포인터⇔정수간의 변환을 안전하게 최적화하기

Seoul National Univ. — **Juneyoung Lee**, Chung-Kil Hur

MPI-SWS — Ralf Jung

University of Utah — Zhengyang Liu, John Regehr

Microsoft Research — Nuno P. Lopes

European LLVM Developers' Meeting (EuroLLVM)

- 1년에 2번 열리는 LLVM Developers' Meeting 중 하나
- 1박 2일동안 다양한 출신의 프로그래머/연구자들이 참여

European LLVM Developers' Meeting (EuroLLVM)

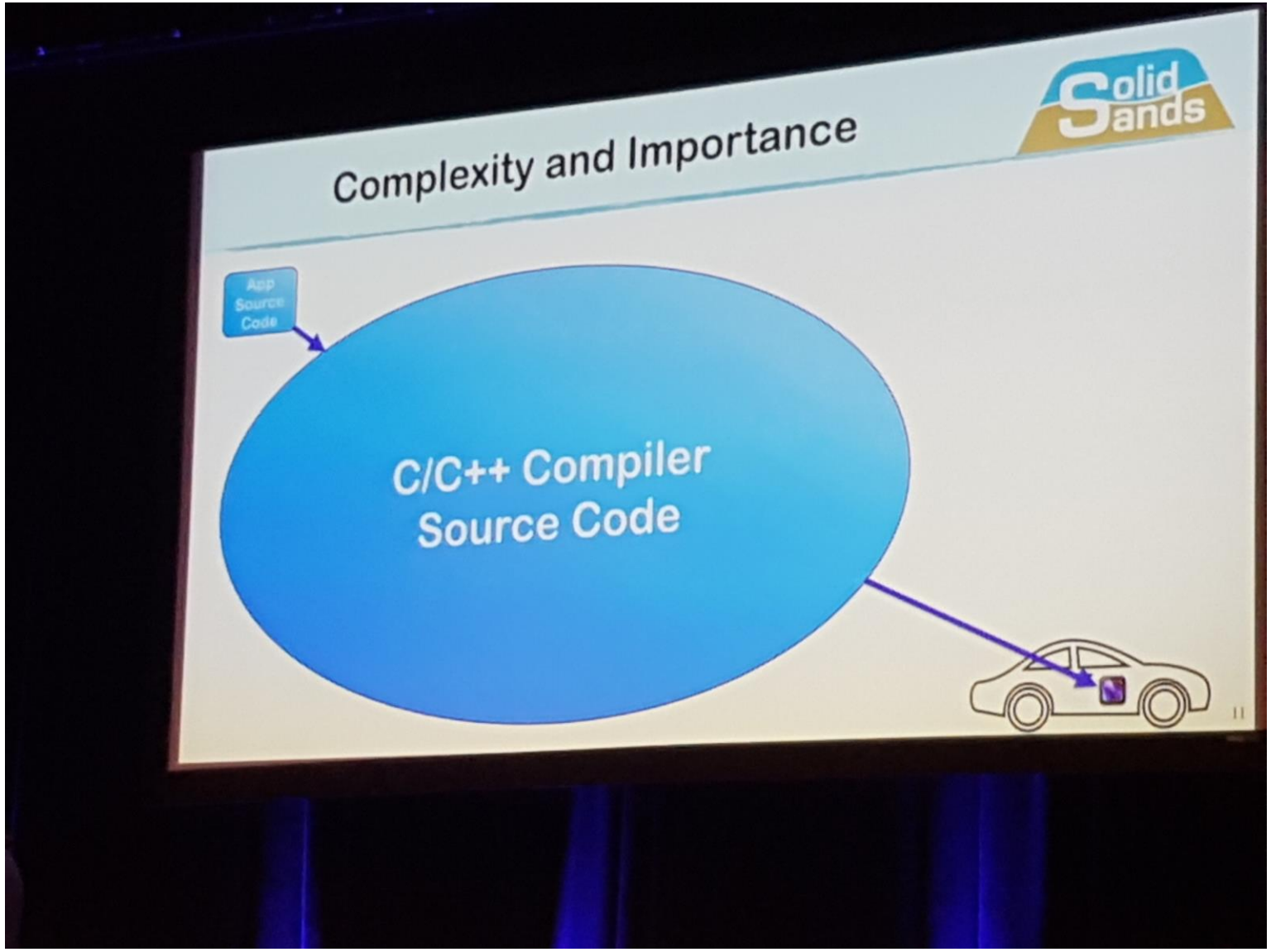
- 1년에 2번 열리는 LLVM Developers' Meeting 중 하나
- 1박 2일동안 다양한 출신의 프로그래머/연구자들이 참여
- Tutorial / Tech Talk / Student Research Competition(SRC) / ..
  - SRC: 총 6명의 발표가 서로 경합 // 이 주제로 2등상 수상하였습니다 ☺

