# CCured: Type-Safe Retrofitting of Legacy Software

G. C. Necula, J. Condit, M. Harren, S. McPeak, and W. Weimer

(TOPLAS '05)

Jaemin Hong 20203698

# Undefined behaviors in C cause vulnerabilities.

```c
void foo(int *ptr, int idx, int val) {
    *(ptr + idx) = val;
}
```

# Undefined behaviors in C cause vulnerabilities.

```c
void foo(int *ptr, int idx, int val) {
    *(ptr + idx) = val;
}

int arr[10];
foo(arr, N, M);
```

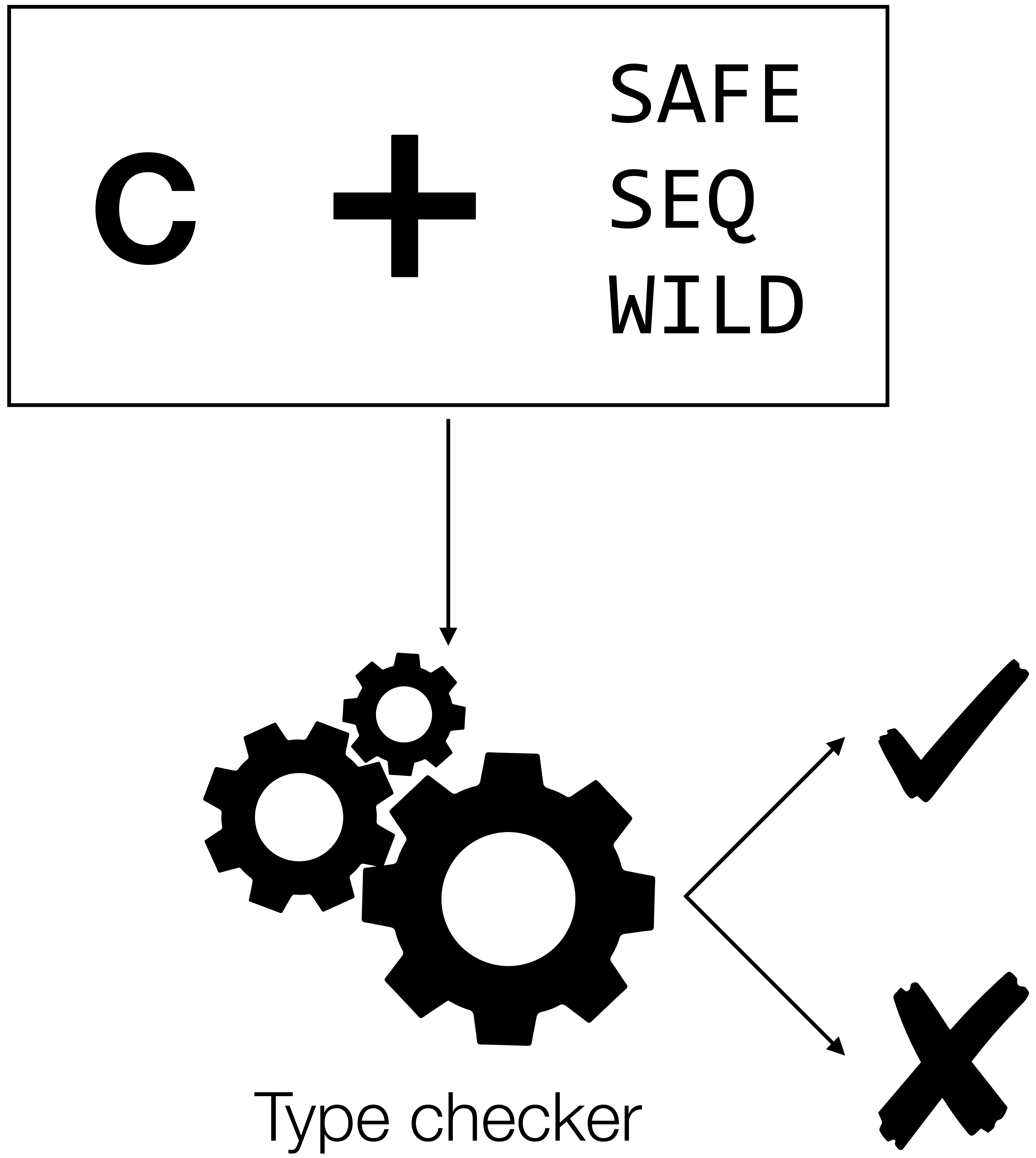
# Undefined behaviors in C cause vulnerabilities.

```
struct a { int x; };
struct b { int *p; };

int i;
struct b sb = { &i };



*sb.p = M;
```

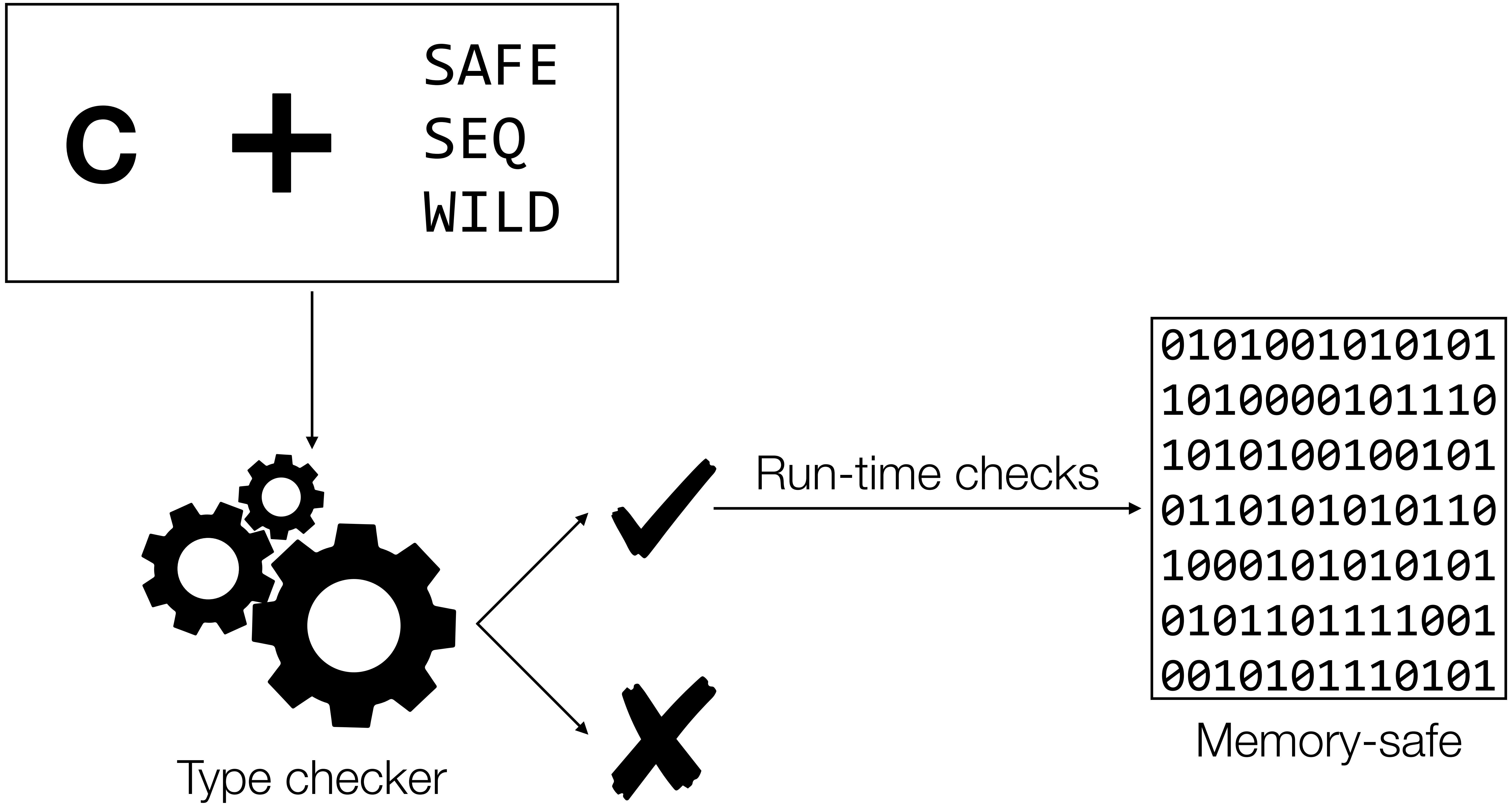
# Undefined behaviors in C cause vulnerabilities.

```
struct a { int x; };
struct b { int *p; };

int i;
struct b sb = { &i };
struct a *sa = (struct a *) &sb;
sa->x = N;
*sb.p = M;
```

