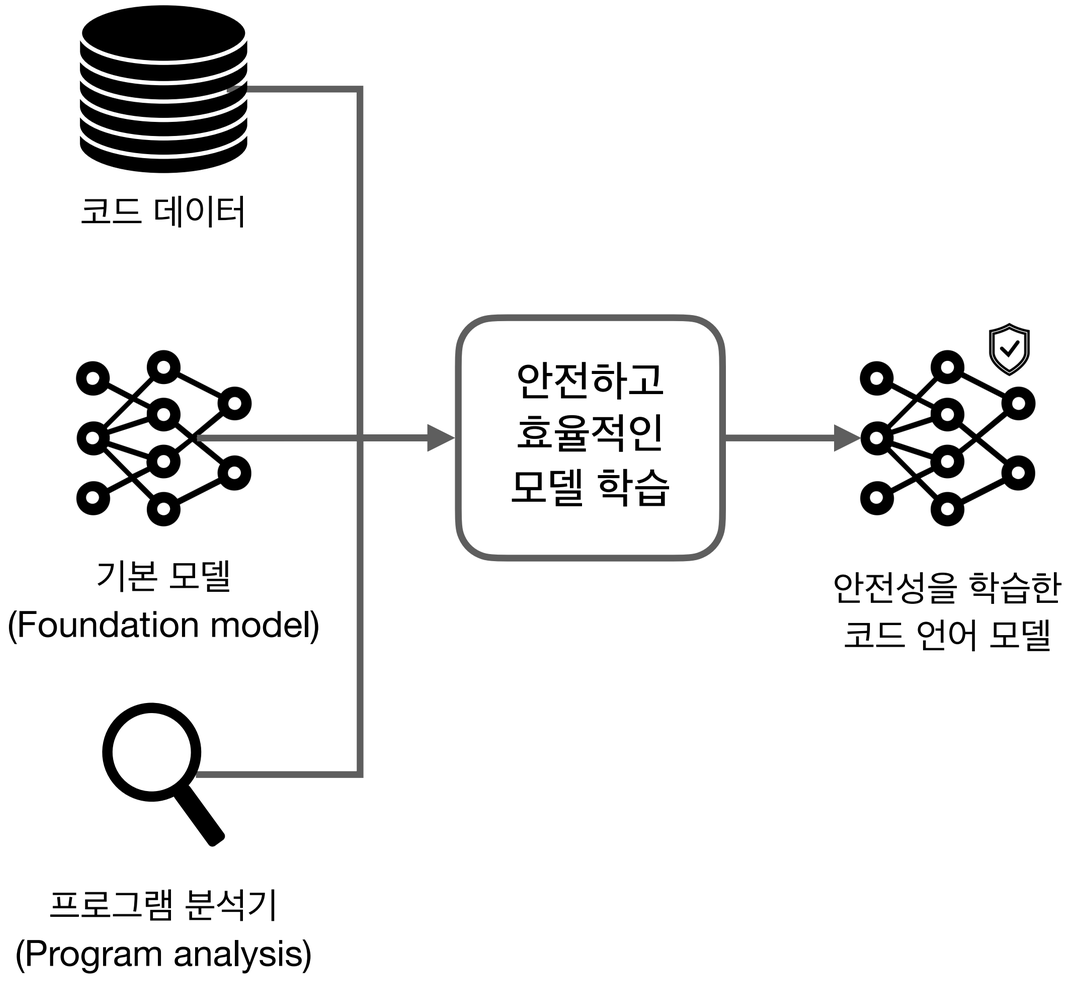
코드 데이터



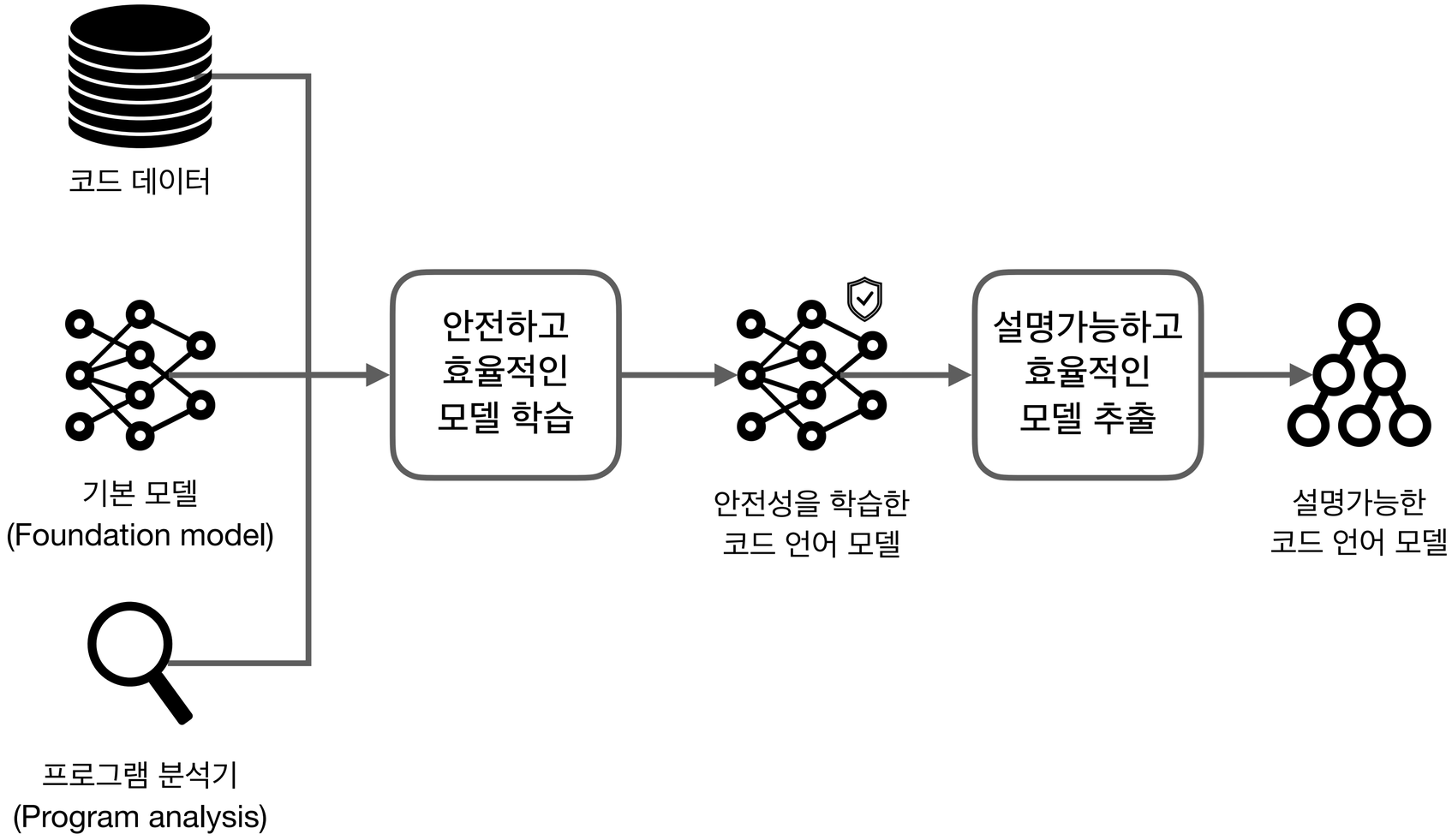
기본 모델
(Foundation model)

프로그램 분석기
(Program analysis)