EuroLLVM에서 얻은 좋은 경험들

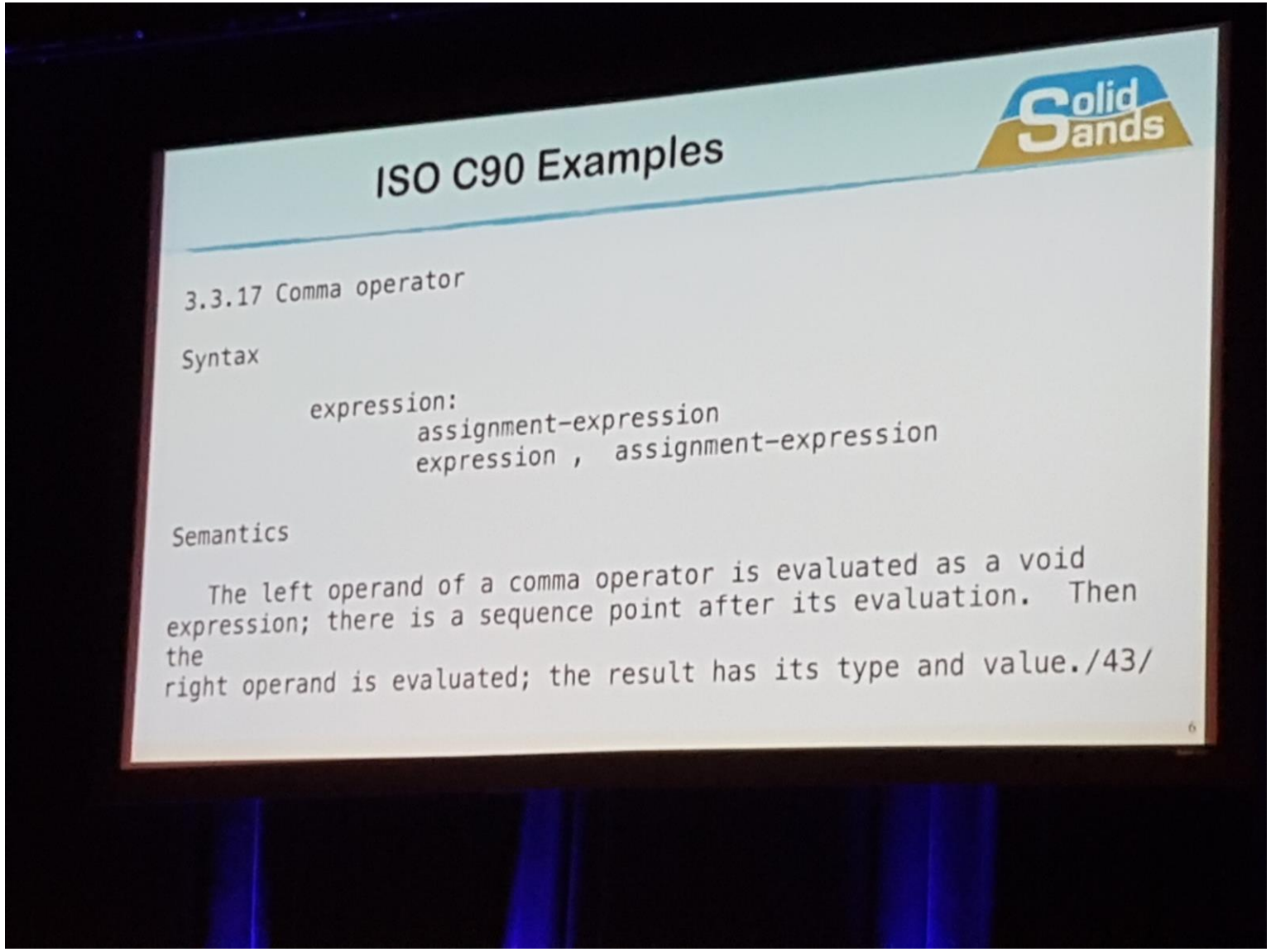
- 인상 깊었던 톡
  - XLA 텐서플로우 컴파일러 최적화 (“Kernel Fusion”)
  - 리눅스를 Clang으로 컴파일할 때 만났던 이슈들
  - 페이스북 리퀘스트의 처리 로직을 Just-In-Time 컴파일 하기
- 여러 사람들과의 대화
  - Rust 개발자, Apple 개발자, ECU 개발자 등등

안전성과 관련된 Technical Talk

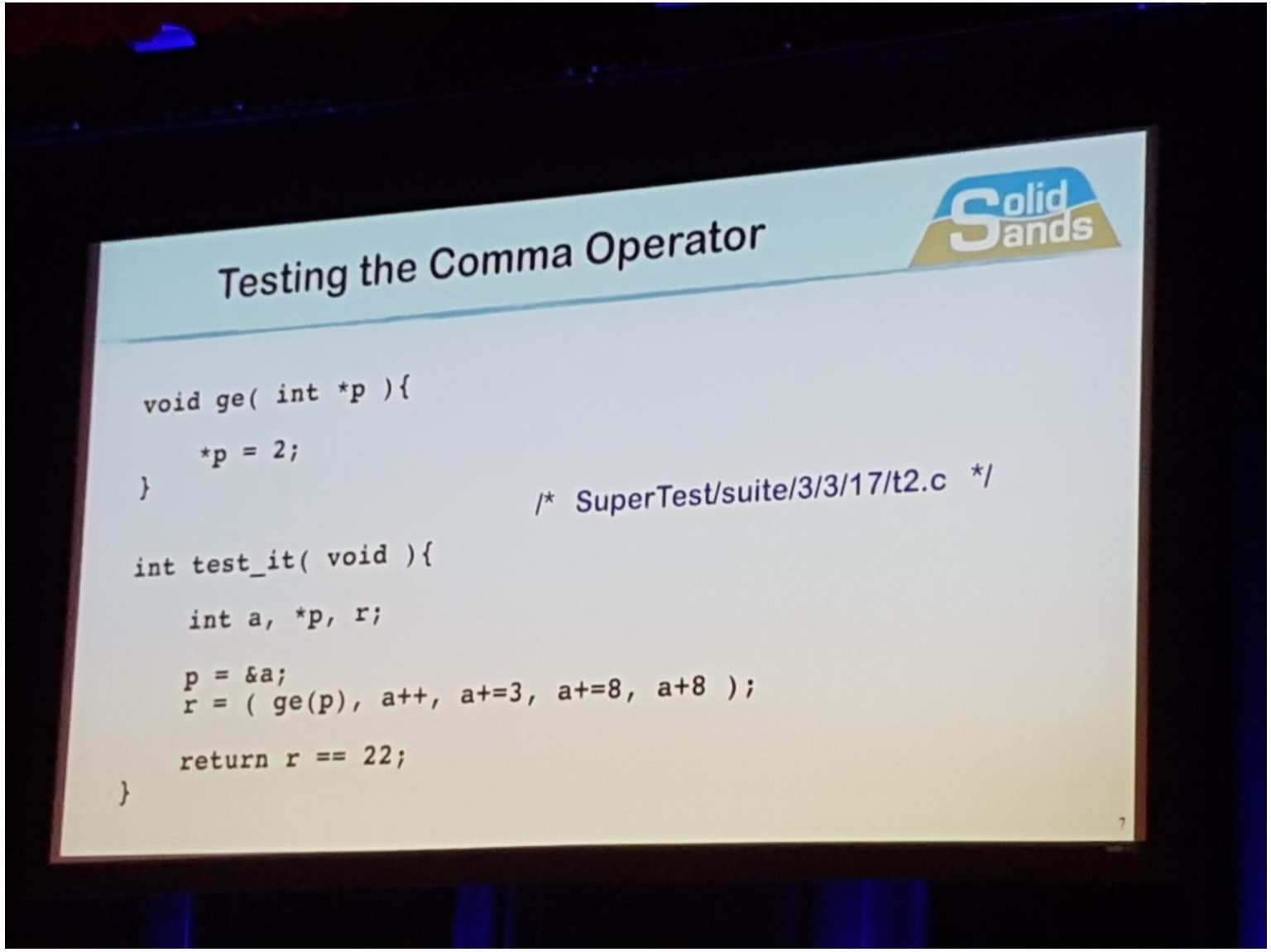
안전성과 관련된 Technical Talk



안전성과 관련된 Technical Talk



안전성과 관련된 Technical Talk



개괄

	Assembly (x86-64, ARM, ..)	**LLVM IR**
Pointer	$[0, 2^{64})$	$[0, 2^{64})$ + *출신지(provenance)*
Integer	$[0, 2^{64})$	$[0, 2^{64})$ + **?**

문제: 포인터가 순수한 숫자라면?

We use C syntax for LLVM IR code for readability

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?

We use C syntax for LLVM IR code for readability

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?