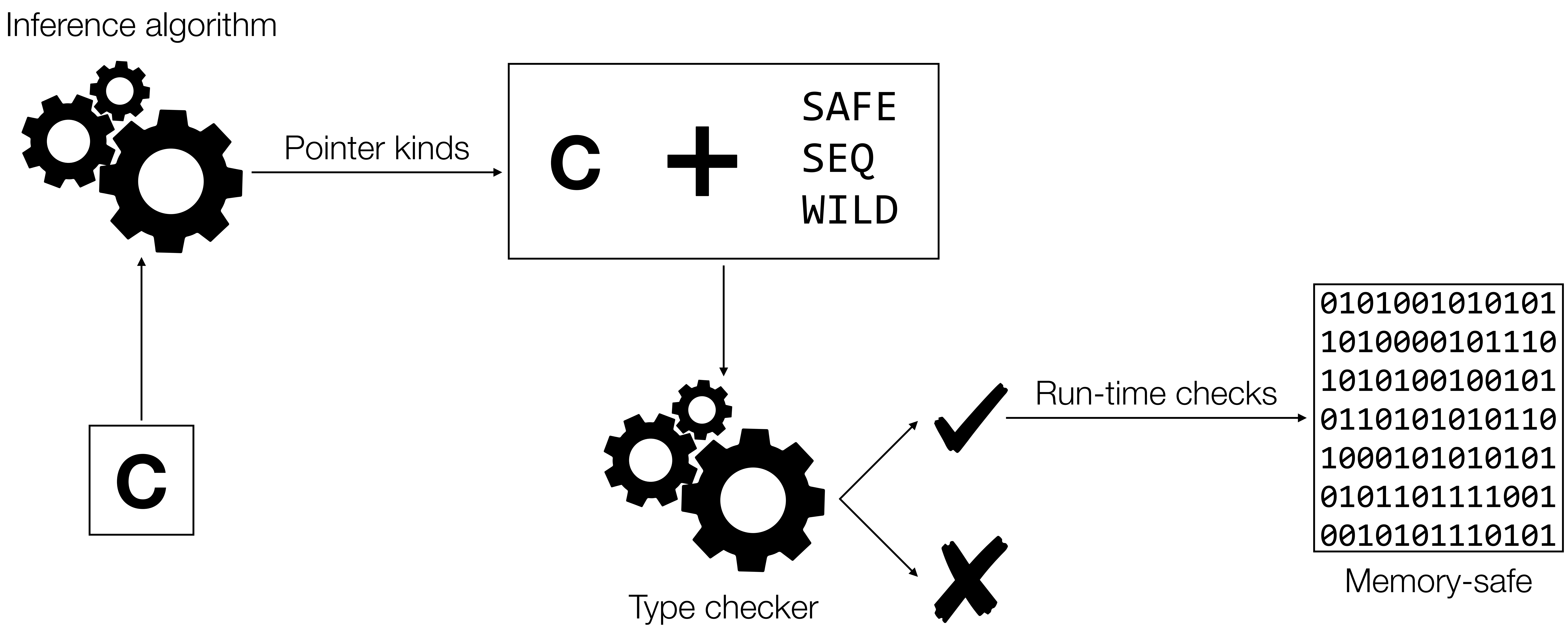
# CCured transforms C programs to achieve memory safety guarantees.



# CCured transforms C programs to achieve memory safety guarantees.



# CCured transforms C programs to achieve memory safety guarantees.



# CCured classifies pointers into 3 kinds.

$$\text{Type} \quad \tau \quad ::= \quad \texttt{int} \mid \tau\ *$$

**Types**

`int`

`int * *`

`int *`

...

# CCured classifies pointers into 3 kinds.

$$\begin{array}{llll} \text{Type} & \tau & ::= & \mathtt{int} \mid \tau * q \\ \text{Kind} & q & ::= & \mathtt{SAFE} \mid \mathtt{SEQ} \mid \mathtt{WILD} \end{array}$$

**Types**

`int * WILD * WILD`

`int`

`int * SEQ`

`int * SAFE`

...

# CCured classifies pointers into 3 kinds.

$$\begin{array}{llll} \text{Type} & \tau & ::= & \texttt{int} \mid \tau * q \\ \text{Kind} & q & ::= & \texttt{SAFE} \mid \texttt{SEQ} \mid \texttt{WILD} \end{array}$$

Static restrictions

More ← Less

**Types**

`int * WILD * WILD`

`int`

`int * SEQ`

`int * SAFE`

...

# CCured classifies pointers into 3 kinds.

$$
\begin{array}{llll}
\text{Type} & \tau & ::= & \texttt{int} \mid \tau * q \\
\text{Kind} & q & ::= & \texttt{SAFE} \mid \texttt{SEQ} \mid \texttt{WILD}
\end{array}
$$

Static restrictions

More $\longleftarrow$ Less

Less $\longrightarrow$ More

Run-time checks

**Types**

`int * WILD * WILD`

`int`

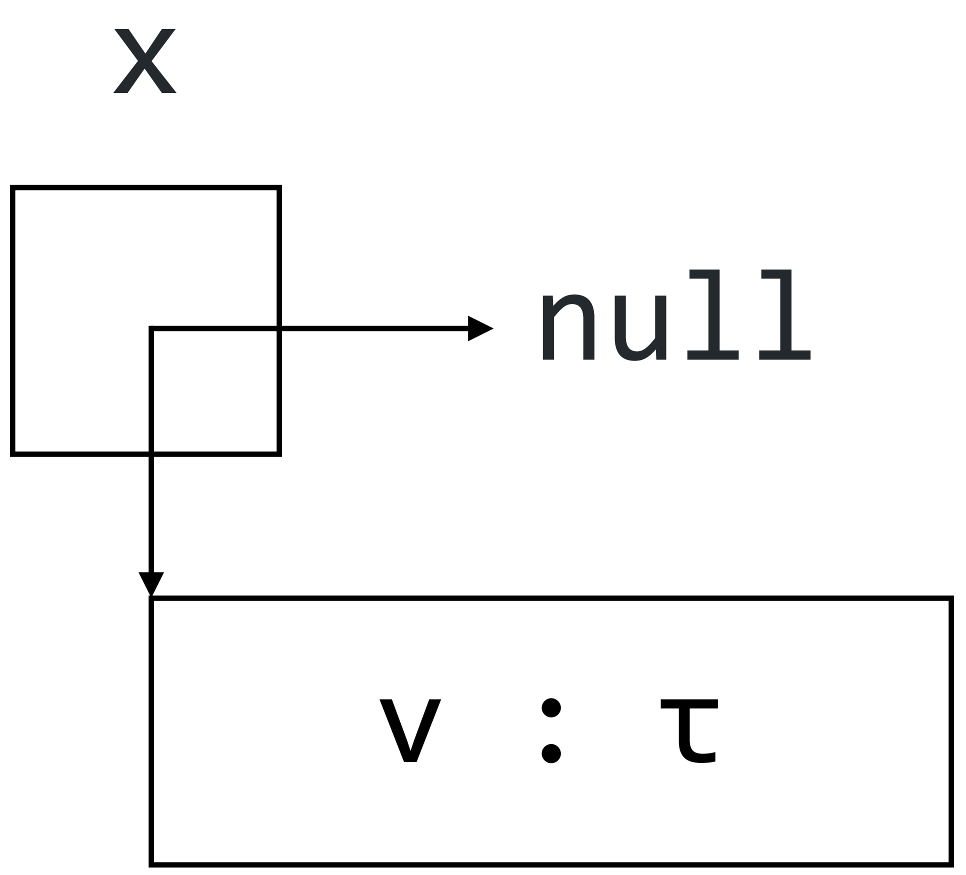
`int * SEQ`

`int * SAFE`

...