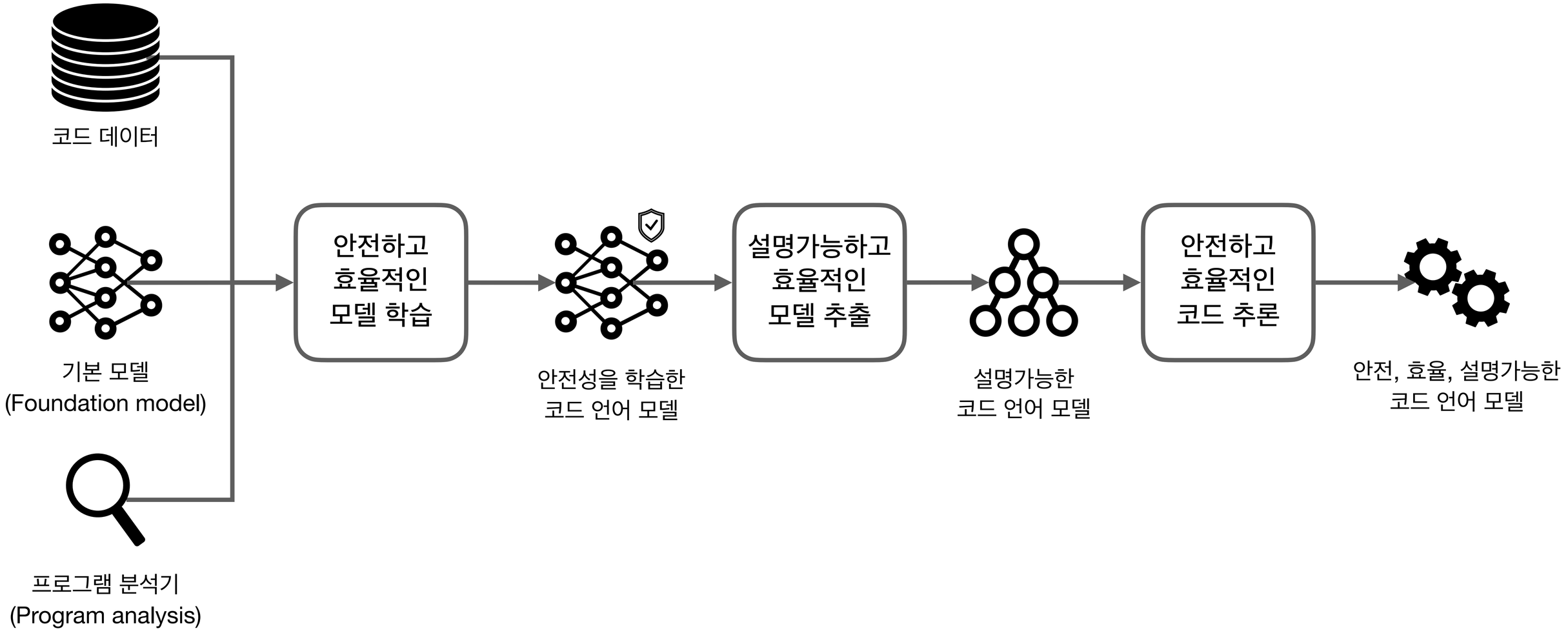
SW 오류 재발 방지를 위한 코드 언어 모델



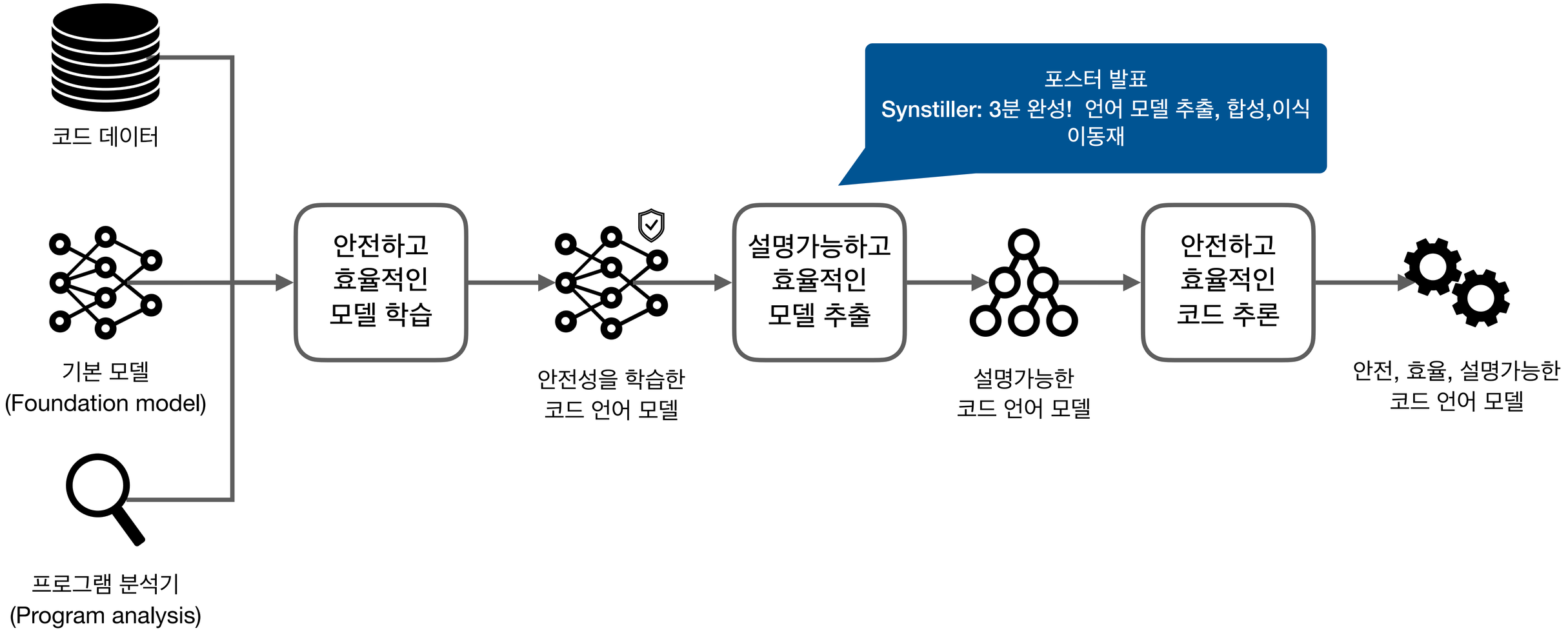
SW 오류 재발 방지를 위한 코드 언어 모델



SW 오류 재발 방지를 위한 코드 언어 모델

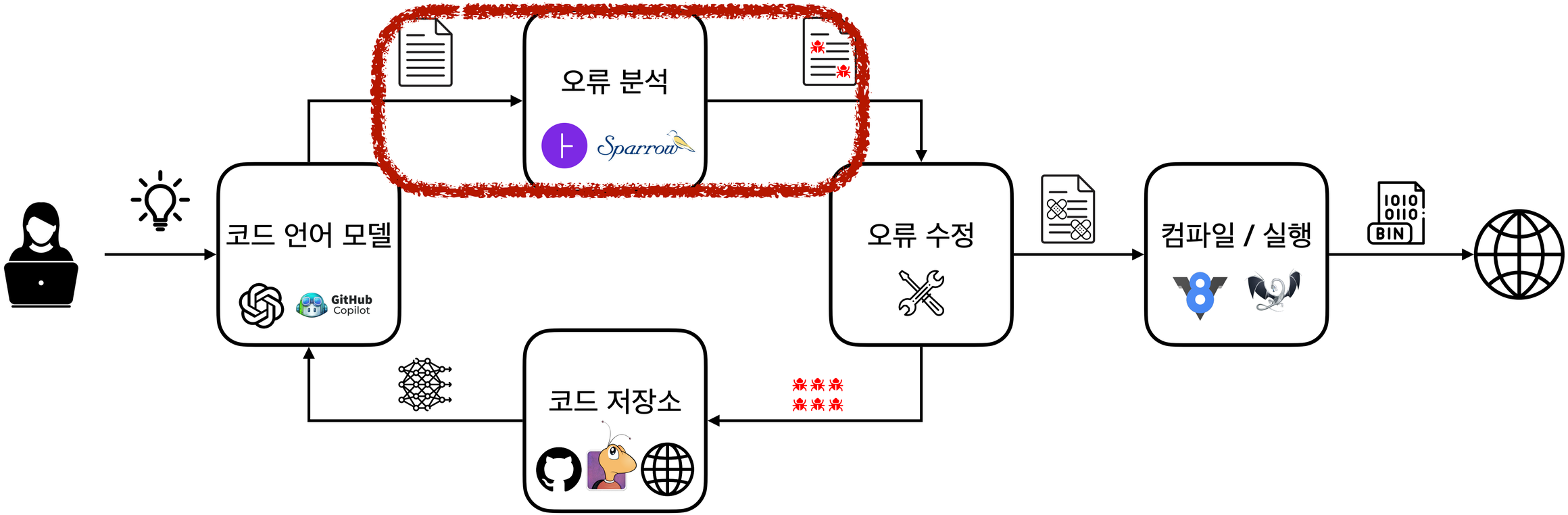


SW 오류 재발 방지를 위한 코드 언어 모델



우리가 꿈꾸는 프로그래밍

- 목표: 손쉽고 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



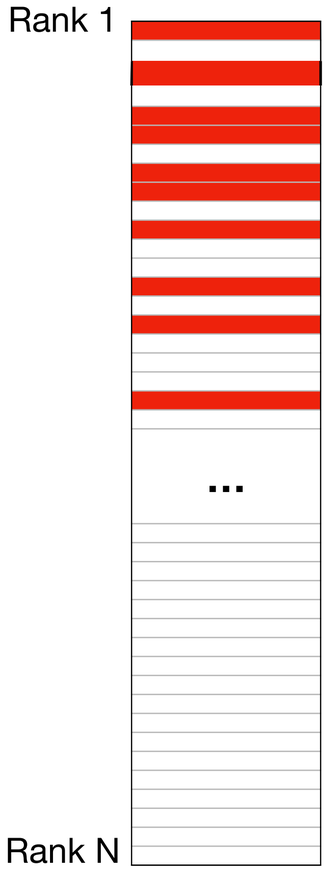
순위 기반 오류 보고서

순위 기반 오류 보고서

- 정적 분석 알람의 점수를 계산, 순위를 매김
  - 예: 특정 커밋과 연관성, 특정 실행과 연관성, 특정 오류와 유사성 등등

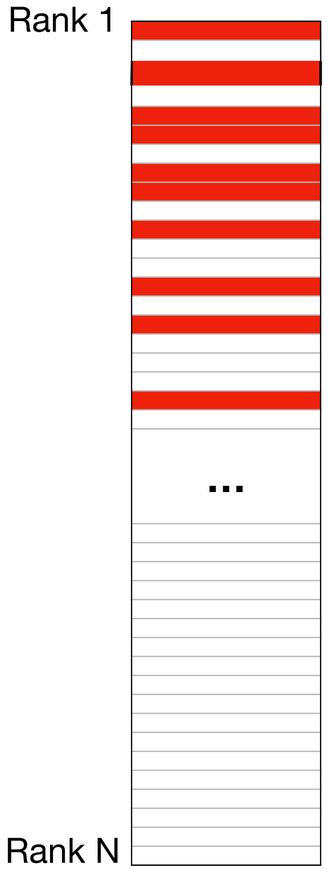
순위 기반 오류 보고서

- 정적 분석 알람의 점수를 계산, 순위를 매김
  - 예: 특정 커밋과 연관성, 특정 실행과 연관성, 특정 오류와 유사성 등등



순위 기반 오류 보고서

- 정적 분석 알람의 점수를 계산, 순위를 매김
  - 예: 특정 커밋과 연관성, 특정 실행과 연관성, 특정 오류와 유사성 등등



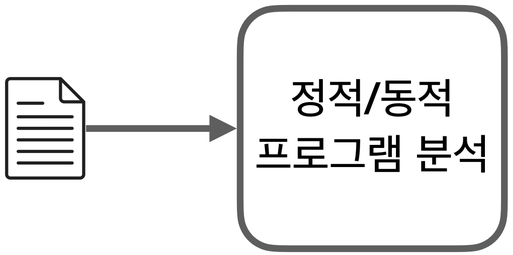
Tracer 성능

Score	Precision
>0.95	87.5%
>0.90	85.7%
>0.85	78.1%
<0.85	37.0%

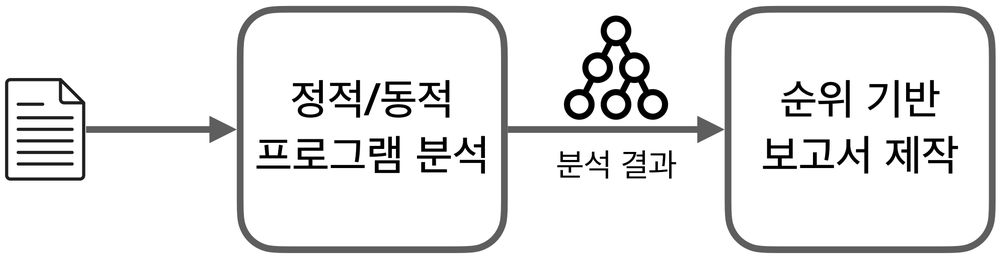
프로그램 분석과 순위 기반 반례 생성

프로그램 분석과 순위 기반 반례 생성

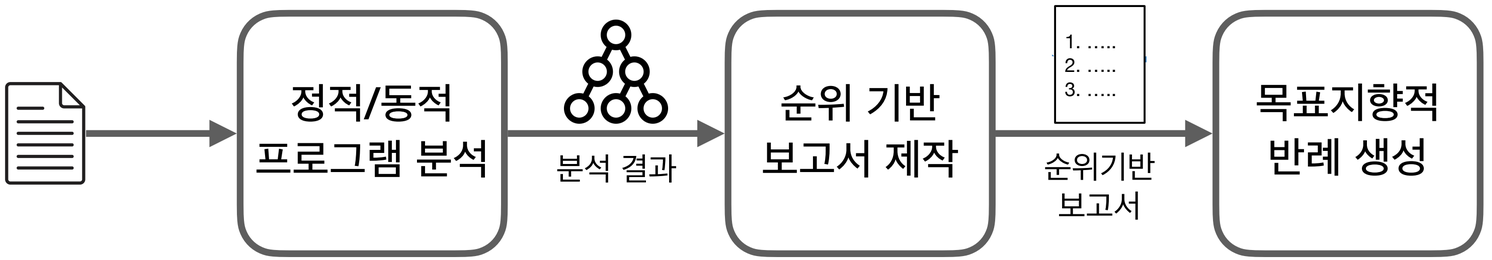
프로그램 분석과 순위 기반 반례 생성



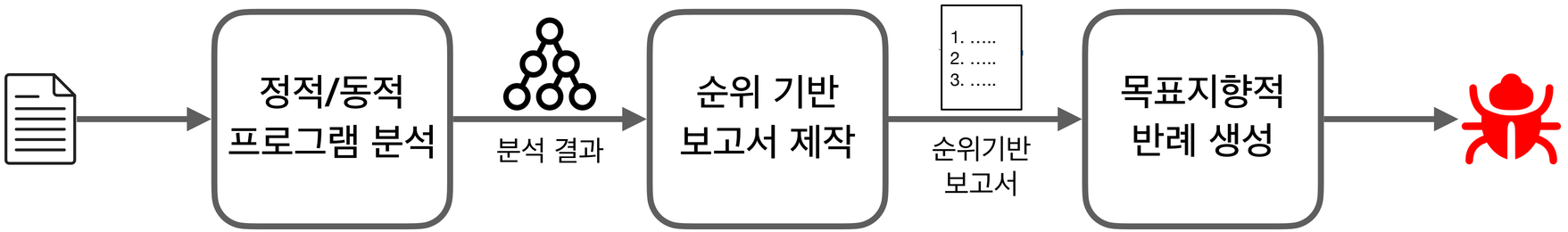
프로그램 분석과 순위 기반 반례 생성



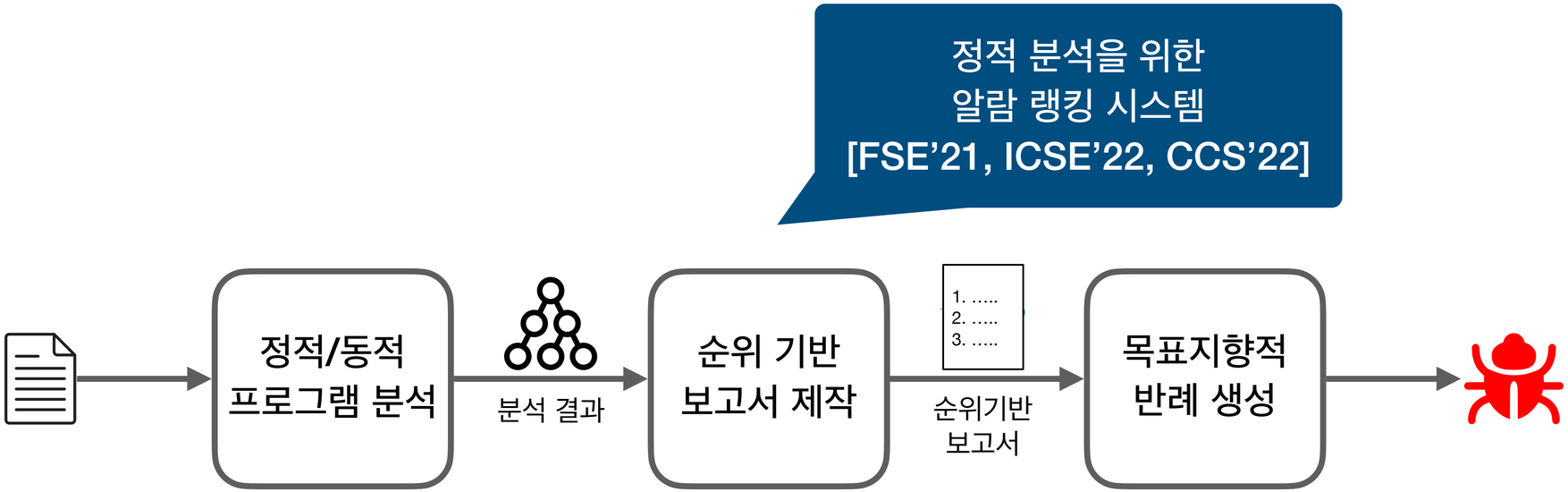
프로그램 분석과 순위 기반 반례 생성