We use C syntax for LLVM IR code for readability

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?

We use C syntax for LLVM IR code for readability

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?

We use C syntax for LLVM IR code for readability

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?

Memory:

0x0

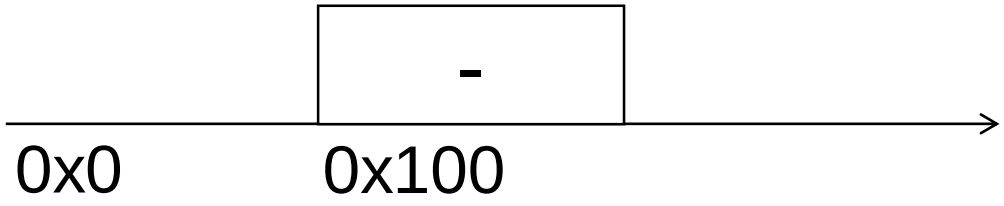
```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?



0x100

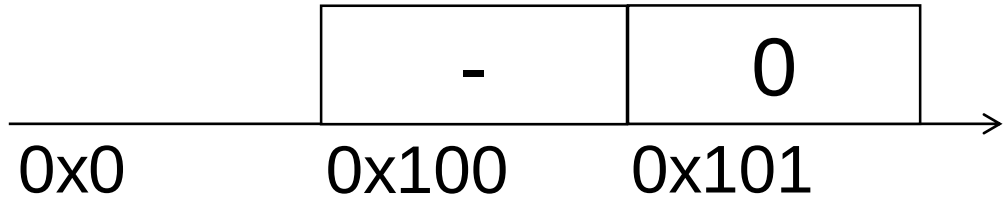
```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?