# Safe pointers are either null or valid.

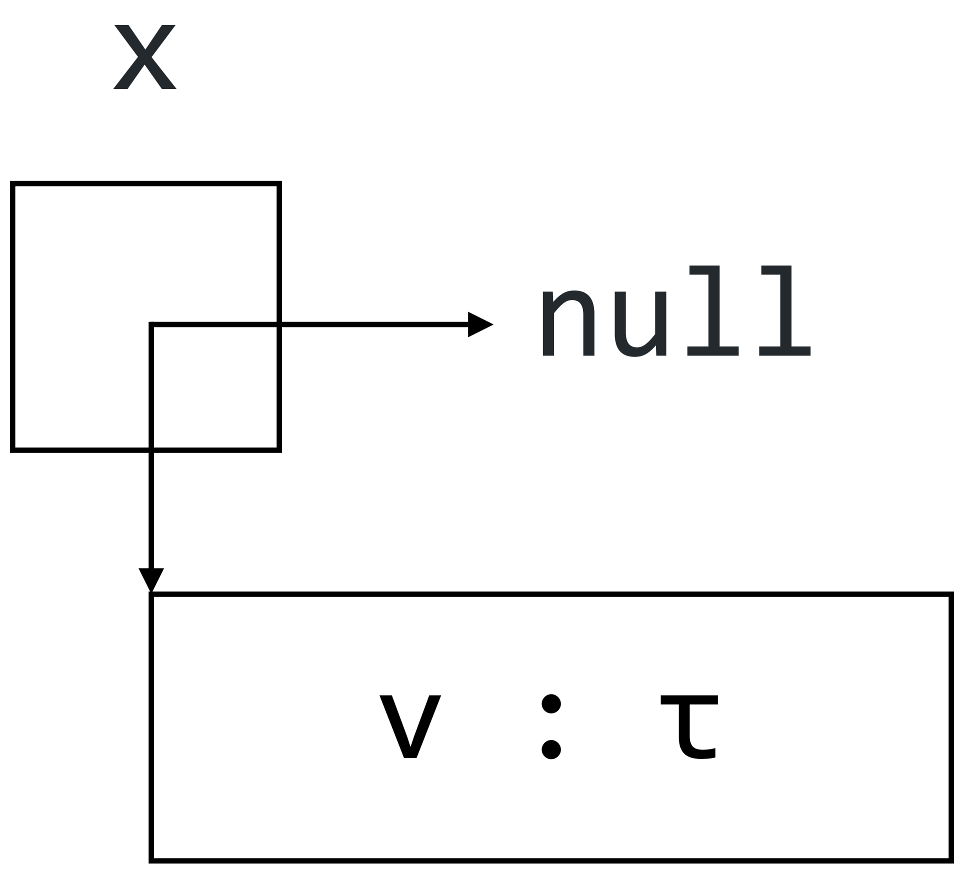
$$Rep(\tau \ * \ \mathsf{SAFE}) = Rep(\tau)\ *$$




$$x : \tau \ * \ \mathsf{SAFE}$$

# Safe pointers need null checks.

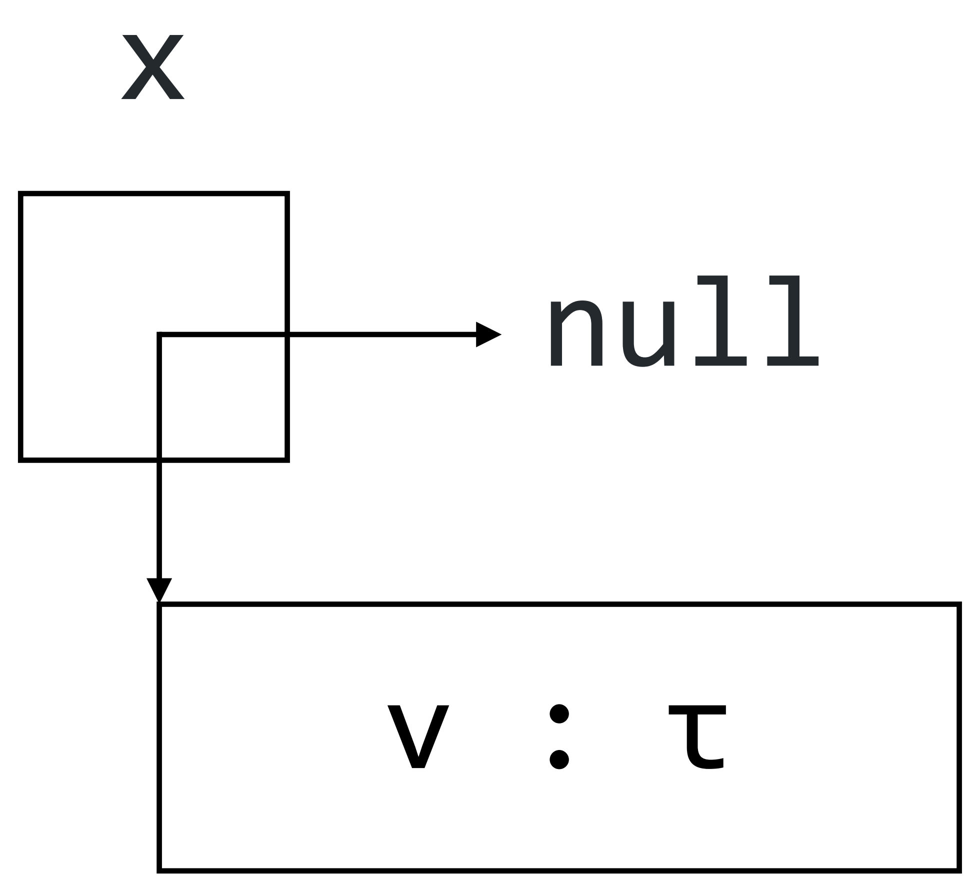
$$Rep(\tau\ *\ \mathsf{SAFE}) = Rep(\tau)\ *$$




$$\frac{x : \tau\ *\ \mathsf{SAFE}}{*x \rightsquigarrow \qquad\qquad\qquad\qquad *x}$$