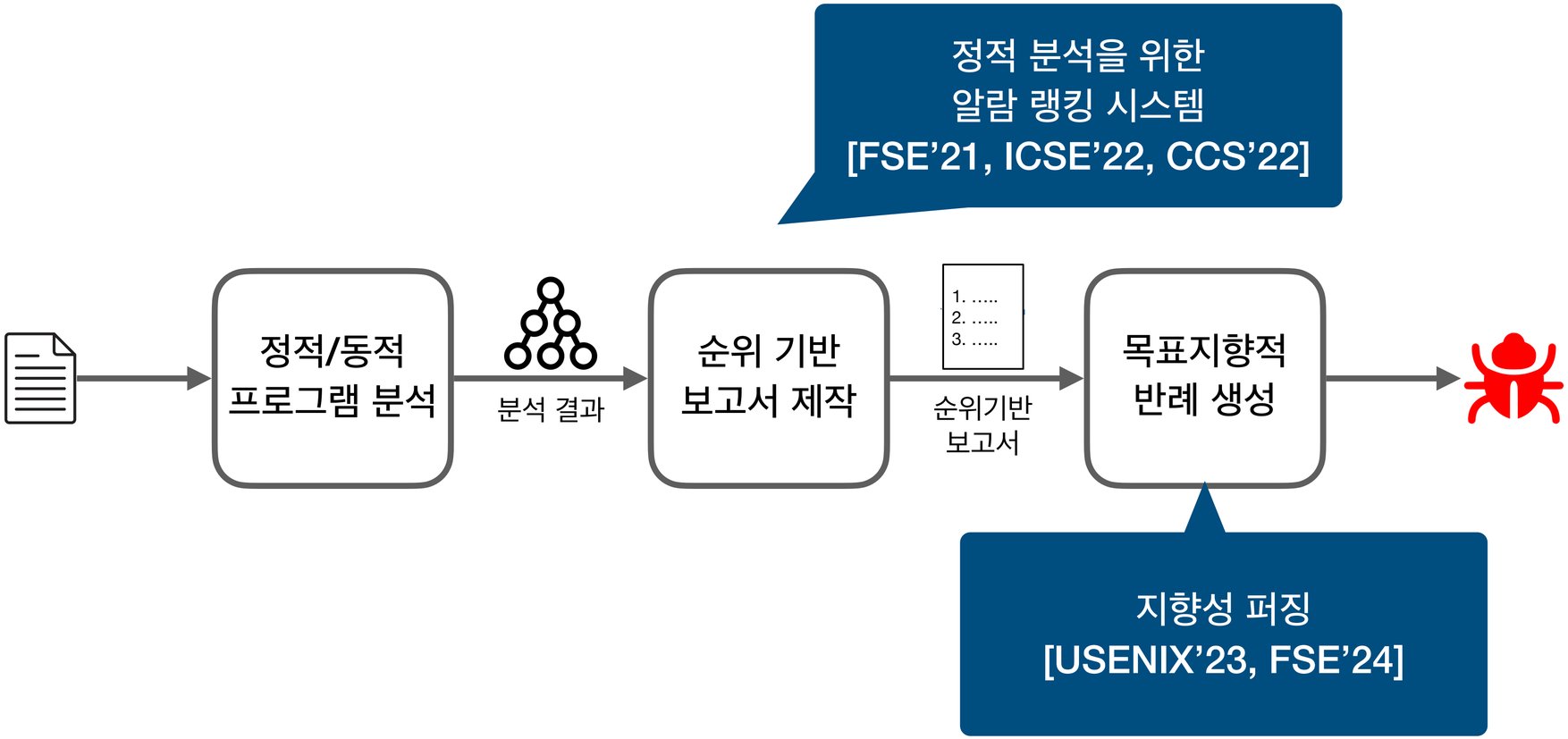
프로그램 분석과 순위 기반 반례 생성



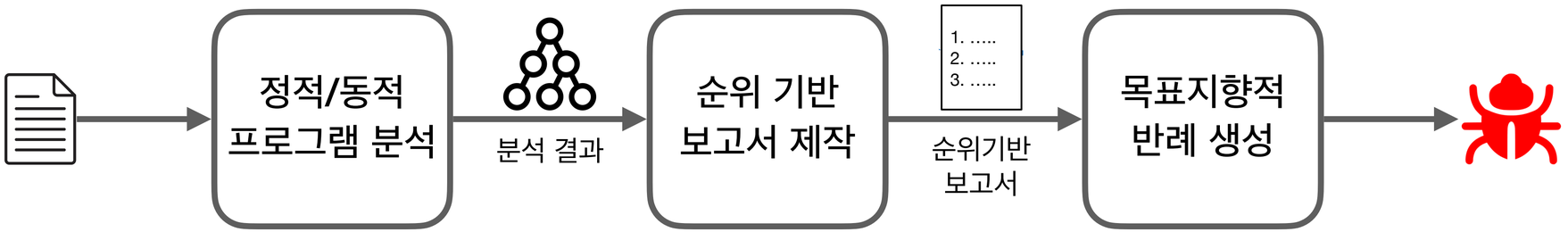
프로그램 분석과 순위 기반 반례 생성



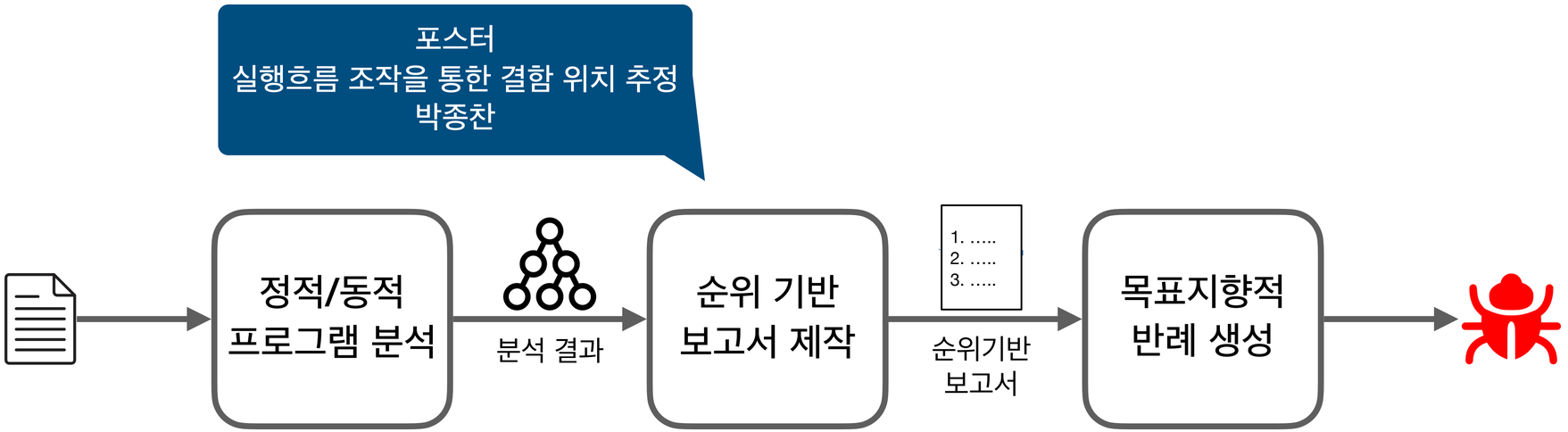
프로그램 분석과 순위 기반 반례 생성



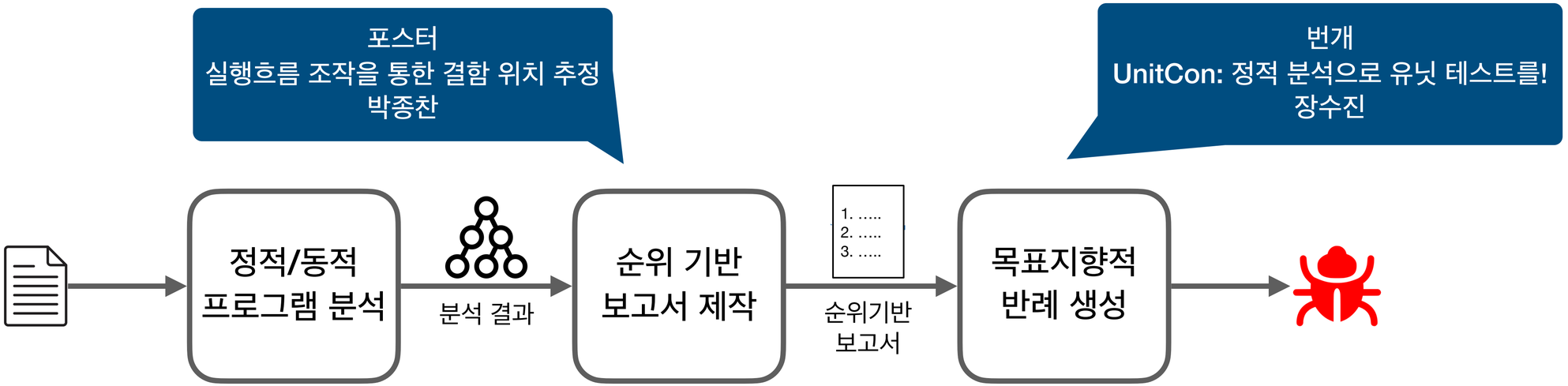
프로그램 분석과 순위 기반 반례 생성



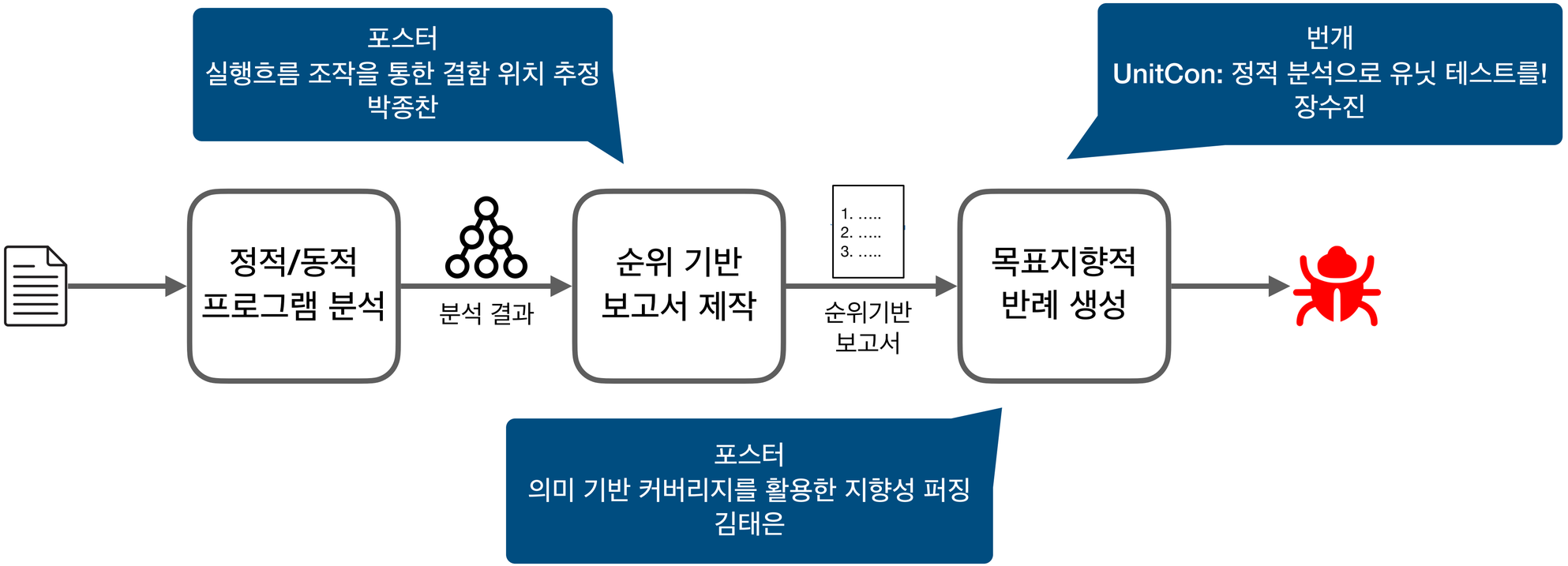
프로그램 분석과 순위 기반 반례 생성



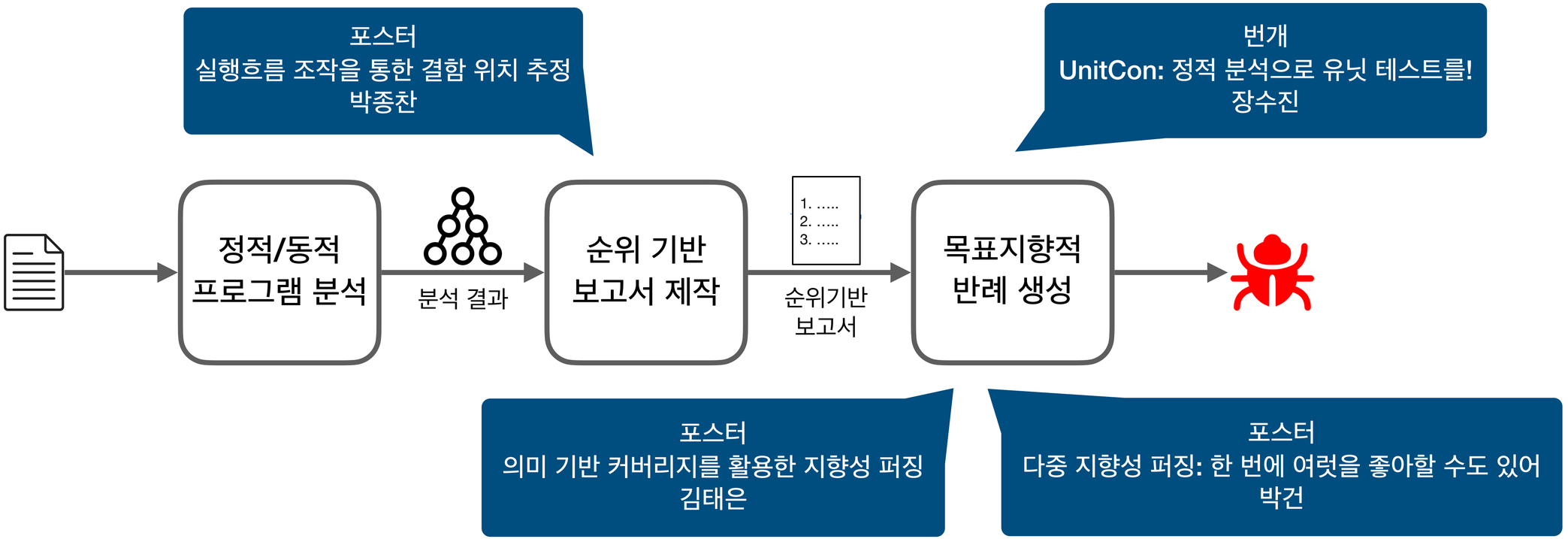
프로그램 분석과 순위 기반 반례 생성



프로그램 분석과 순위 기반 반례 생성

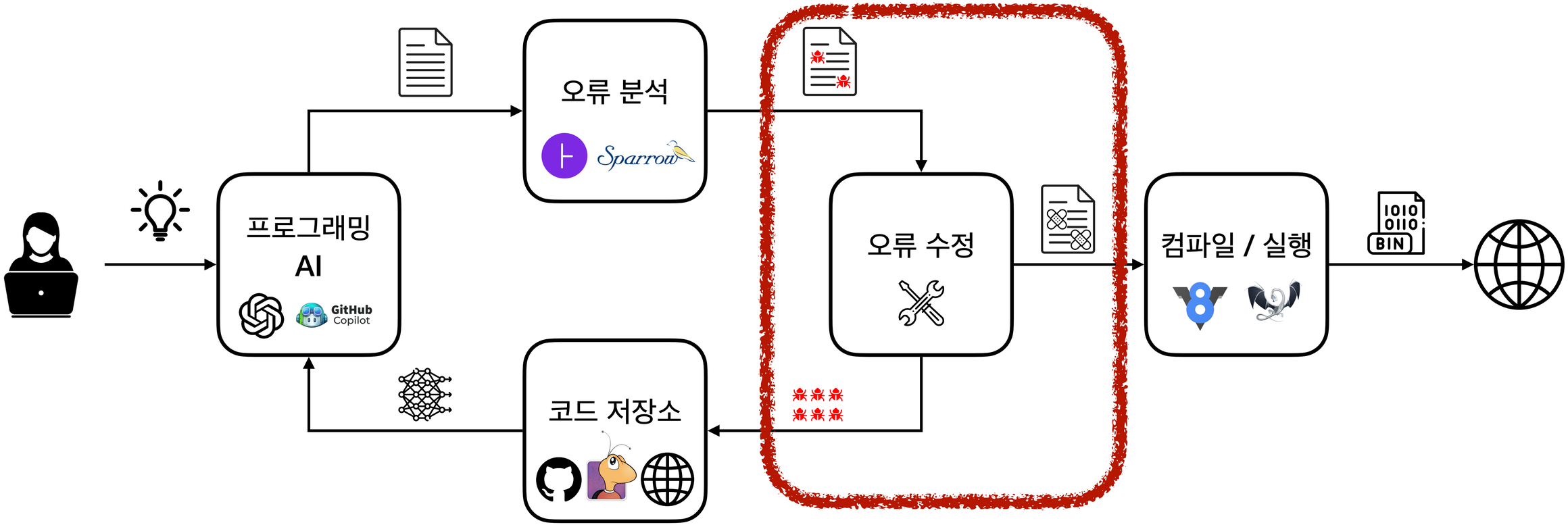


프로그램 분석과 순위 기반 반례 생성



우리가 꿈꾸는 프로그래밍

- 목표: 손쉽고 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



반복되는 오류의 패치?

```
long ToL (char *pbuffer) { return (puffer[0] | puffer[1]<<8 | puffer[2]<<16 | puffer[3]<<24); }

short ToS (char *pbuffer) { return ((short)(puffer[0] | puffer[1]<<8)); }

gint32 ReadBMP (gchar *name, GError **error) {
    if (fread(buffer, Bitmap_File_Head.biSize - 4, fd) != 0)
        FATALP ("BMP: Error reading BMP file header #3");
    Bitmap_Head.biWidth = ToL (&buffer[0x00]);
    Bitmap_Head.biBitCnt = ToS (&buffer[0x0A]);

+    if (Bitmap_Head.biWidth * Bitmap_Head.biBitCnt > G_MAXINT32) {