0x100 0x101

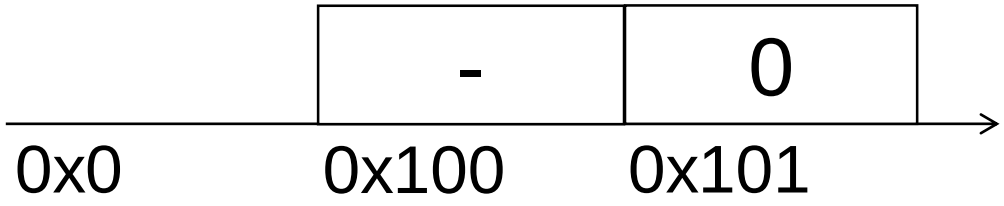
```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?



0x100 0x101

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

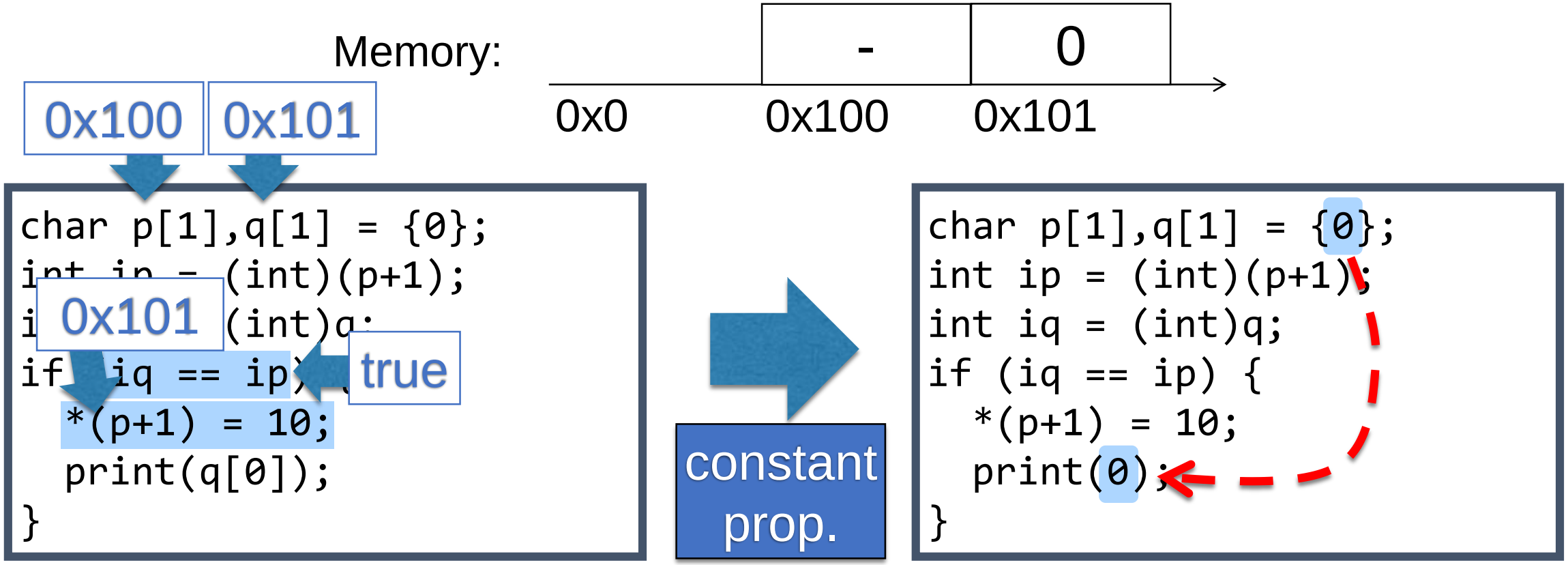
true

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

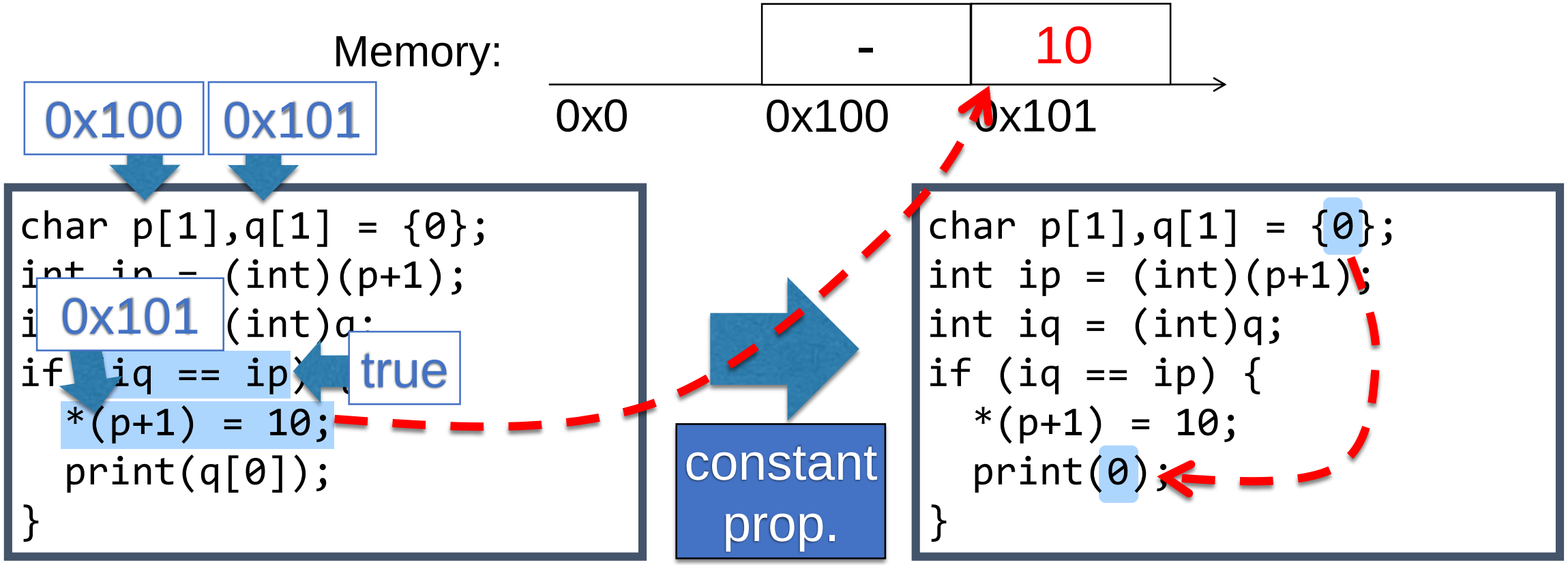
* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?



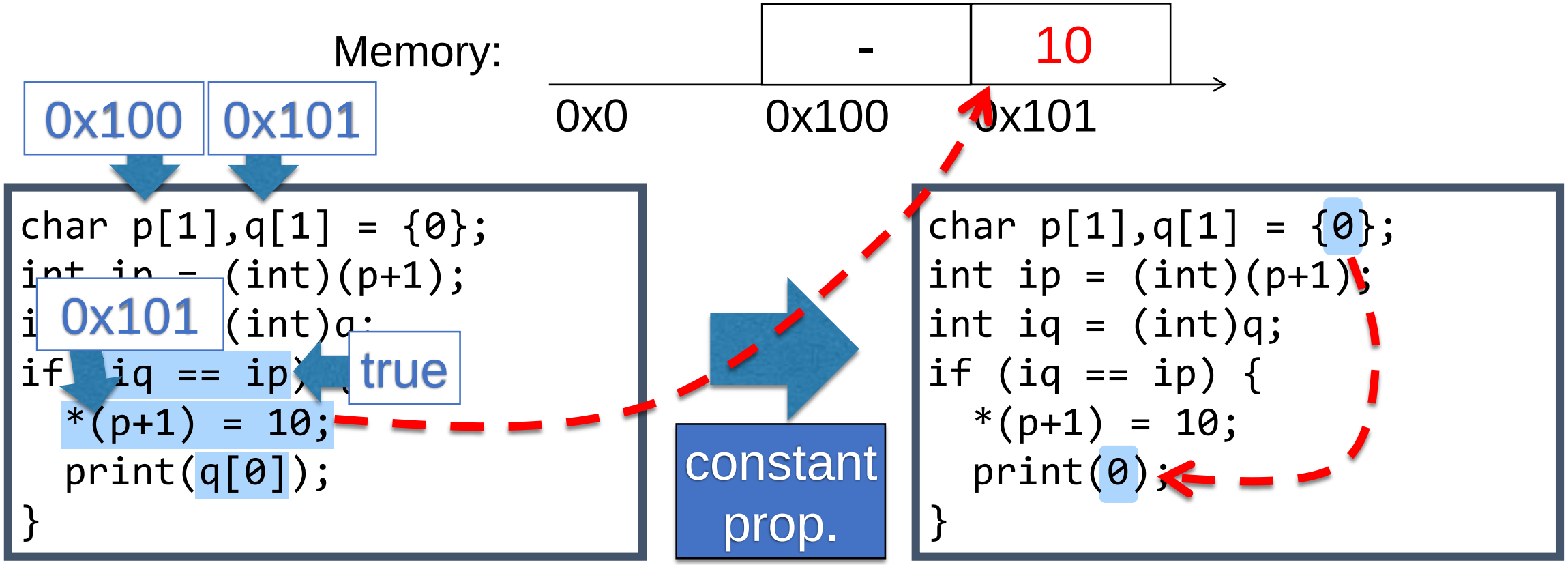
* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?



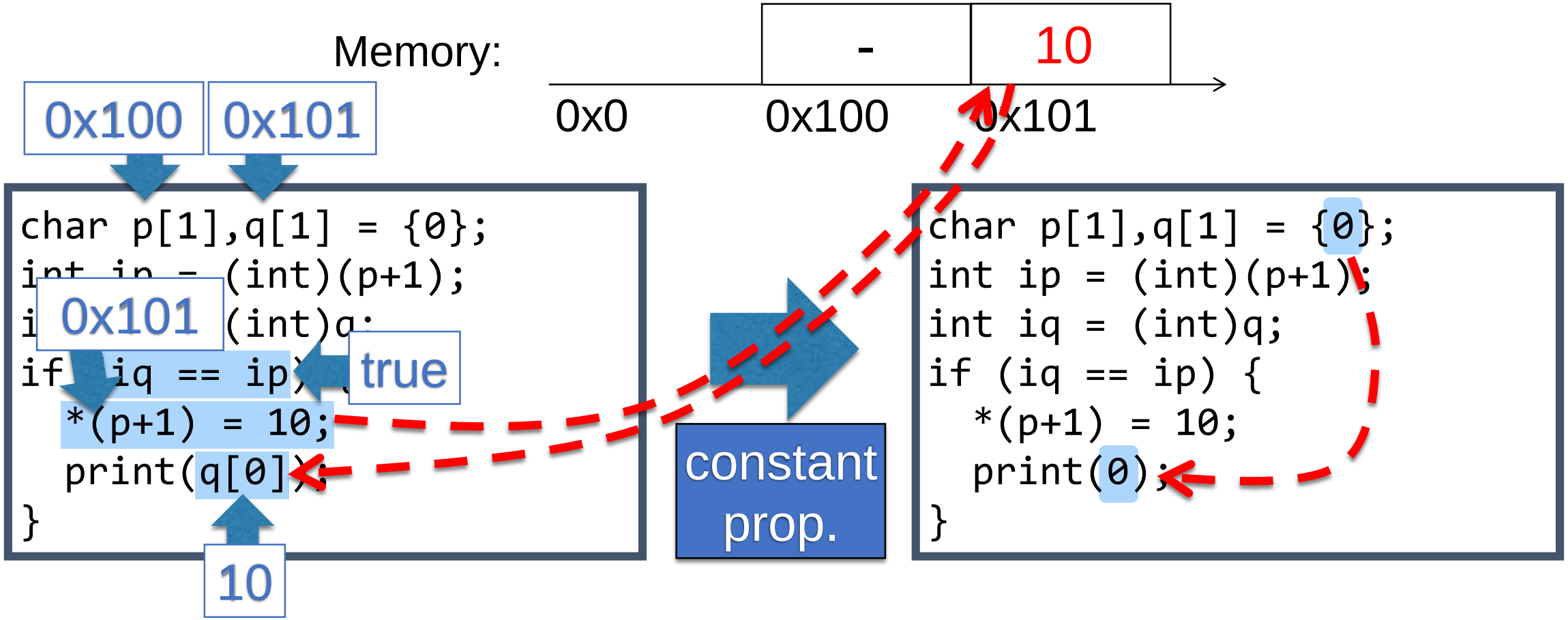
* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?



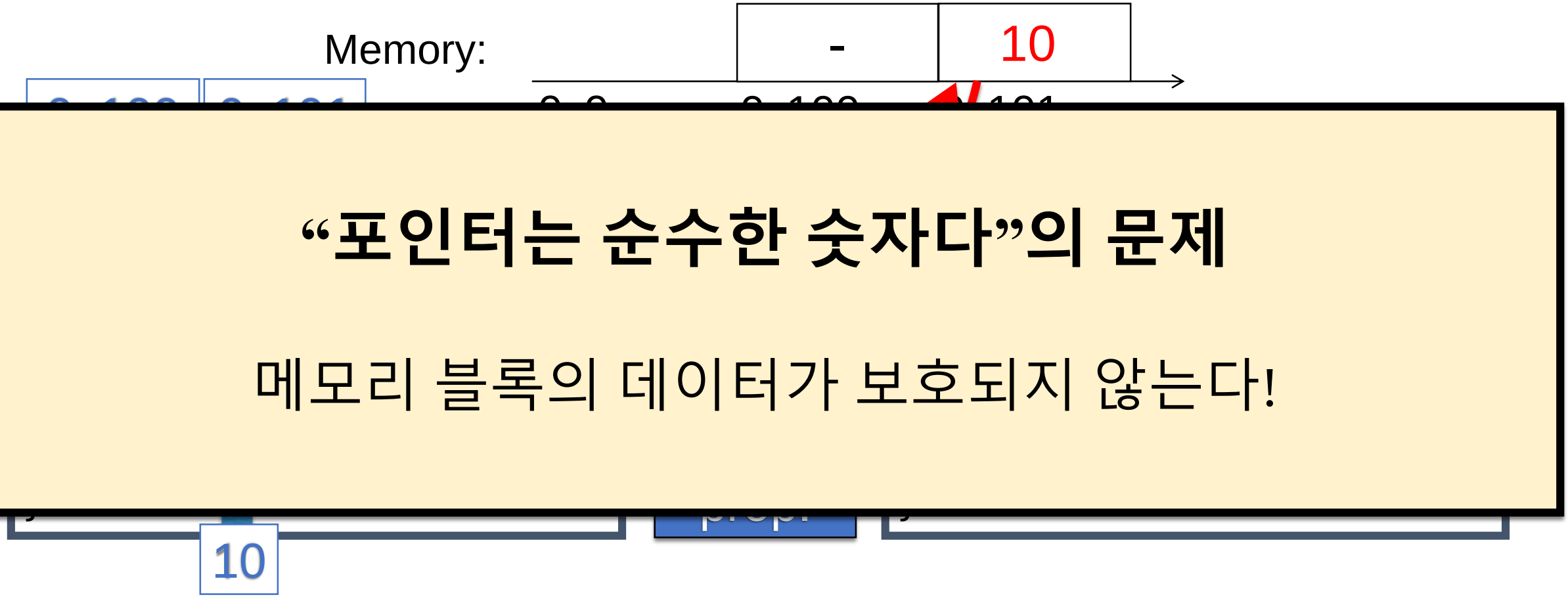
* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?



* https://godbolt.org/z/9eNt6w

문제: 포인터가 순수한 숫자라면?

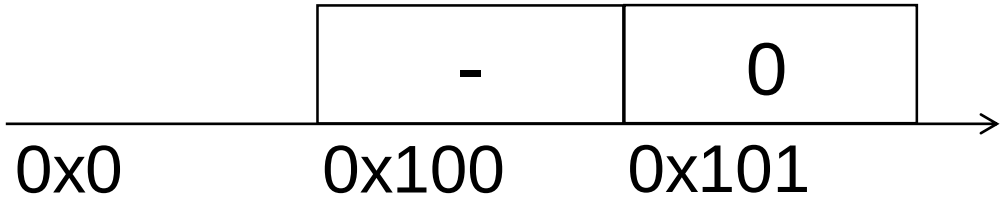


“포인터는 순수한 숫자다”의 문제

메모리 블록의 데이터가 보호되지 않는다!

* https://godbolt.org/z/9eNt6w

LLVM의 해결책: 포인터는 자기 출신을 기억한다



0x100 0x101

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

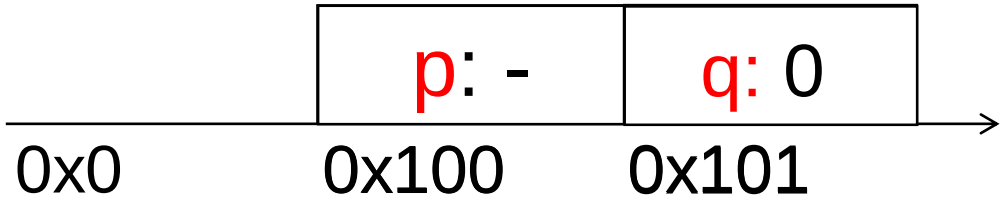
* https://godbolt.org/z/9eNt6w

LLVM의 해결책: 포인터는 자기 출신을 기억한다

나의 출신

Memory:

(p,0x100) (q,0x101)