# Safe pointers need null checks.

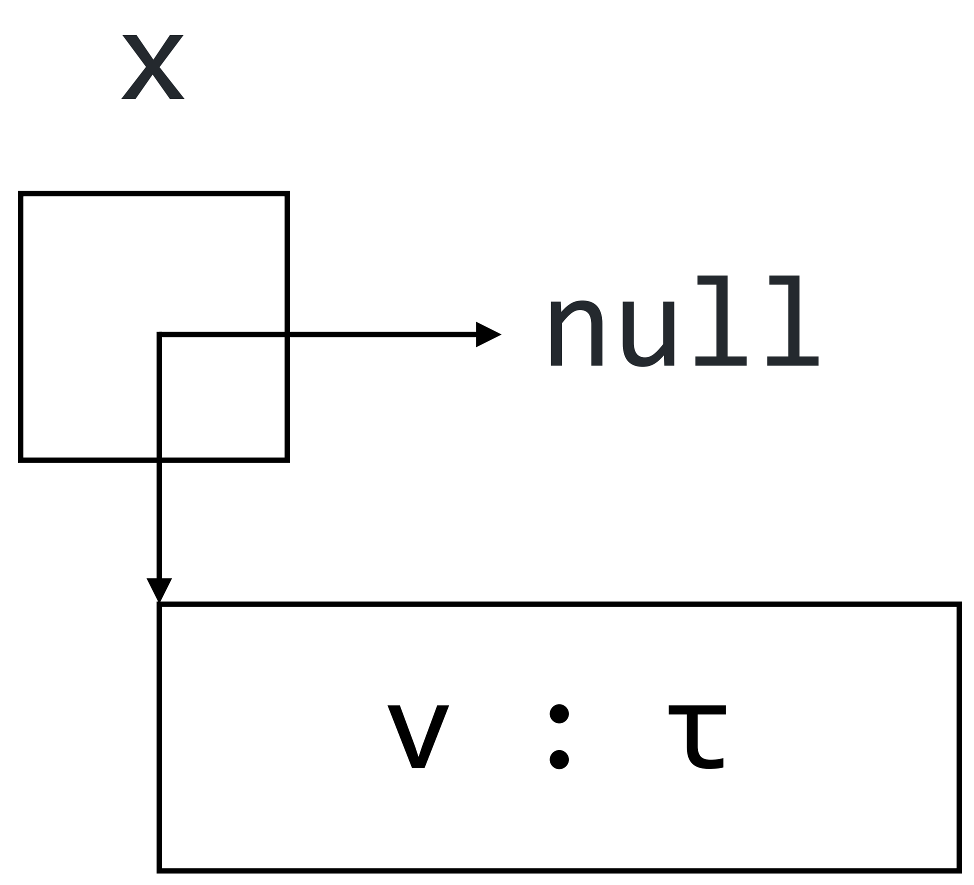
$$Rep(\tau \ * \ \mathsf{SAFE}) = Rep(\tau) *$$




$$\frac{x : \tau \ * \ \mathsf{SAFE}}{*x \leadsto \mathtt{assert}(x \neq \mathtt{null}); *x}$$

# Safe pointers need null checks.

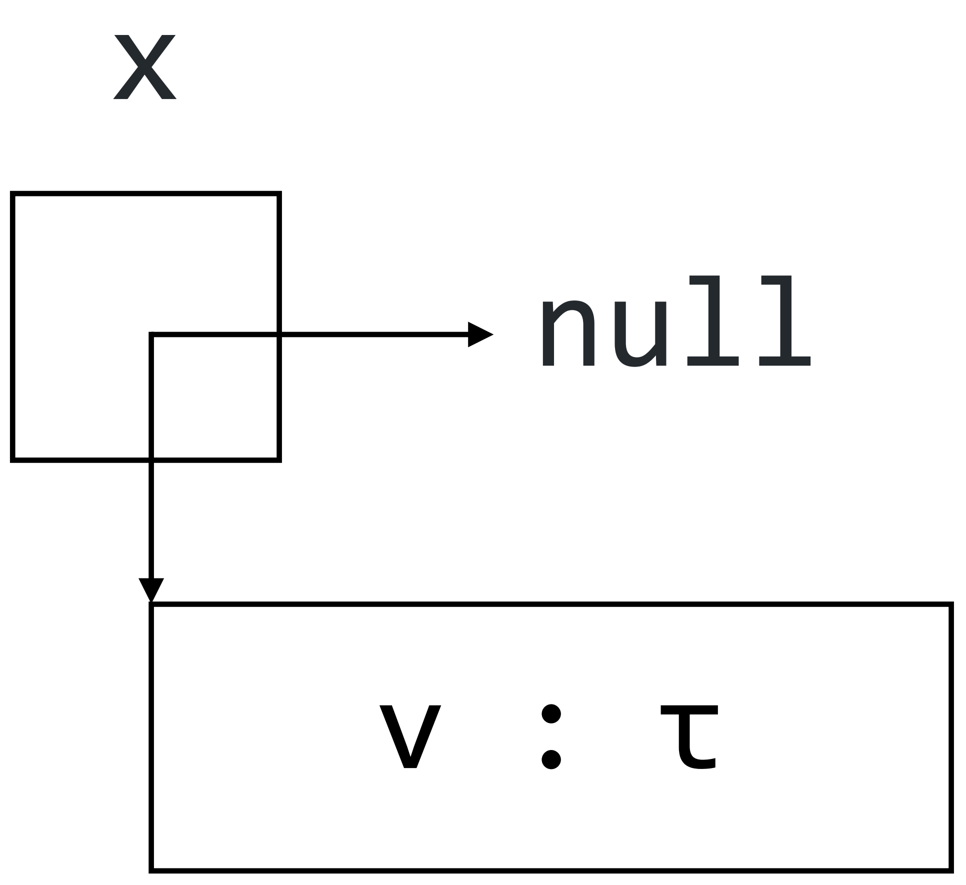
$$Rep(\tau\ *\ \mathsf{SAFE}) = Rep(\tau)\ *$$




$$\frac{x : \tau\ *\ \mathsf{SAFE} \qquad y : \tau}{*x = y \rightsquigarrow \qquad\qquad *x = y}$$

# Safe pointers need null checks.

$$Rep(\tau \;*\; \mathsf{SAFE}) = Rep(\tau) \;*$$

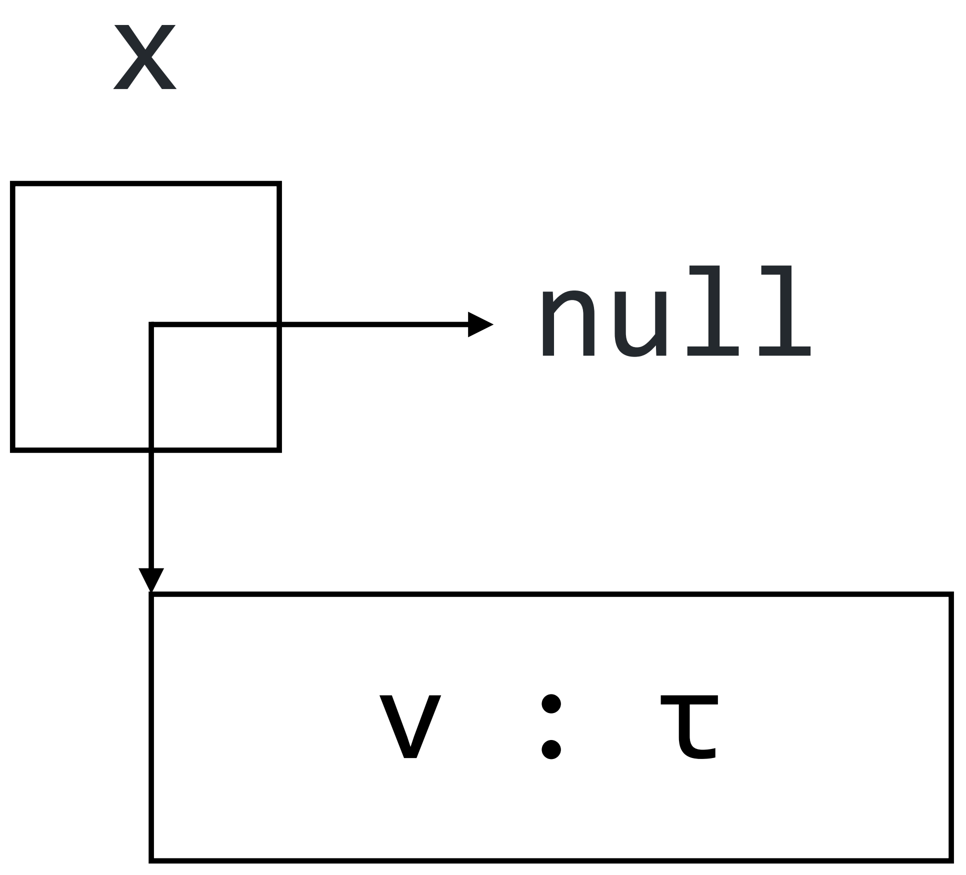



$$\frac{x : \tau \;*\; \mathsf{SAFE} \qquad y : \tau}{*x = y \rightsquigarrow \mathtt{assert}(x \neq \mathtt{null}); *x = y}$$