+        g_set_error(error);
+        return -1;
+    }

    rowbytes = ((Bitmap_Head.biWidth * Bitmap_Head.biBitCnt - 1) / 32) * 4 + 4;
    image_ID = ReadImage (rowbytes);
    ...
}

gint32 ReadImage (int rowbytes) {
    buffer = malloc(rowbytes);      // malloc with overflowed size
    ...
}
```

```
long ToL (char *pbuffer) { return (puffer[0] | puffer[1]<<8 | puffer[2]<<16 | puffer[3]<<24); }

short ToS (char *pbuffer) { return ((short)(puffer[0] | puffer[1]<<8)); }

bitmap_type bmp_load_image (FILE* filename) {
    if (fread(buffer, Bitmap_File_Head.biSize - 4, fd) != 0)
        FATALP ("BMP: Error reading BMP file header #3");
    Bitmap_Head.biWidth = ToL (&buffer[0x00]);
    Bitmap_Head.biBitCnt = ToS (&buffer[0x0A]);

                    ?

    rowbytes = ((Bitmap_Head.biWidth * Bitmap_Head.biBitCnt - 1) / 32) * 4 + 4;
    image.bitmap = ReadImage (rowbytes);
    ...
}

unsigned char* ReadImage (int rowbytes) {
    unsigned char *buffer = (unsigned char*) new char[rowbytes];    // malloc with overflowed size
    ...
}
```