```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

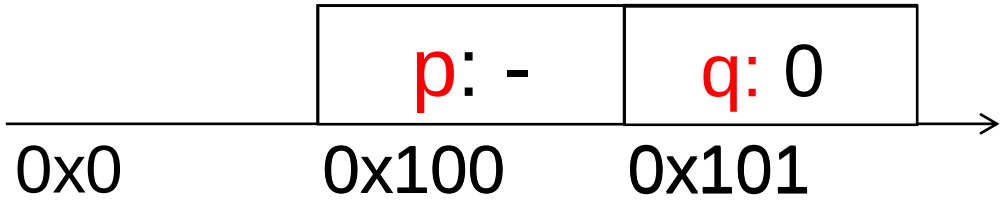
* https://godbolt.org/z/9eNt6w

LLVM의 해결책: 포인터는 자기 출신을 기억한다

나의 출신

Memory:

(p,0x100) (q,0x101)




```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

true

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

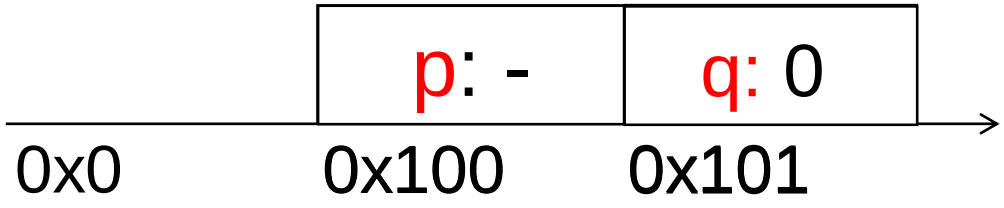
* https://godbolt.org/z/9eNt6w

LLVM의 해결책: 포인터는 자기 출신을 기억한다

나의 출신

Memory:

(p,0x100) (q,0x101)




```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

true

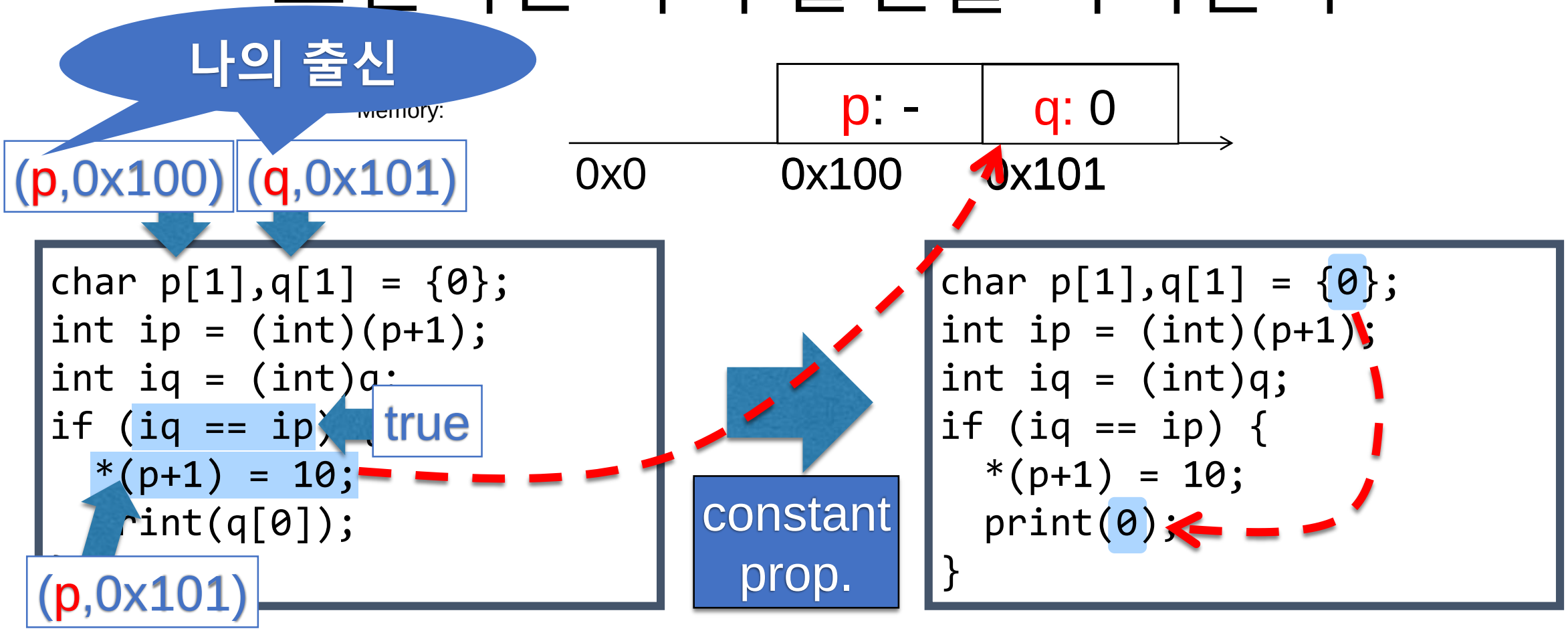
(p,0x101)

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

* https://godbolt.org/z/9eNt6w

LLVM의 해결책:
포인터는 자기 출신을 기억한다



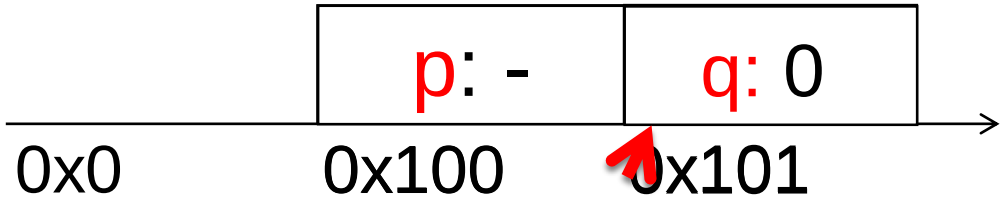
* https://godbolt.org/z/9eNt6w

LLVM의 해결책: 포인터는 자기 출신을 기억한다

나의 출신

Memory:

(p,0x100) (q,0x101)




```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  printf(q[0]);
```

true

(p,0x101)

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
```

출신 p ≠ q 이기 때문에
정의되지 않은 행동
(Undefined Behavior)

* https://godbolt.org/z/9eNt6w

그렇다면 정수는?

	Assembly (x86-64, ARM, ..)	LLVM IR
Pointer	$[0, 2^{64})$	$[0, 2^{64})$ + *출신지*
Integer	$[0, 2^{64})$	$[0, 2^{64})$ + ?

변환

포인터-정수 변환 관련 버그

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

10

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

10

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

10

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

포인터-정수 변환 관련 버그

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

LLVM & GCC 에서 둘 다 이 버그가 존재합니다!

어느 최적화가 문제일까?

원인은 정수의 정의에 따라 다르다

정수도 자기 출신을
기억한다면 이것을 설명 못함

```
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq.
prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
```

cast
elim.

정수가 순수한 숫자면
이것을 설명 못함

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant
prop.

```
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

정수도 자기 출신을 기억한다면?

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

정수도 자기 출신을 기억한다면?

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

정수도 자기 출신을 기억한다면?

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

정수도 자기 출신을 기억한다면?

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0])
}
```

q 출신이다

cast elim.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1]
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

정수도 자기 출신을 기억한다면?

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

p 출신이다

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0])
}
```

q 출신이다

cast elim.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

정수는 순수한 숫자일 뿐이라면?

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

cast elim

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

정수는 순수한 숫자일 뿐이라면?

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

cast elim

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

정수는 순수한 숫자일 뿐이라면?