# Zero can be casted to a safe pointer.

$$Rep(\tau\ *\ \mathsf{SAFE}) = Rep(\tau)\ *$$

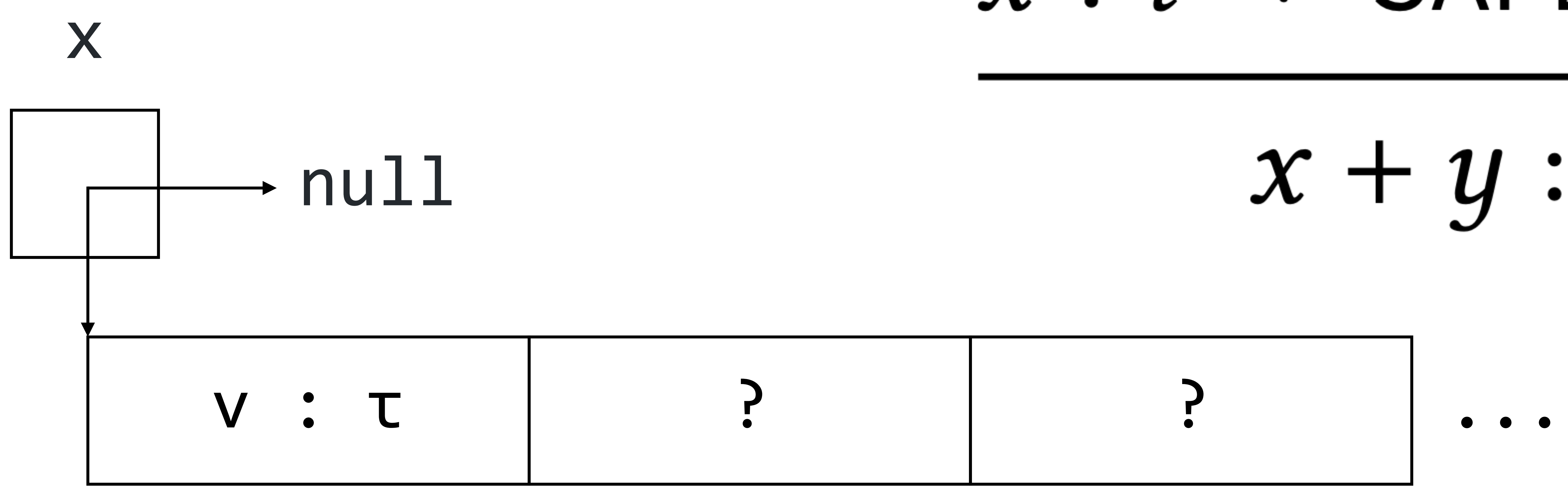
$$(\tau\ *\ \mathsf{SAFE})0 \rightsquigarrow \mathtt{null}$$




$$\frac{x : \mathtt{int} \qquad x \neq 0}{(\tau\ *\ \mathsf{SAFE})x : \text{not ok}}$$

# Pointer arithmetic is disallowed for safe pointers.

$$Rep(\tau \ * \ \mathsf{SAFE}) = Rep(\tau) \, *$$

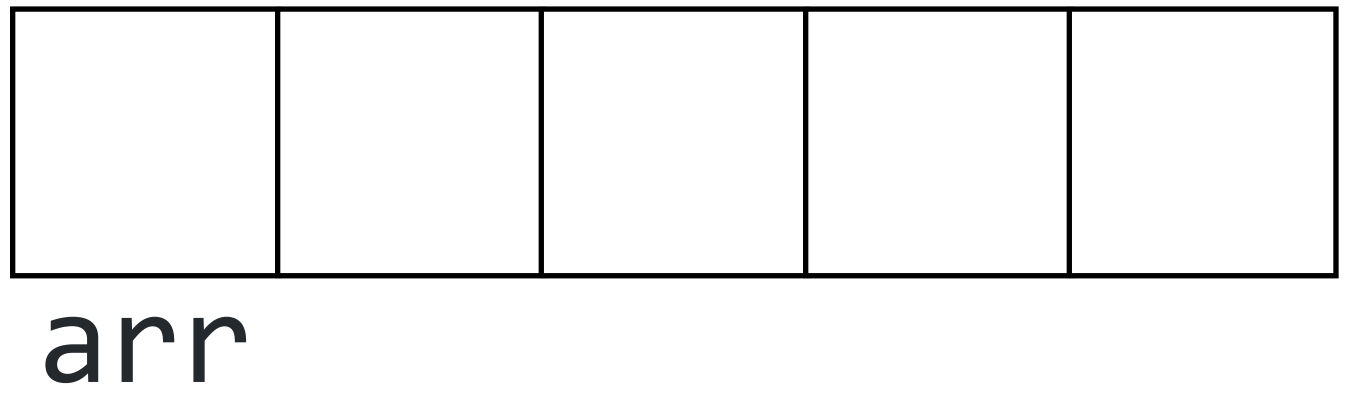



$$\frac{x : \tau \ * \ \mathsf{SAFE} \qquad y : \mathtt{int}}{x + y : \text{not ok}}$$