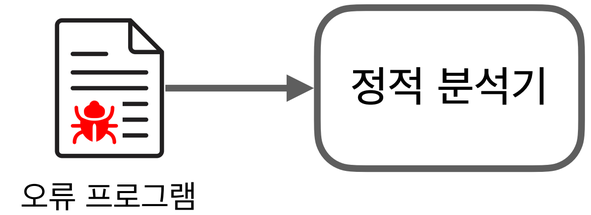
반복되는 오류를 위한 패치 이식

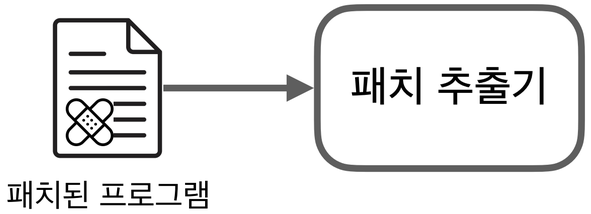
반복되는 오류를 위한 패치 이식

오류 프로그램

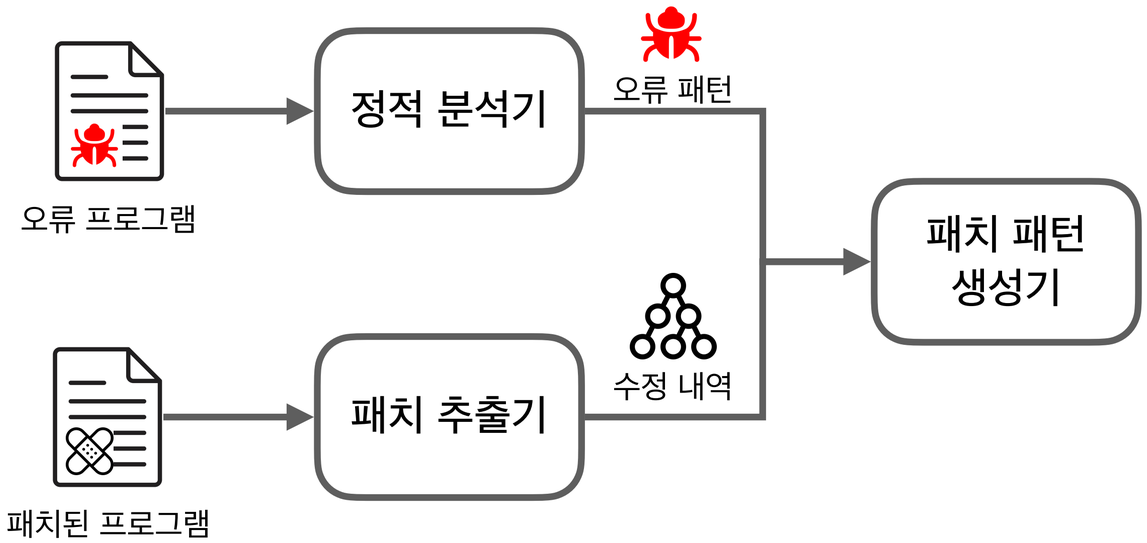
패치된 프로그램

반복되는 오류를 위한 패치 이식

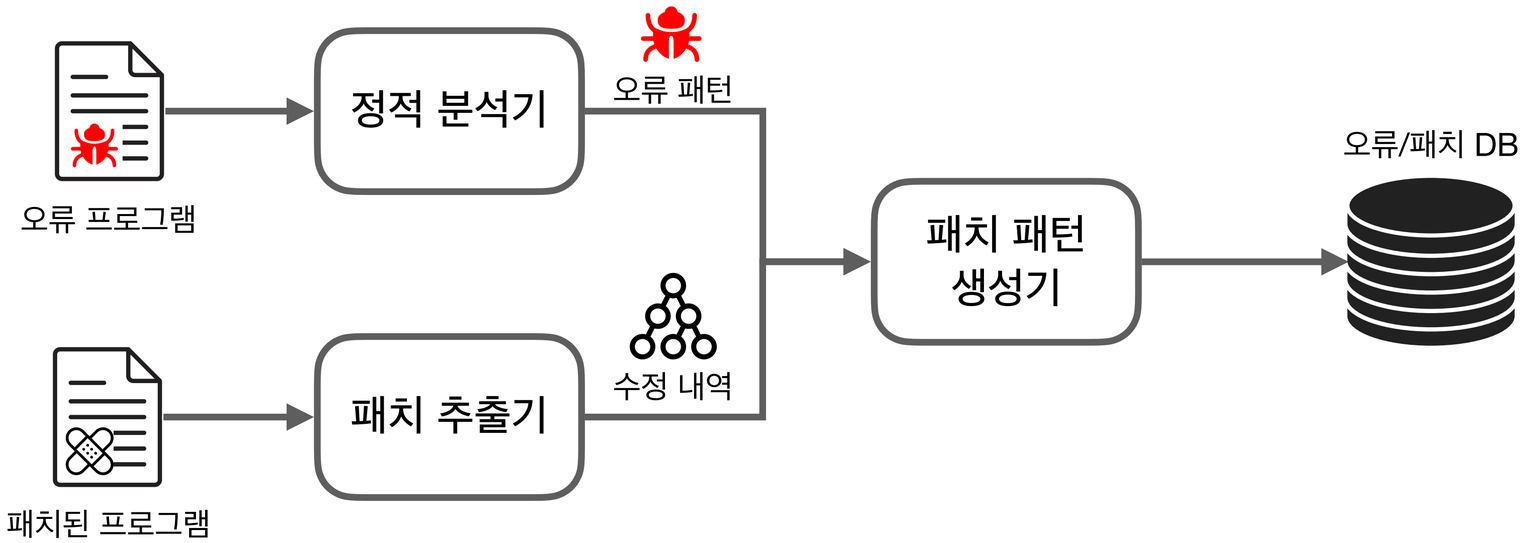




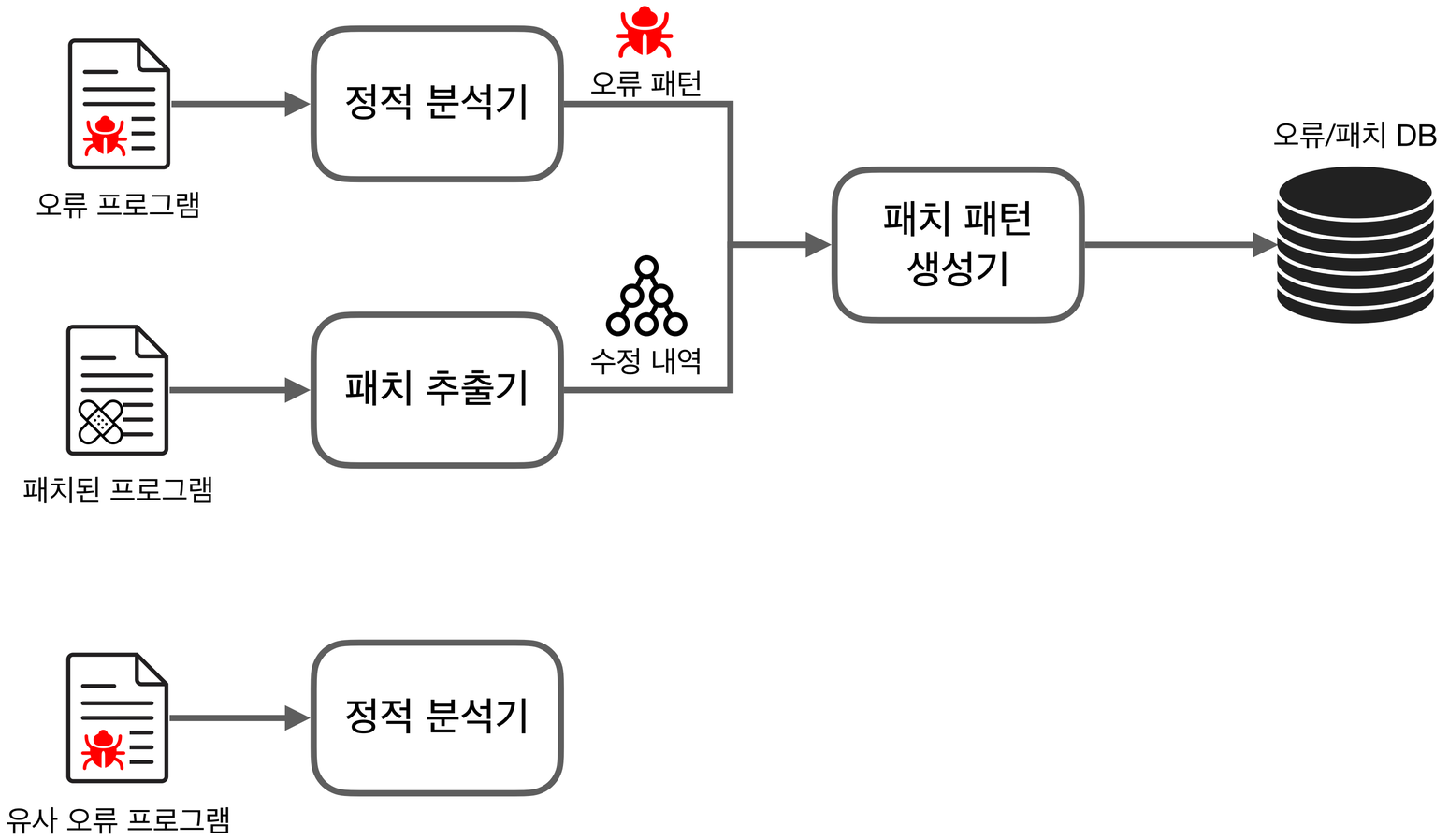
반복되는 오류를 위한 패치 이식



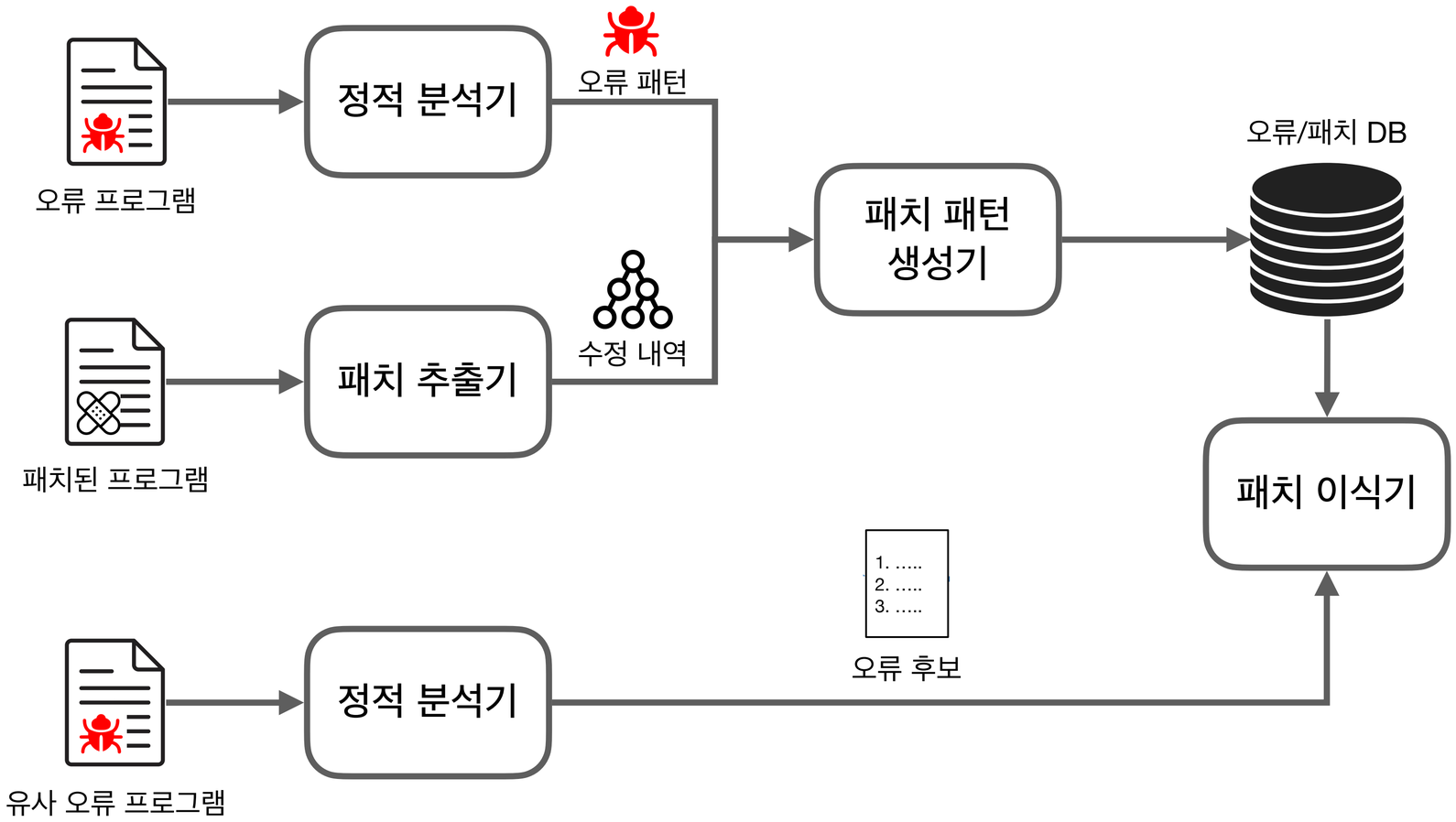
반복되는 오류를 위한 패치 이식



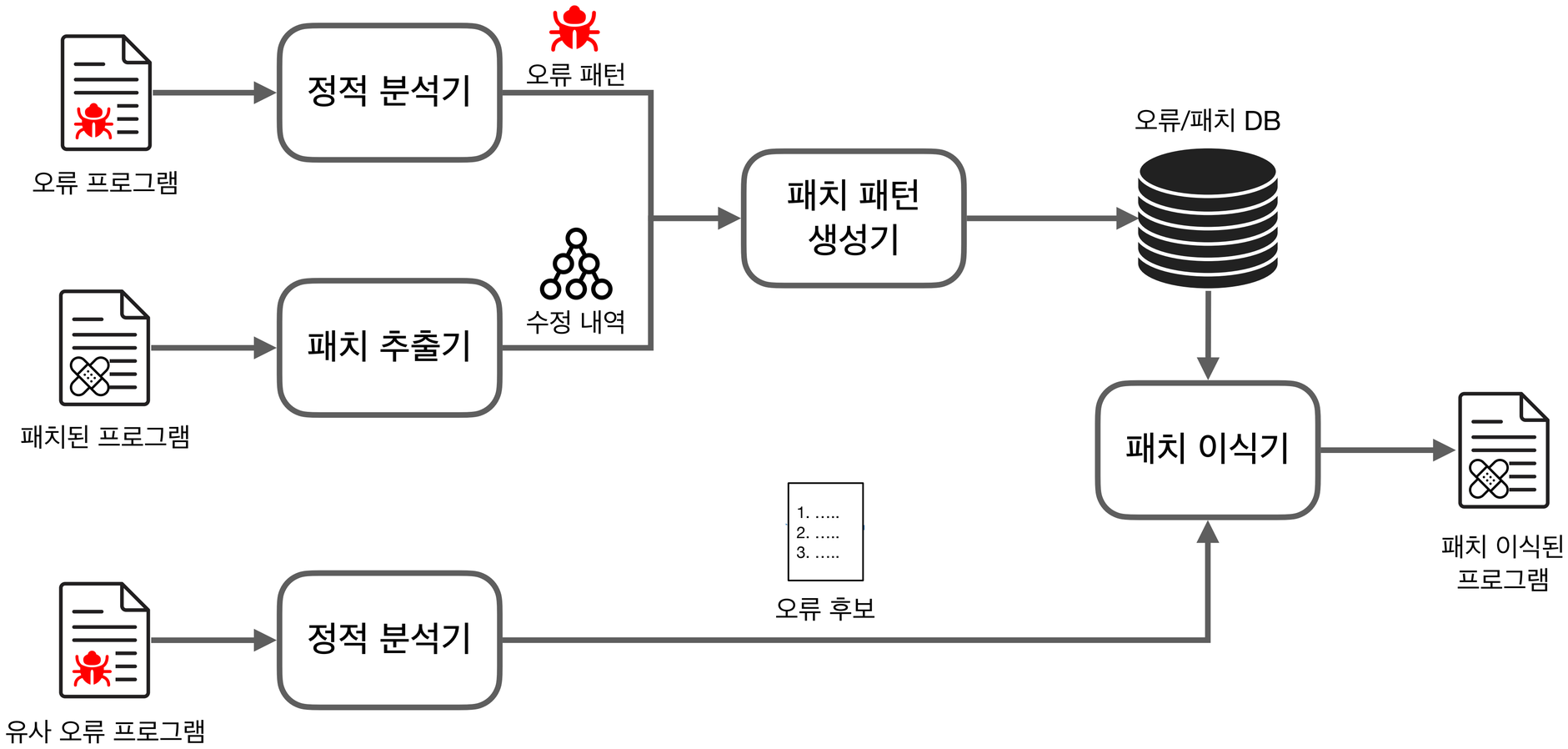
반복되는 오류를 위한 패치 이식



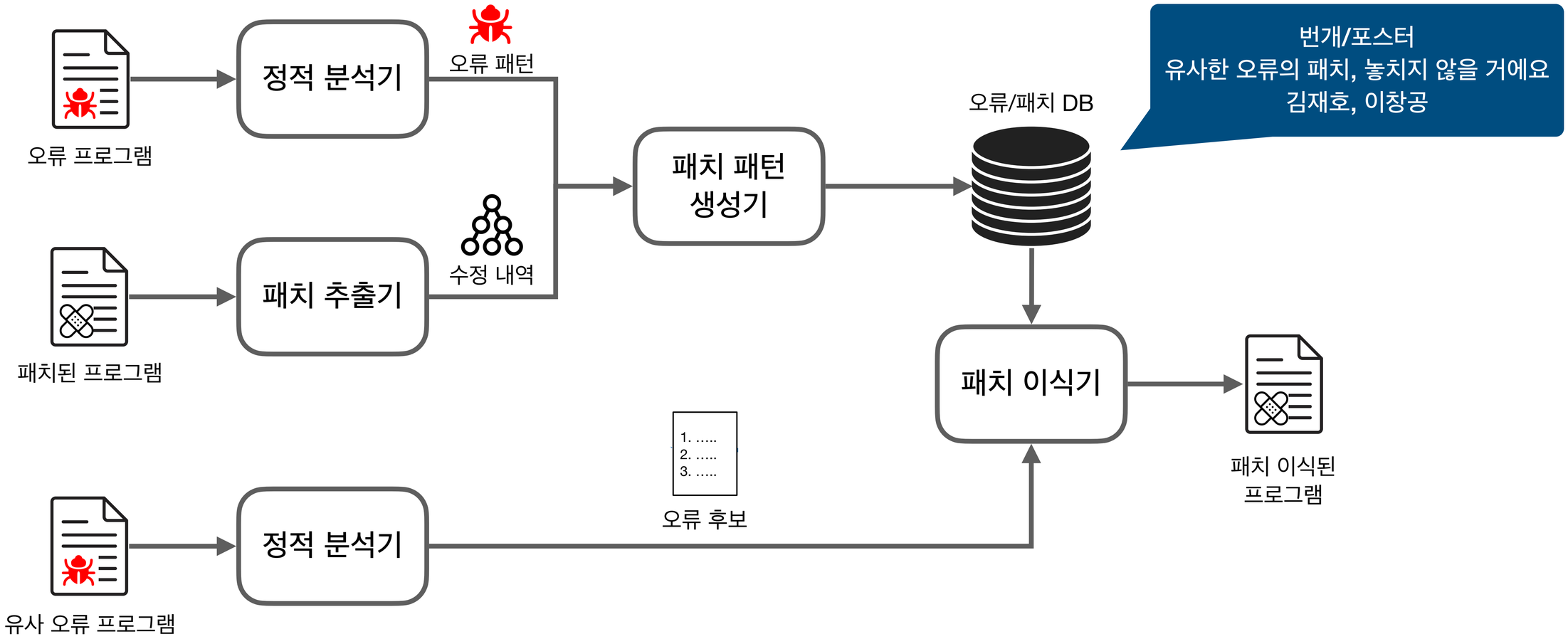
반복되는 오류를 위한 패치 이식



반복되는 오류를 위한 패치 이식

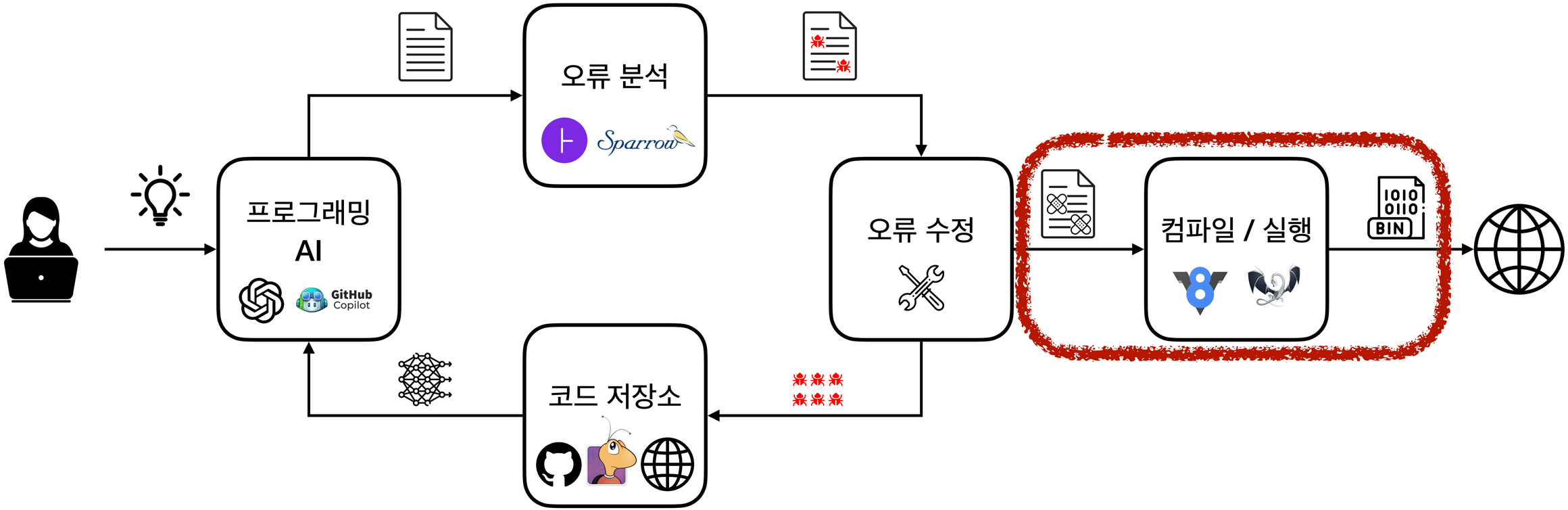


반복되는 오류를 위한 패치 이식



우리가 꿈꾸는 프로그래밍

- 목표: 손쉽고 믿을 수 있는 프로그래밍 시스템 실현
- 기술: 프로그램 분석, 합성, 검증, 학습



컴파일러에서 반복되는 오류

Comment 459 by Git Watcher on Mon, Jan 8, 2024 at 9:12 PM GMT+9 (a day ago) **Project Member**

The following revision refers to this bug:
https://chromium.googlesource.com/v8/v8/+/8baf82c45a5a441bfe834c5448546999c415ffb8

commit 8baf82c45a5a441bfe834c5448546999c415ffb8
Author: Darius M <dmercadier@chromium.org>
Date: Thu Dec 21 15:59:28 2023

[turboshaft][arm64] Add missing ShiftOutZeros covering optimization

This was done by the Turbofan instruction selector, but we forgot this
case in the Turboshaft one.

Chromium Code Reviews

Issue 7604028: Fix three bugs with handling negative zero in the optimizing compiler. (Closed)

Created:
9 years, 4 months ago by fschneider

Modified:
9 years, 4 months ago

Reviewers:
Kevin Millikin (Chromium)

CC:
v8-dev

Base URL:
http://v8.googlecode.com/svn/branches/bleeding_edge/

Visibility:
Public.

Description

```
Fix three bugs with handling negative zero in the optimizing compiler.

* Bug fix for range analysis (contributed by Andy Wingo). Ranges of
double values have to include negative zero. Original code review:
 http://codereview.chromium.org/7514040/

* Fix a bug in optimized Math.round on ARM. When emitting minus-zero checks
we previously return a wrong result because of incorrect register assignment.

* Fix performance problem in IA32 and x64. Refine the checks
for minus zero and avoid unnecessary deoptimizations on Math.floor.

* Improve mjsunit test for Math.round to make sure we also
 get the optimized version of the code for each test case.
Committed: http://code.google.com/p/v8/source/detail?r=8877
```

[RISCV] Fix the neutral element in vector 'fadd' reductions

Closed Public

Authored by **frasercrmck** on Jul 13 2021, 9:22 AM.

Details

Reviewers
- craig.topper
- rogfer01
- HsiangKai
- evandro
- arcbbb

Commits rG03a4702c884a: [RISCV] Fix the neutral element in vector 'fadd' reductions

SUMMARY

Using positive zero as the neutral element in 'fadd' reductions, while it generates better code, is incorrect. The correct neutral element is negative zero: 0.0 + -0.0 = 0.0, whereas -0.0 + -0.0 = -0.0.

There are perhaps more optimal lowerings of negative zero avoiding constant-pool loads which could be left as future work.

컴파일러에서 반복되는 오류

Comment 459 by Git Watcher on Mon, Jan 8, 2024 at 9:12 PM GMT+9 (a day ago) **Project Member**

The following revision refers to this bug:
https://chromium.googlesource.com/v8/v8/+/8baf82c45a5a441bfe834c5448546999c415ffb8

commit 8baf82c45a5a441bfe834c5448546999c415ffb8
Author: Darius M <dmercadier@chromium.org>
Date: Thu Dec 21 15:59:28 2023

[turboshaft][arm64] Add missing ShiftOutZeros covering optimization

This was done by the Turbofan instruction selector, but we forgot this case in the Turboshaft one.

[RISCV] Fix the neutral element in vector 'fadd' reductions

Closed · Public

Authored by **frasercrmck** on Jul 13 2021, 9:22 AM.

Details

Reviewers
- craig.topper
- rogfer01
- HsiangKai
- evandro
- arcbbb

Commits rG03a4702c884a: [RISCV] Fix the neutral element in vector 'fadd' reductions

SUMMARY

Using positive zero as the neutral element in 'fadd' reductions, while it generates better code, is incorrect. The correct neutral element is negative zero: 0.0 + -0.0 = 0.0, whereas -0.0 + -0.0 = -0.0.

There are perhaps more optimal lowerings of negative zero avoiding constant-pool loads which could be left as future work.

Chromium Code Reviews

Issue 7604028: Fix three bugs with handling negative zero in the optimizing compiler. (Closed)

Created:
9 years, 4 months ago by fschneider

Modified:
9 years, 4 months ago

Reviewers:
Kevin Millikin (Chromium)

CC:
v8-dev

Base URL:
http://v8.googlecode.com/svn/branches/bleeding_edge/

Visibility:
Public.

Description

```
Fix three bugs with handling negative zero in the optimizing compiler.

* Bug fix for range analysis (contributed by Andy Wingo). Ranges of
double values have to include negative zero. Original code review:
 http://codereview.chromium.org/7514040/

* Fix a bug in optimized Math.round on ARM. When emitting minus-zero checks
we previously return a wrong result because of incorrect register assignment.