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

자기 출신을
잊어버림

cast elim

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

int. eq. prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

정수는 순수한 숫자일 뿐이라면?

```
char p[1
int ip =
int iq =
if (iq ==
  *(char*)(int)(p+1)=10;
  print(q[0]);
}
```

자기 출신을
잊어버림

cast
elim

```
char
int
int
if (iq  ip) {
  *(p+1) = 10;
  print(q[0]);
}
```

자기 출신을
기억함

int. eq.
prop.

constant
prop.

```
char p[1],q[1]={0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(char*)iq = 10;
  print(q[0]);
}
```

```
char p[1],q[1] = {0};
int ip = (int)(p+1);
int iq = (int)q;
if (iq == ip) {
  *(p+1) = 10;
  print(0);
}
```

출신을 기억하는 정수는 자연스럽지 않다

- Hard to explain integer equality propagation
- Hard to explain many other transformations as well

```
r = (i + j) - k
```

⇔

```
r = i + (j - k)
```

```
r = (int)(float)j
```

⇨

```
r = j
```

출신을 기억하는 정수는 자연스럽지 않다

- Hard to explain integer equality propagation
- Hard to explain many other transformations as well

prov. + prov.?

prov. – prov.?

```
r = (i + j) - k
```

⇔

```
r = i + (j - k)
```

```
r = (int)(float)j
```

⇨

```
r = j
```

출신을 기억하는 정수는 자연스럽지 않다

- Hard to explain integer equality propagation
- Hard to explain many other transformations as well

prov. + prov.?

prov. – prov.?

```
r = (i + j) - k
```

⇔

```
r = i + (j - k)
```

```
r = (int)(float)j
```

⇨

```
r = j
```

provenance in float types?

우리의 제안 [OOPSLA'18]: 정수는 순수한 수일 뿐인 모델

	Assembly (x86-64, ARM, ..)	**LLVM IR**
Pointer	$[0, 2^{64})$	$[0, 2^{64})$ + *출신지*
Integer	$[0, 2^{64})$	$\mathbf{[0, 2^{64})}$

정수는 순수한 수일 뿐인 모델

- 변환의 의미
- 문제가 되는 최적화들
- 어떻게 성능을 회복할까?

포인터⇔정수 변환의 의미 [OOPSLA'18]

1. 포인터 → 정수 변환의 결과는 출신을 잃어버린다
2. 정수 → 포인터 변환의 결과는 모든 곳 출신(full provenance) 이다

메모리 블록의 데이터를 잘 보호할 수 있을까?

Nondeterministic allocation 을 사용하자!

모든곳 출신 포인터에 대한 in-bounds 확인은 어떻게?

By recording in-bounds offsets at the pointer & checking when dereferenced

우리 모델에서 옳지 않은 최적화

1. Cast Elimation

```
p2 = (char*)(int)p
```

⇨

```
p2 = p
```

2. Integer Comparison to Pointer Comparison

```
c = icmp eq (int)p, (int)q
```

⇨

```
c = icmp eq p, q
```

우리 모델에서 옳지 않은 최적화

1. Cast Elimation

```
p2 = (char*)(int)p
```

최대 출신

⇨

```
p2 = p
```

p 출신

2. Integer Comparison to Pointer Comparison

```
c = icmp eq (int)p, (int)q
```

⇨

```
c = icmp eq p, q
```

우리 모델에서 옳지 않은 최적화

1. Cast Elimation

```
p2 = (char*)(int)p
```

최대 출신

⇨

```
p2 = p
```

p 출신

2. Integer Comparison to Pointer Comparison

```
c = icmp eq (int)p, (int)q
```

Comparison of integers

⇨

```
c = icmp eq p, q
```

Comparison of pointers

성능 문제

- **Cast elimination 은 많은 변환 연산을 지워준다**
  - 13% of ptrtoints, 40% of inttoptrs from C/C++ benchmarks *
- **Cast elimination 을 끄면 다른 최적화에 영향이 있다**
  - ptrtoint makes variables escaped
  - inttoptr is regarded as pointing to an unknown object
- **Cast elimination 를 끄면 느려진다**
  - 1% slowdown in perlbench_r, blender_r

* SPEC2017rate + LLVM test-suite, -O3

우리의 해결책

1. **처음부터 포인터↔정수 변환을 만들지 말자**
   - 86% of Ptr↔Int casts are introduced by LLVM, not by programmers
     - Ptr → Int casts are generated from pointer subtractions
     - Int → Ptr casts are from canonicalizing loads/stores as int types
   - **How:** by introducing new features
2. **기존의 최적화를 조건적으로 허용하자**
   - **How:** by developing an analyzer to check such conditions

포인터→정수 변환을 줄이기 위해: 포인터 뺄셈 연산 도입

Before Fix (Uses ptrtoint)

```
ip = ptrtoint p
iq = ptrtoint q
i = ip - iq
```

After Fix (Uses psub)

```
i = psub p, q
```

$$\texttt{psub}\ p,\ q \stackrel{\text{def}}{=} \begin{cases} p - q & \text{If } prov(p) = prov(q) \lor prov(p) = \texttt{full} \lor prov(q) = \texttt{full} \\ \texttt{poison} & \text{Otherwise} \end{cases}$$

정수→포인터 변환을 줄이기 위해: load/store의 일반화 멈춤

```
v = load i64* p
v2= load i8** p
```

* https://godbolt.org/z/y48Mkt

정수→포인터 변환을 줄이기 위해: load/store의 일반화 멈춤

```
v = load i64* p
v2= load i8** p
```

⇩

```
v = load i64* p
v2= inttoptr v
```

* https://godbolt.org/z/y48Mkt

정수→포인터 변환을 줄이기 위해: load/store의 일반화 멈춤

```
v = load i8** p
v2= load i8** p
```

⇩

```
v = load i64* p
v2= load i8** p
```

⇩

```
v = load i64* p
v2= inttoptr v
```

* https://godbolt.org/z/y48Mkt

정수→포인터 변환을 줄이기 위해: load/store의 일반화 멈춤

```
v = load i8** p
v2= load i8** p
```

Use ‘d64’ (data type) instead

```
v = load i64* p
v2= load i8** p
```

```
v = load i64* p
v2= inttoptr v
```

	출신을 기억하는가?	정수 연산을 지원하는가?
d64	예	아니오
i64	아니오	예

Unlike cast between int⇔ptr, d64⇔ptr preserves provenance.

* https://godbolt.org/z/y48Mkt

정수→포인터 변환을 줄이기 위해: load/store의 일반화 멈춤

```
v = load i8** p
v2= load i8** p
```

Use 'd64' (data type) instead

```
v = load i64* p
v2= load i8** p
```

```
v = load i64* p
v2= inttoptr v
```

	출신을 기억하는가?	정수 연산을 지원하는가?
d64	예	아니오
i64	아니오	예

Unlike cast between int⇔ptr, d64⇔ptr preserves provenance.