# Sequence pointers refer to elements in arrays.

$$Rep(\tau \ * \ \mathrm{SEQ}) = \texttt{struct}\{Rep(\tau) \ * \ p, b, e; \}$$

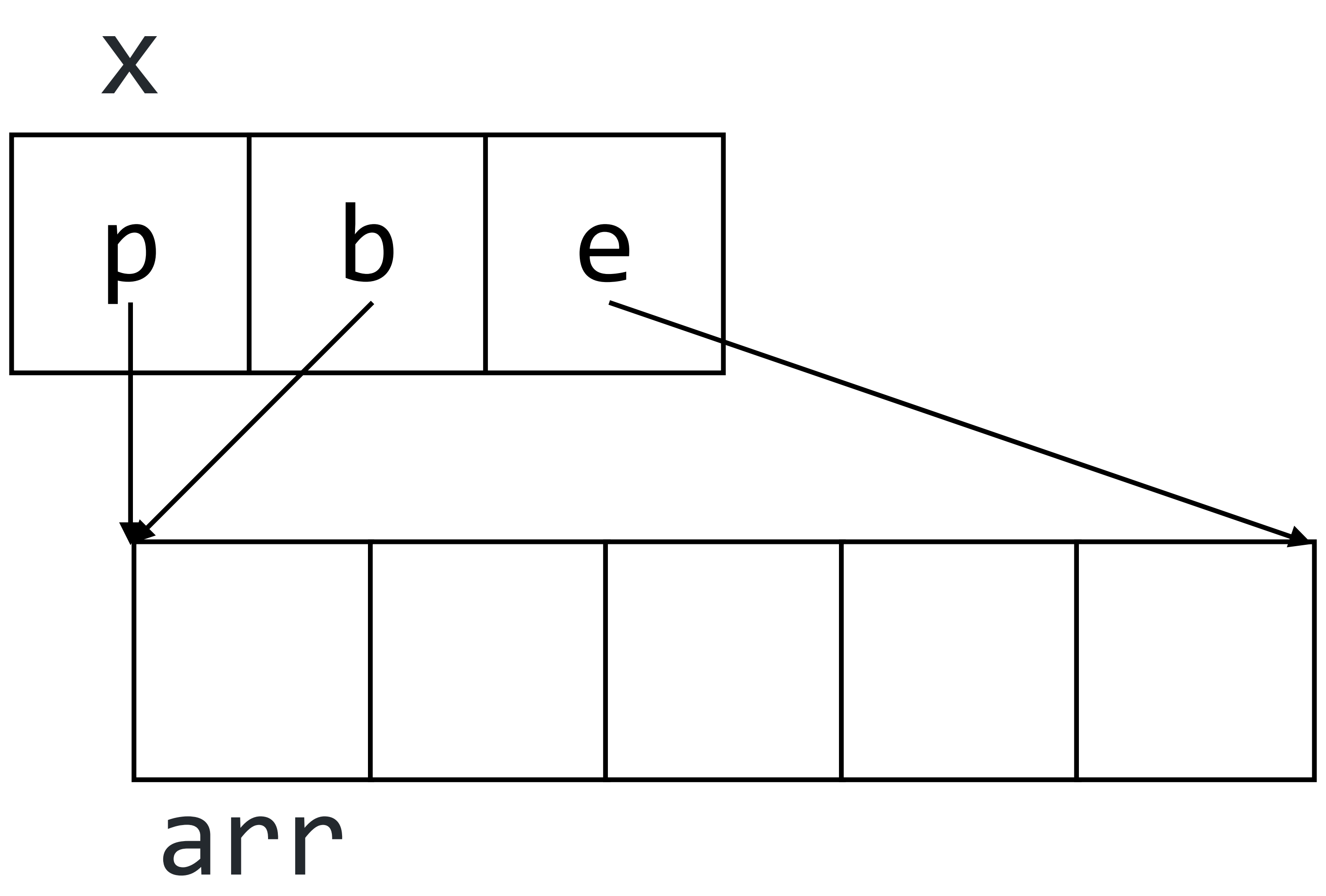
```
int arr[5];
```



# Sequence pointers refer to elements in arrays.

$$Rep(\tau\ *\ \mathsf{SEQ}) = \texttt{struct}\{Rep(\tau)\ *\ p, b, e;\}$$

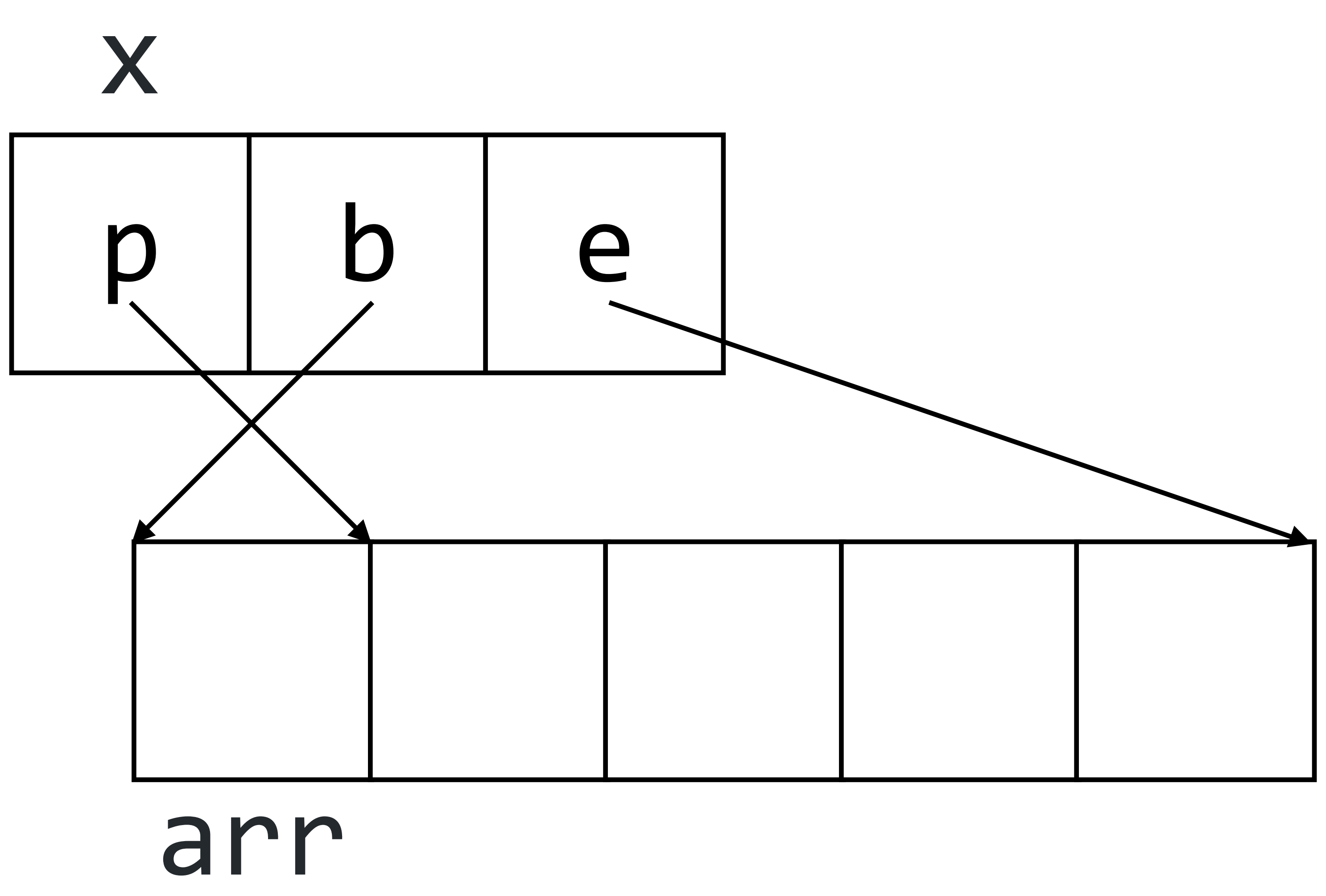
```
int arr[5];
int * SEQ x = arr;
```