* Fix performance problem in IA32 and x64. Refine the checks
for minus zero and avoid unnecessary deoptimizations on Math.floor.

* Improve mjsunit test for Math.round to make sure we also
 get the optimized version of the code for each test case.
Committed: http://code.google.com/p/v8/source/detail?r=8877
```

컴파일러에서 반복되는 오류

Comment 459 by Git Watcher on Mon, Jan 8, 2024 at 9:12 PM GMT+9 (a day ago) **Project Member**

The following revision refers to this bug:
https://chromium.googlesource.com/v8/v8/+/8baf82c45a5a441bfe834c5448546999c415ffb8

commit 8baf82c45a5a441bfe834c5448546999c415ffb8
Author: Darius M <dmercadier@chromium.org>
Date: Thu Dec 21 15:59:28 2023

[turboshaft][arm64] Add missing ShiftOutZeros covering optimization

This was done by the Turbofan instruction selector, but we forgot this case in the Turboshaft one.

[RISCV] Fix the neutral element in vector 'fadd' reductions

Closed Public

Authored by **frasercrmck** on Jul 13 2021, 9:22 AM.

Details

Reviewers
- craig.topper
- rogfer01
- HsiangKai
- evandro
- arcbbb

Commits rG03a4702c884a: [RISCV] Fix the neutral element in vector 'fadd' reductions

SUMMARY

Using positive zero as the neutral element in 'fadd' reductions, while it generates better code, is incorrect. The correct neutral element is negative zero: 0.0 + -0.0 = 0.0, whereas -0.0 + -0.0 = -0.0.

There are perhaps more optimal lowerings of negative zero avoiding constant-pool loads which could be left as future work.

Chromium Code Reviews

Issue 7604028: Fix three bugs with handling negative zero in the optimizing compiler. (Closed)

Created:
9 years, 4 months ago by fschneider

Modified:
9 years, 4 months ago

Reviewers:
Kevin Millikin (Chromium)

CC:
v8-dev

Base URL:
http://v8.googlecode.com/svn/branches/bleeding_edge/

Visibility:
Public.

Description

```
Fix three bugs with handling negative zero in the optimizing compiler.

* Bug fix for range analysis (contributed by Andy Wingo). Ranges of
double values have to include negative zero. Original code review:
 http://codereview.chromium.org/7514040/

* Fix a bug in optimized Math.round on ARM. When emitting minus-zero checks
we previously return a wrong result because of incorrect register assignment.

* Fix performance problem in IA32 and x64. Refine the checks
for minus zero and avoid unnecessary deoptimizations on Math.floor.

* Improve mjsunit test for Math.round to make sure we also
 get the optimized version of the code for each test case.
Committed: http://code.google.com/p/v8/source/detail?r=8877
```

컴파일러에서 반복되는 오류

Comment 459 by Git Watcher on Mon, Jan 8, 2024 at 9:12 PM GMT+9 (a day ago) **Project Member**

The following revision refers to this bug:
https://chromium.googlesource.com/v8/v8/+/8baf82c45a5a441bfe834c5448546999c415ffb8

commit 8baf82c45a5a441bfe834c5448546999c415ffb8
Author: Darius M <dmercadier@chromium.org>
Date: Thu Dec 21 15:59:28 2023

[turboshaft][arm64] Add missing ShiftOutZeros covering optimization

This was done by the Turbofan instruction selector, but we forgot this case in the Turboshaft one.

Chromium Code Reviews

Issue 7604028: Fix three bugs with handling negative zero in the optimizing compiler. (Closed)

Created:
9 years, 4 months ago by fschneider

Modified:
9 years, 4 months ago

Reviewers:
Kevin Millikin (Chromium)

CC:
v8-dev

Base URL:
http://v8.googlecode.com/svn/branches/bleeding_edge/

Visibility:
Public.

Description

```
Fix three bugs with handling negative zero in the optimizing compiler.

* Bug fix for range analysis (contributed by Andy Wingo). Ranges of
double values have to include negative zero. Original code review:
 http://codereview.chromium.org/7514040/

* Fix a bug in optimized Math.round on ARM. When emitting minus-zero checks
we previously return a wrong result because of incorrect register assignment.

* Fix performance problem in IA32 and x64. Refine the checks
for minus zero and avoid unnecessary deoptimizations on Math.floor.

* Improve mjsunit test for Math.round to make sure we also
 get the optimized version of the code for each test case.
Committed: http://code.google.com/p/v8/source/detail?r=8877
```

[RISCV] Fix the neutral element in vector 'fadd' reductions

Closed Public

Authored by frasercrmck on Jul 13 2021, 9:22 AM.

Details

Reviewers craig.topper
rogfer01
HsiangKai
evandro
arcbbb

Commits rG03a4702c884a: [RISCV] Fix the neutral element in vector 'fadd' reductions

SUMMARY

Using positive zero as the neutral element in 'fadd' reductions, while it generates better code, is incorrect. The correct neutral element is negative zero: 0.0 + -0.0 = 0.0, whereas -0.0 + -0.0 = -0.0.

There are perhaps more optimal lowerings of negative zero avoiding constant-pool loads which could be left as future work.