* https://godbolt.org/z/y48Mkt

부분적으로 최적화 허용하기

```
// p 와 q 는 같은 출신일 때
p2 = inttoptr(ptrtoint p)
c  = icmp eq/ne p2, q
```

⇨

```
c  = icmp eq/ne p, q
```

```
p2 = inttoptr(ptrtoint p)
c  = psub p2, q
```

⇨

```
c  = psub p, q
```

- More examples & descriptions are listed at https://github.com/aqjune/eurollvm19

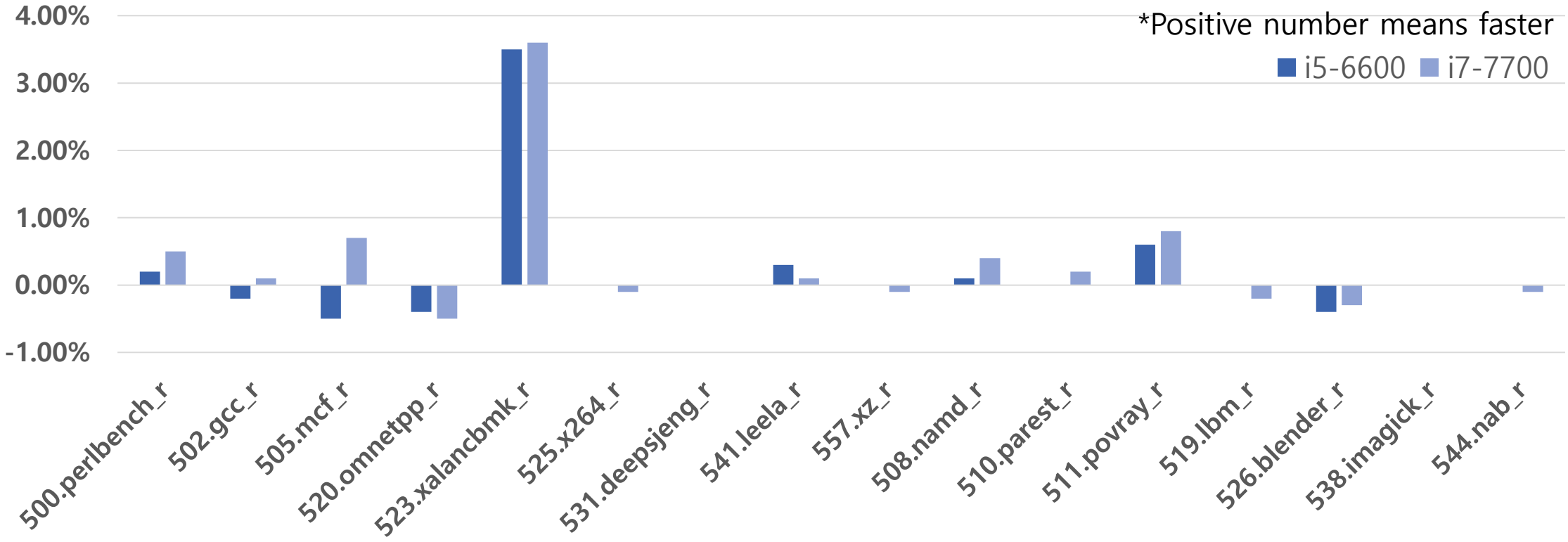
실험 결과: 1. 변환 연산의 수

Disable unsound opts. → Add psub, stop load/store to int → Conditionally allow cast elim.

		Baseline (LLVM 8.0)	No Cast Fold	Reduce Cast Introduction	Conditionally Fold
-O3 이전	# of ptrtoints	44K	44K	14K	14K
	# of inttoptrs	1.5K	1.5K	1.5K	1.5K
-O3 이후	# of ptrtoints	57K	66K	11K	11K
	# of inttoptrs	29K	45K	5K	4.8K

- SPEC2017rate + LLVM Nightly Tests 에서 C/C++ 벤치마크들이 사용됨
- 81% of ptrtoints / 83% of inttoptrs 를 성공적으로 지움 (baseline과 비교)

실험 결과: 2. 성능 변화



<SPEC2017rate 속도 향상>

- LLVM Nightly Tests (C/C++): ~0.1% avg. slowdown (-1% ~ 3.6%)

결론

- 컴파일러는 포인터의 출신 정보를 이용해 최적화를 한다.
- 정수가 자기 출신을 기억하면 컴파일러 최적화를 설명하기 힘들다.
- 우리 모델은 정수와 포인터는 서로 다르게 정의한다.
- 올바르지 않은 최적화를 끄고 나서 성능은 큰 저하가 없었다.

https://github.com/aqjune/eurollvm19

결론

We're updating Alive to support pointer-integer casts! ☺

```
PROGRAM: Name: ptrintload3
  ENTRY:
    v16 = ptrtoint i8* p1 to i16
    p2 = inttoptr i16 v16 to i8*
    v2 = load i8* p2
    v1 = load i8* p1
PRECONDS:
        Instruction "v2 = load i8* p2" has no UB.
CHECK:
        Instruction "v1 = load i8* p1" has no UB?
        v1 === v2?
Result: INCORRECT
```

https://github.com/aqjune/eurollvm19

supplementary slides

Constant Propagation and Readonly function

```
char p[1],q[1] = {0};


if (foo(p, q)) { //readonly
  *(p+i) = 10;
  print(q[0]);
}
```

constant prop.

```
char p[1],q[1] = {0};


if (foo(p, q)) { //readonly
  *(p+i) = 10;
  print(0);
}
```

Constant Propagation and Readonly function

```
char p[1],q[1] = {0};
```

return (int)(p+1) == (int)q?

```
if (foo(p, q)) { //readonly
  *(p+i) = 10;
  print(q[0]);
}
```

1?

constant prop.

```
char p[1],q[1] = {0};


if (foo(p, q)) { //readonly
  *(p+i) = 10;
  print(0);
}
```

Integer Equality Propagation and Performance

- Performed by many optimizations
  - CVP, Instruction Simplify, GVN, Loop Exit Value Rewrite, …
- Reduces code size

  -10% in minisat, -6% in smg2000, -4% in simple_types_constant_folding, …
- Boosts performance in small benchmarks
  - x2000 speedup in nestedloop

Sound Optimizations that are already in LLVM

```
gep(p, -(int)q)
```

➡

```
(void*)((int)p-(int)q)
```

```
select (p==null), p, null
```

➡

```
null // null=(void*)0
```

Rationale

It is safe to replace p with (void*)(int)p.

Delayed Inbounds Checking

```
p = (char*)0x100 // p=(0x100,*)
p2 = gep p, 1    // p=(0x101,*)

p3 = gep inbounds p, 1
                 // p = (0x101,*,{0x100,0x101})

load p3          // 0x100, 0x101 should be
                 // in-bounds addrs of the
                 // object at 0x101
```