# Sequence pointers refer to elements in arrays.

$$Rep(\tau \ * \ \mathsf{SEQ}) = \texttt{struct}\{Rep(\tau) \ * \ p, b, e;\}$$

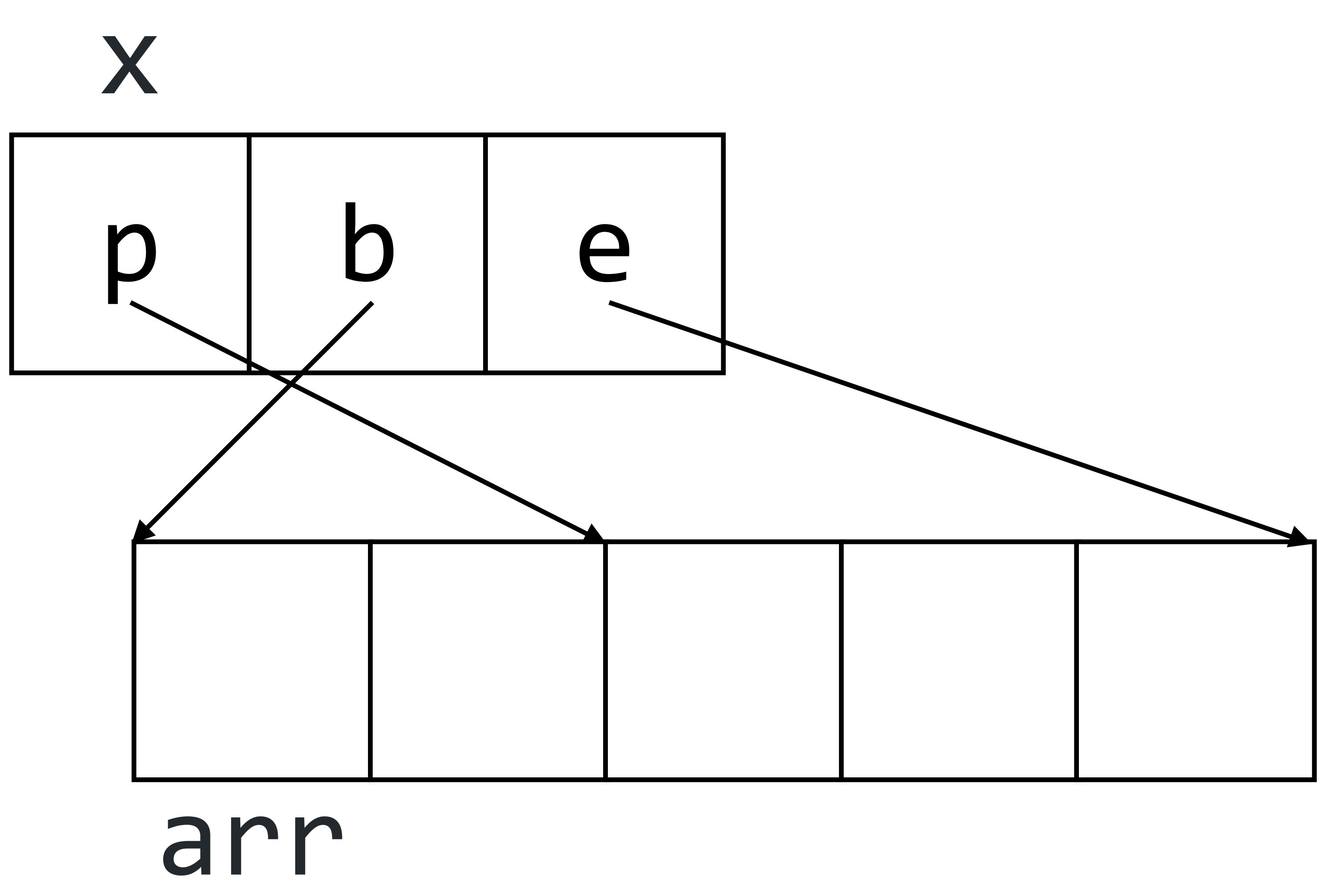
```
int arr[5];
int * SEQ x = arr;
x++;
```



# Sequence pointers refer to elements in arrays.

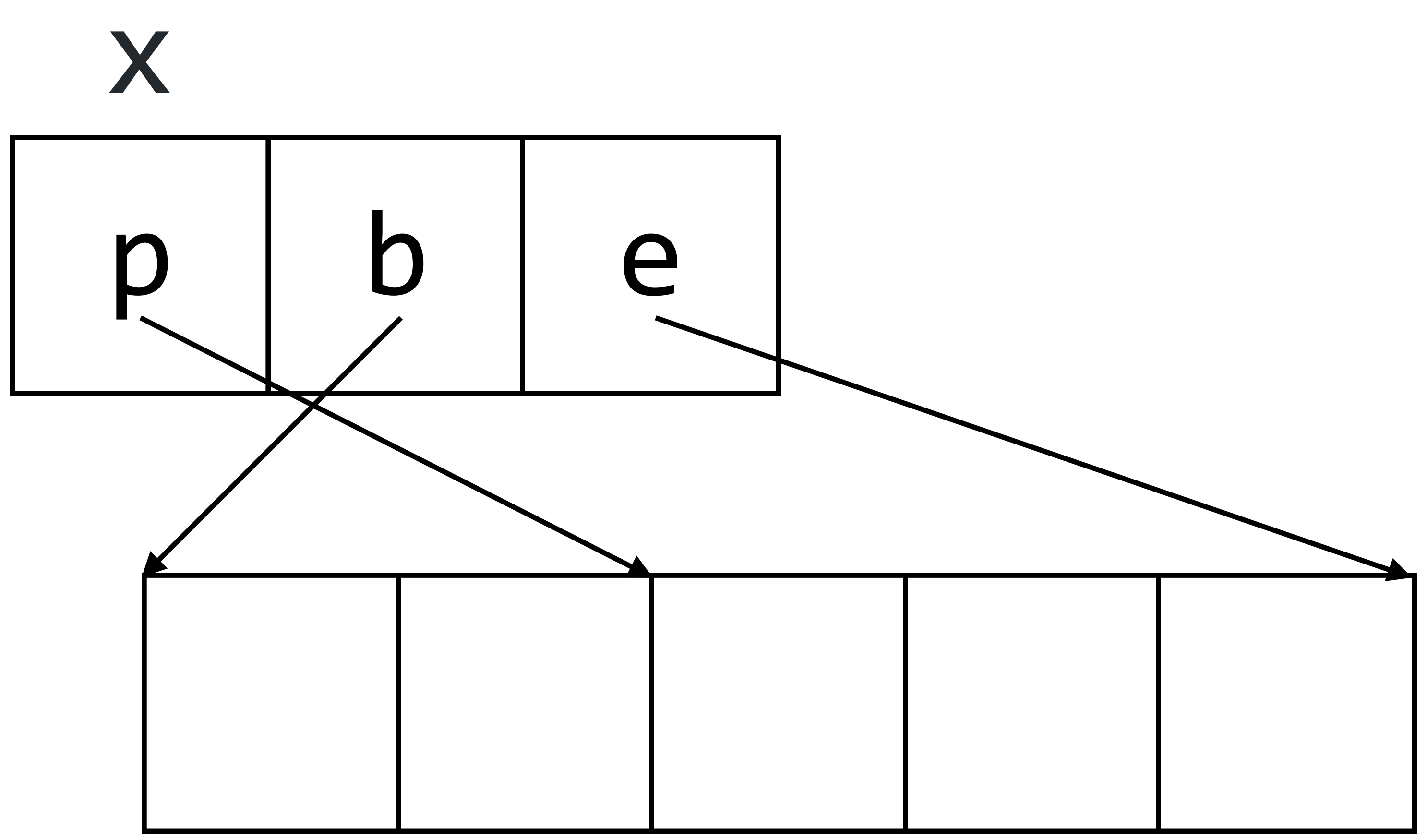
$$Rep(\tau * \mathsf{SEQ}) = \texttt{struct}\{Rep(\tau) * p, b, e;\}$$

```
int arr[5];
int * SEQ x = arr;
x++; x++;
```