반복되는 컴파일러 최적화 오류

반복되는 컴파일러 최적화 오류

- 반복되는 이유: 비슷한 코드를 짜면서 비슷한 실수를 하기 때문
  - 예: 산술 최적화, IEEE-754 부동소수점 표준

반복되는 컴파일러 최적화 오류

- 반복되는 이유: 비슷한 코드를 짜면서 비슷한 실수를 하기 때문
  - 예: 산술 최적화, IEEE-754 부동소수점 표준
- 미연에 방지하는 방법
  - 비슷한 실수 방지: 컴파일러의 최적화를 엄밀하게 검사
  - 비슷한 코드 작성 방지: 컴파일러의 최적화 코드를 자동으로 생성

컴파일러 올바름 검사

컴파일러 올바름 검사

	컴파일러 검증	컴파일러 검산 (번역 검산)

컴파일러 올바름 검사

	컴파일러 검증	컴파일러 검산 (번역 검산)
접근법	컴파일러 코드를 엄밀히 검증	특정 컴파일의 올바름 검사

컴파일러 올바름 검사

	컴파일러 검증	컴파일러 검산 (번역 검산)
접근법	컴파일러 코드를 엄밀히 검증	특정 컴파일의 올바름 검사
장점	구현 올바름 전면 보장	구현과 무관, 언어 의미에 집중

컴파일러 올바름 검사

	컴파일러 검증	컴파일러 검산 (번역 검산)
접근법	컴파일러 코드를 엄밀히 검증	특정 컴파일의 올바름 검사
장점	구현 올바름 전면 보장	구현과 무관, 언어 의미에 집중
단점	큰 코드, 불명확 명세, 잦은 변경에 취약	관찰한 것만 올바름 보장

컴파일러 올바름 검사

	컴파일러 검증	컴파일러 검산 (번역 검산)
접근법	컴파일러 코드를 엄밀히 검증	특정 컴파일의 올바름 검사
장점	구현 올바름 전면 보장	구현과 무관, 언어 의미에 집중
단점	큰 코드, 불명확 명세, 잦은 변경에 취약	관찰한 것만 올바름 보장
성공사례	CompCert	Alive2

컴파일러 올바름 검사

	컴파일러 검증	컴파일러 검산 (번역 검산)
접근법	컴파일러 코드를 엄밀히 검증	특정 컴파일의 올바름 검사
장점	구현 올바름 전면 보장	구현과 무관, 언어 의미에 집중
단점	큰 코드, 불명확 명세, 잦은 변경에 취약	관찰한 것만 올바름 보장
성공사례	CompCert	Alive2

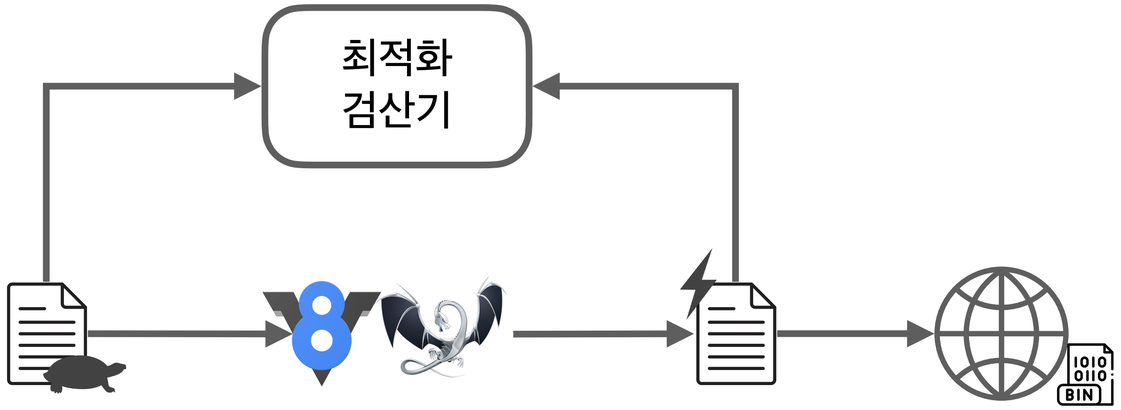
안전한 컴파일러 최적화를 위한 검산과 이식

안전한 컴파일러 최적화를 위한 검산과 이식

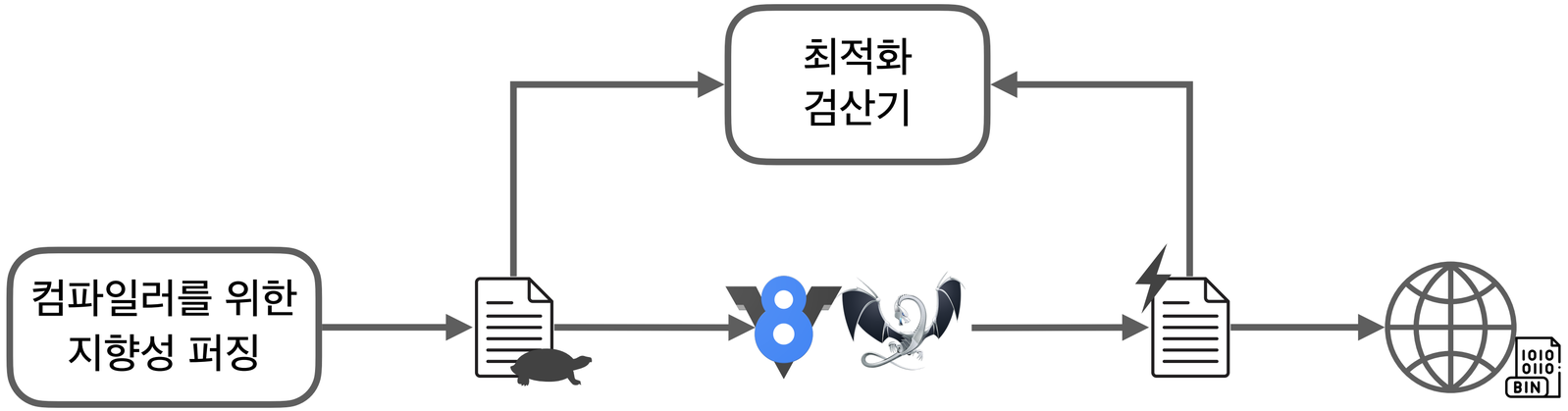
안전한 컴파일러 최적화를 위한 검산과 이식

안전한 컴파일러 최적화를 위한 검산과 이식

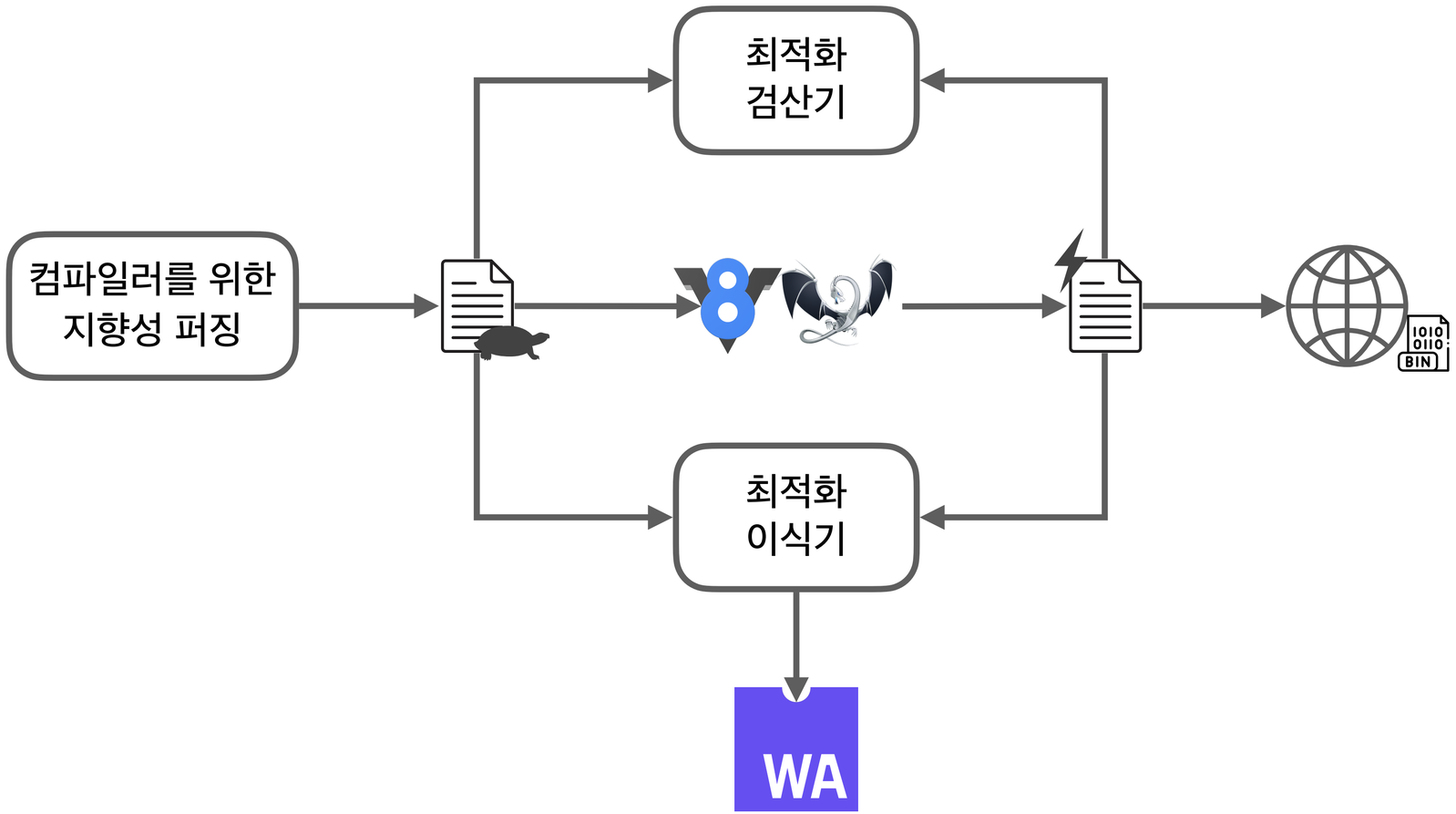
안전한 컴파일러 최적화를 위한 검산과 이식



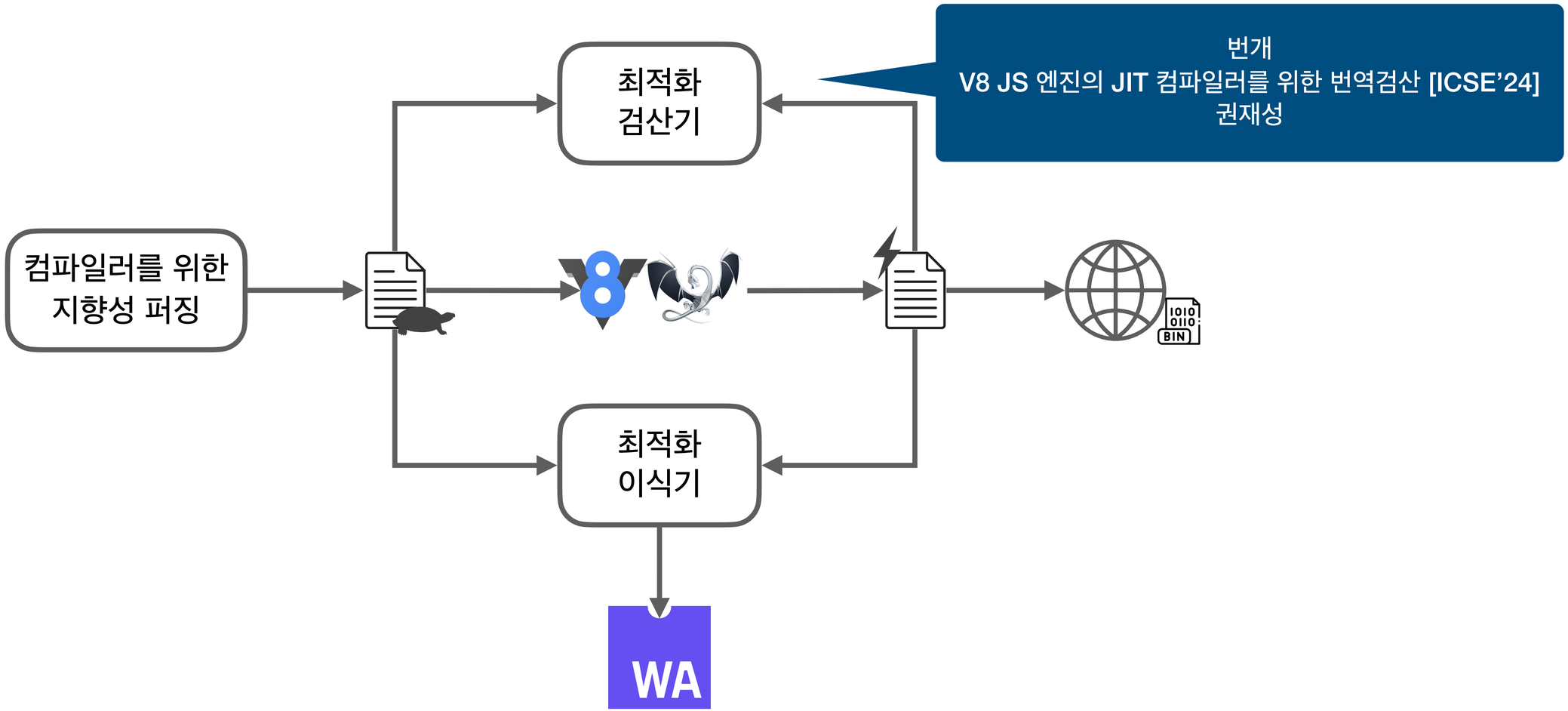
안전한 컴파일러 최적화를 위한 검산과 이식



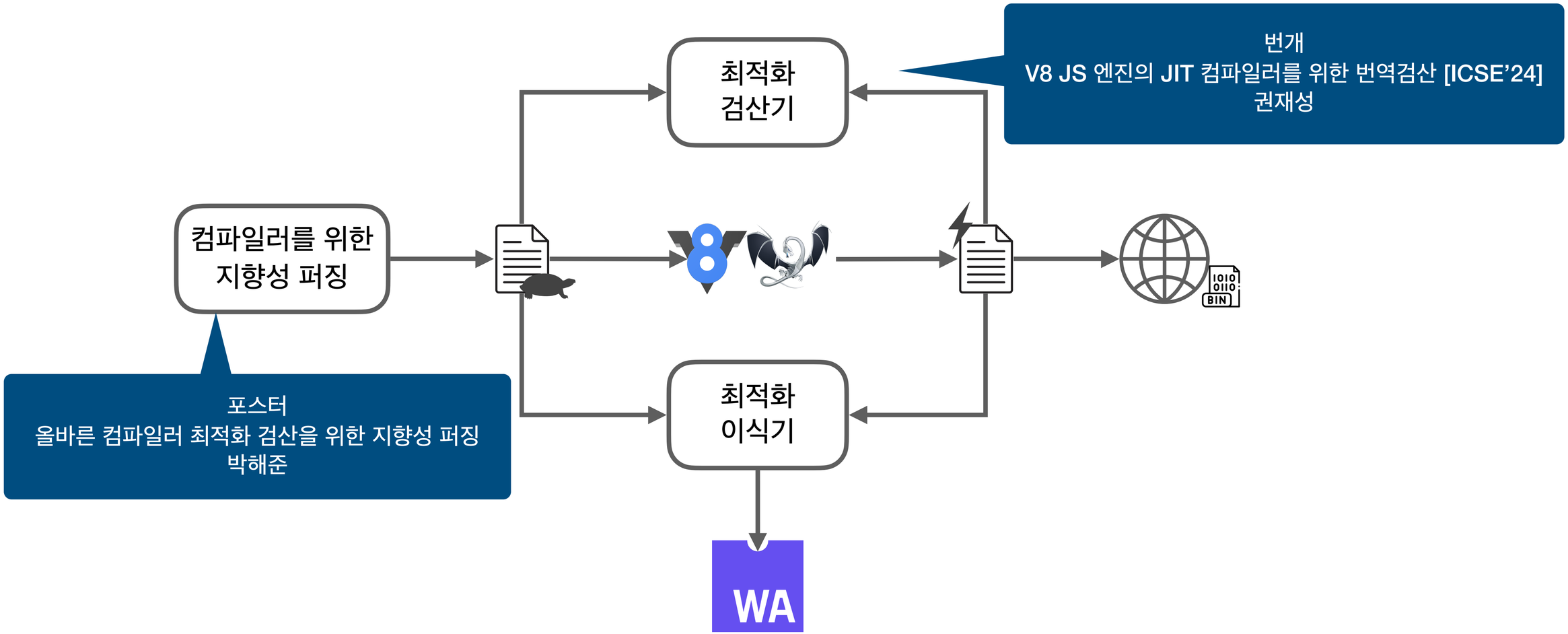
안전한 컴파일러 최적화를 위한 검산과 이식



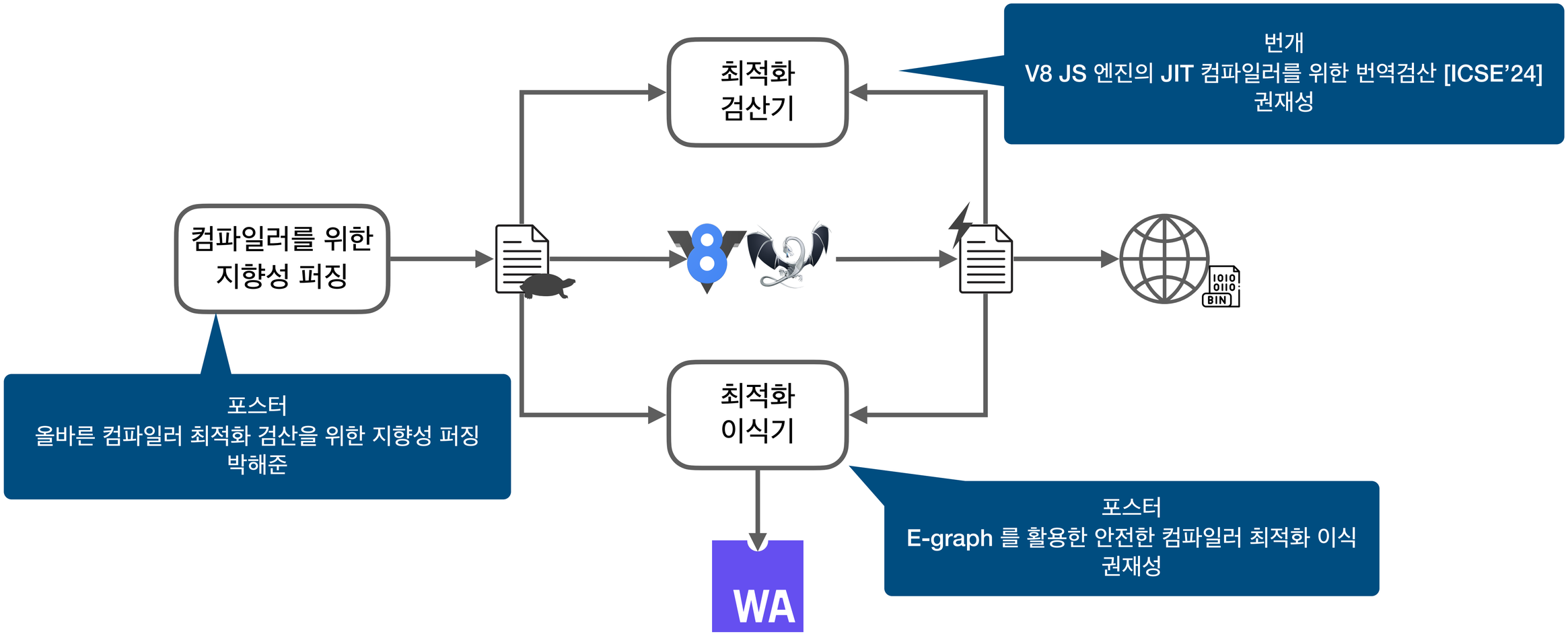
안전한 컴파일러 최적화를 위한 검산과 이식



안전한 컴파일러 최적화를 위한 검산과 이식

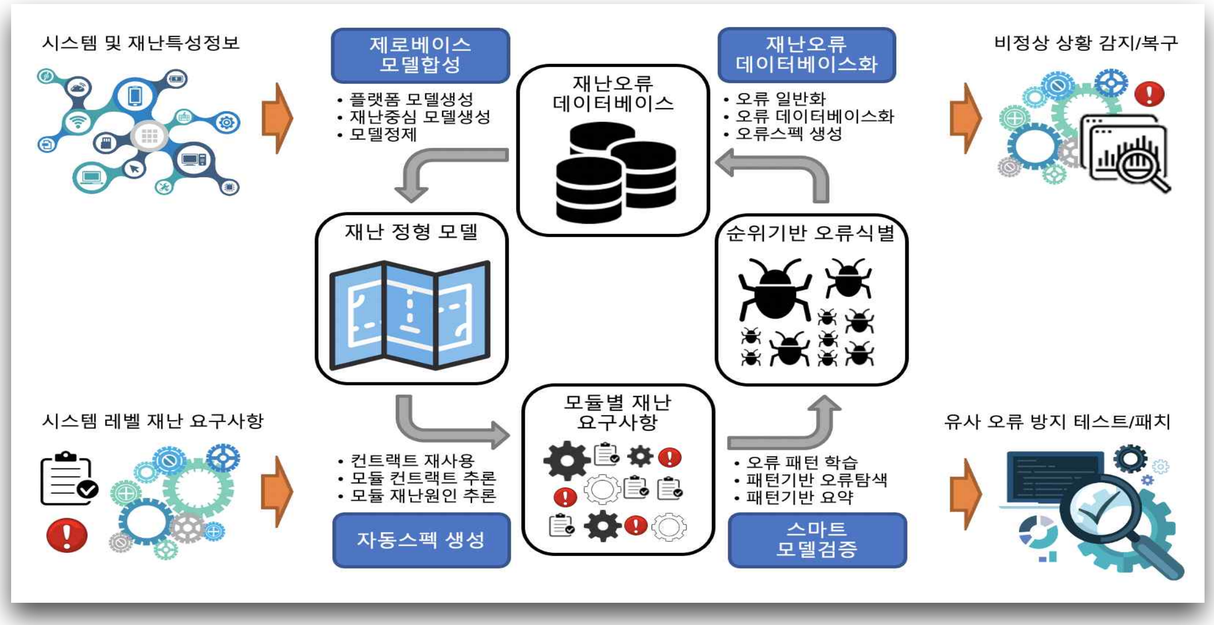


안전한 컴파일러 최적화를 위한 검산과 이식

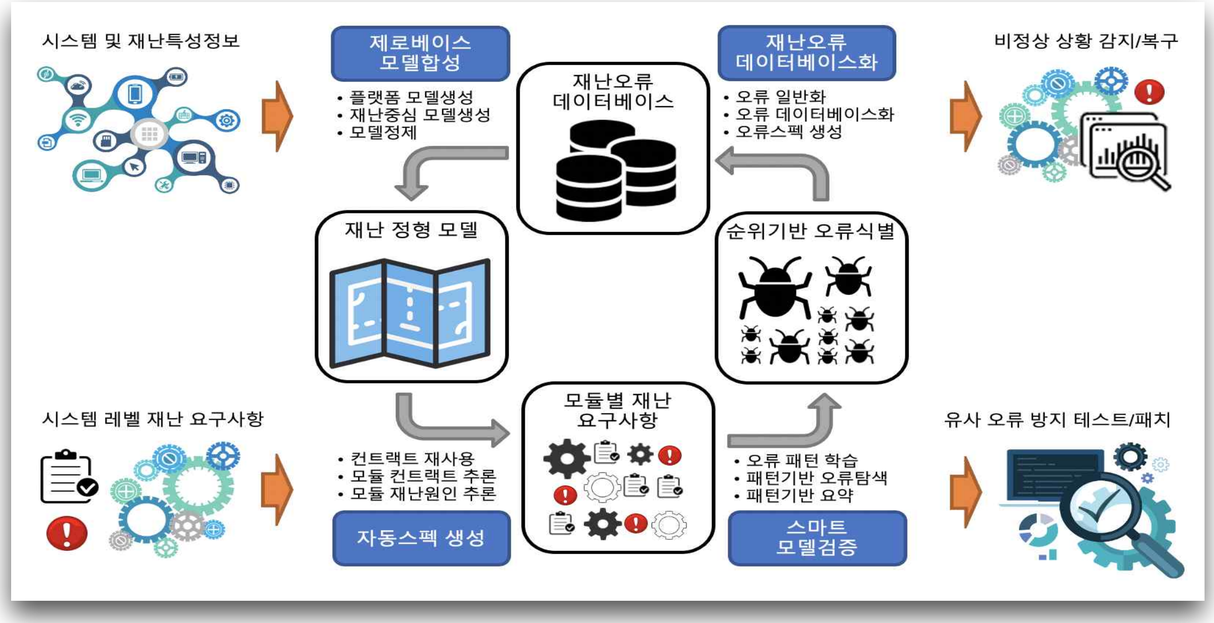


마무리

마무리


시스템 및 재난특성정보
제로베이스
모델합성
• 플랫폼 모델생성
• 재난중심 모델생성
• 모델정제
재난오류
데이터베이스
재난오류
데이터베이스화
• 오류 일반화
• 오류 데이터베이스화
• 오류스펙 생성
비정상 상황 감지/복구
재난 정형 모델
순위기반 오류식별
시스템 레벨 재난 요구사항
• 컨트랙트 재사용
• 모듈 컨트랙트 추론
• 모듈 재난원인 추론
자동스펙 생성
모듈별 재난
요구사항
• 오류 패턴 학습
• 패턴기반 오류탐색
• 패턴기반 요약
스마트
모델검증
유사 오류 방지 테스트/패치

마무리


시스템 및 재난특성정보
제로베이스 모델합성
• 플랫폼 모델생성
• 재난중심 모델생성
• 모델정제
재난오류 데이터베이스
재난오류 데이터베이스화
• 오류 일반화
• 오류 데이터베이스화
• 오류스펙 생성
비정상 상황 감지/복구
재난 정형 모델
순위기반 오류식별
시스템 레벨 재난 요구사항
• 컨트랙트 재사용
• 모듈 컨트랙트 추론
• 모듈 재난원인 추론
자동스펙 생성
모듈별 재난 요구사항
• 오류 패턴 학습
• 패턴기반 오류탐색
• 패턴기반 요약
스마트 모델검증
유사 오류 방지 테스트/패치


KAIST
건강달리기 대회
KAIST FUN RUN COMPETITION
FINISHER
For Healthy Body And Mind