# Sequence pointers need bounds checks.

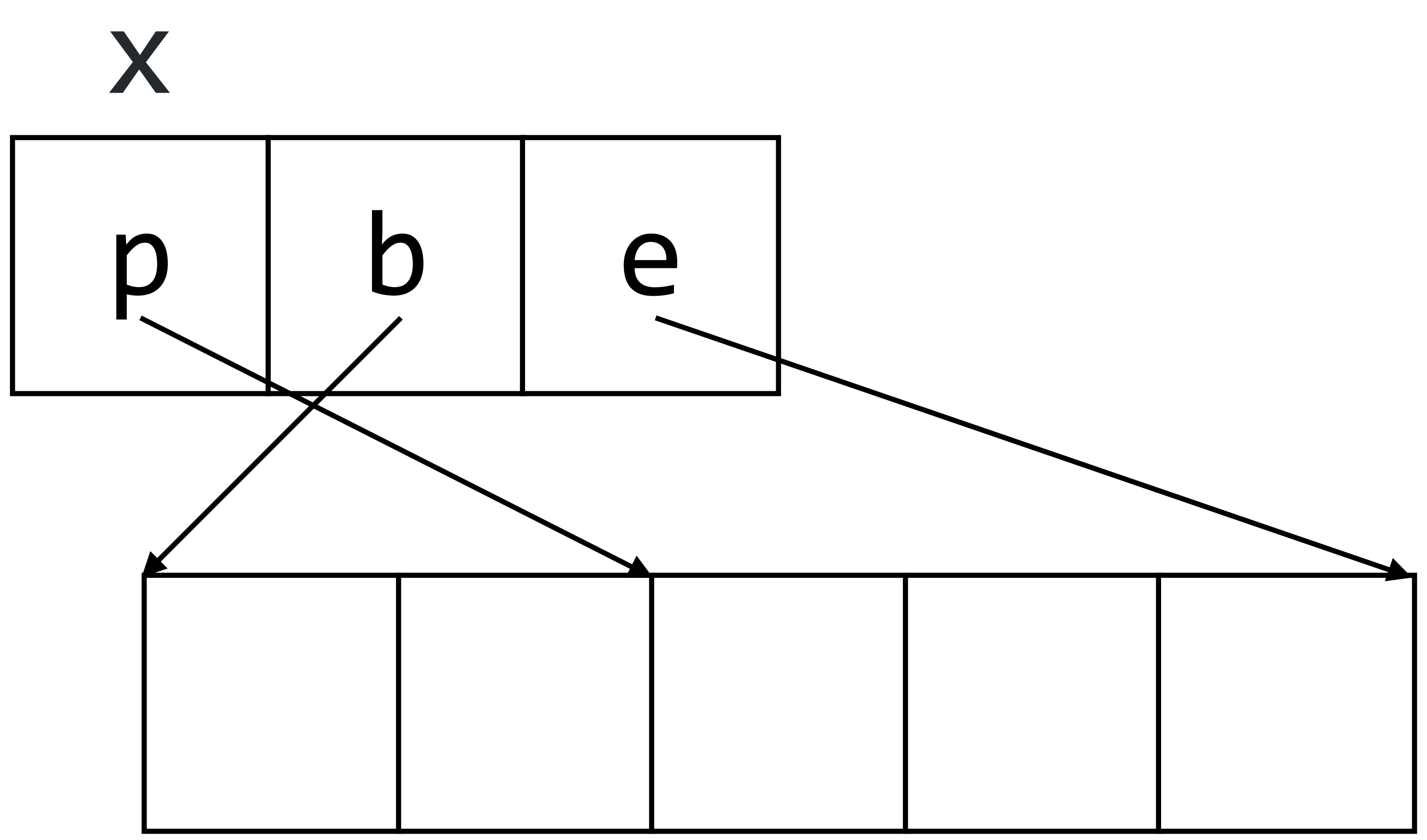
$$Rep(\tau \ * \ \mathtt{SEQ}) = \mathtt{struct}\{Rep(\tau) \ * \ p, b, e;\}$$




$$\frac{x : \tau \ * \ \mathtt{SEQ}}{*x \rightsquigarrow \ *x.p}$$

# Sequence pointers need bounds checks.

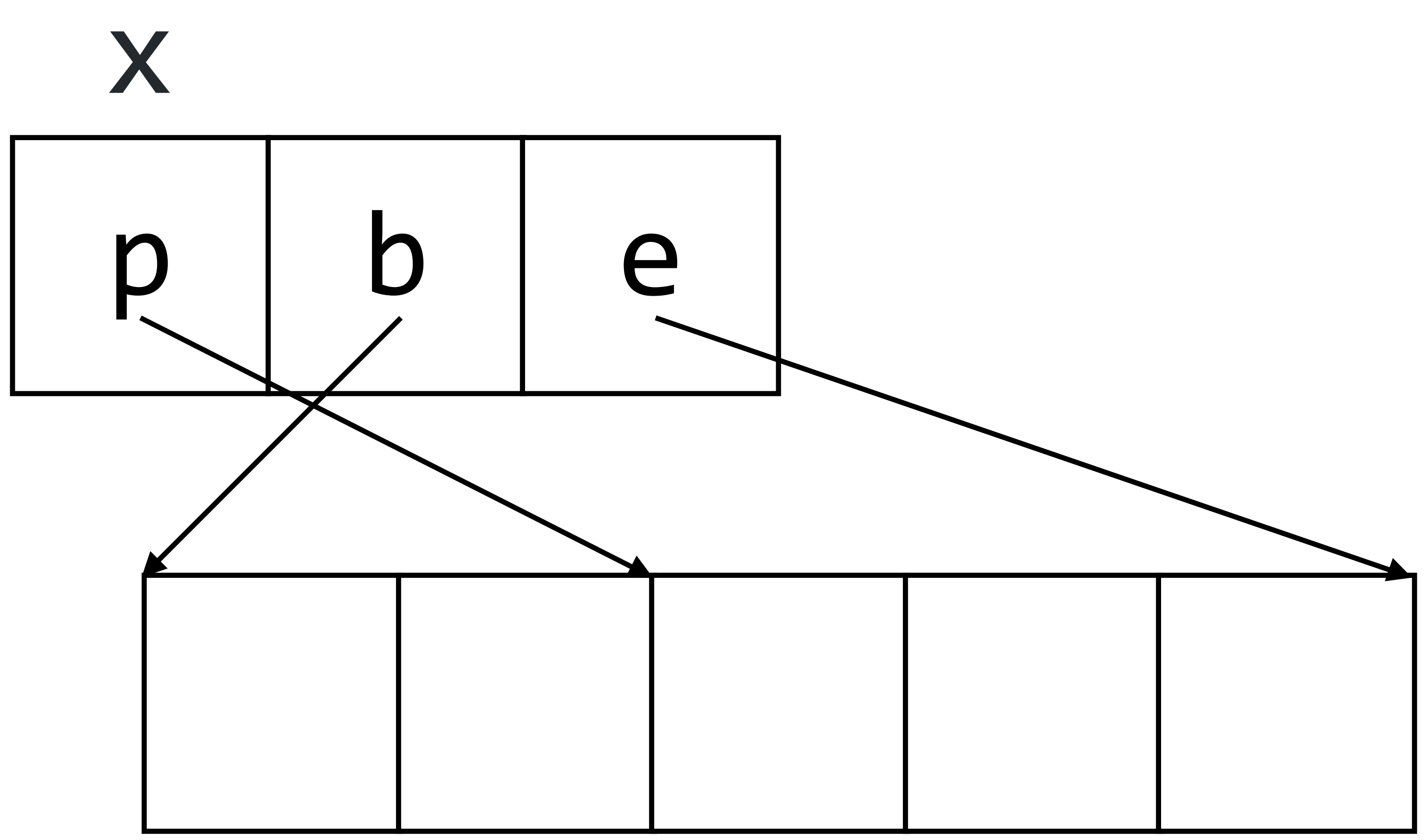
$$Rep(\tau \ * \ \texttt{SEQ}) = \texttt{struct}\{Rep(\tau) \ * \ p, b, e;\}$$




$$\frac{x : \tau \ * \ \texttt{SEQ}}{*x \rightsquigarrow \begin{array}{l} \texttt{assert}(x.b \leq x.p \leq x.e - \texttt{sizeof}(\tau)); \\ *x.p \end{array}}$$

# Sequence pointers need bounds checks.

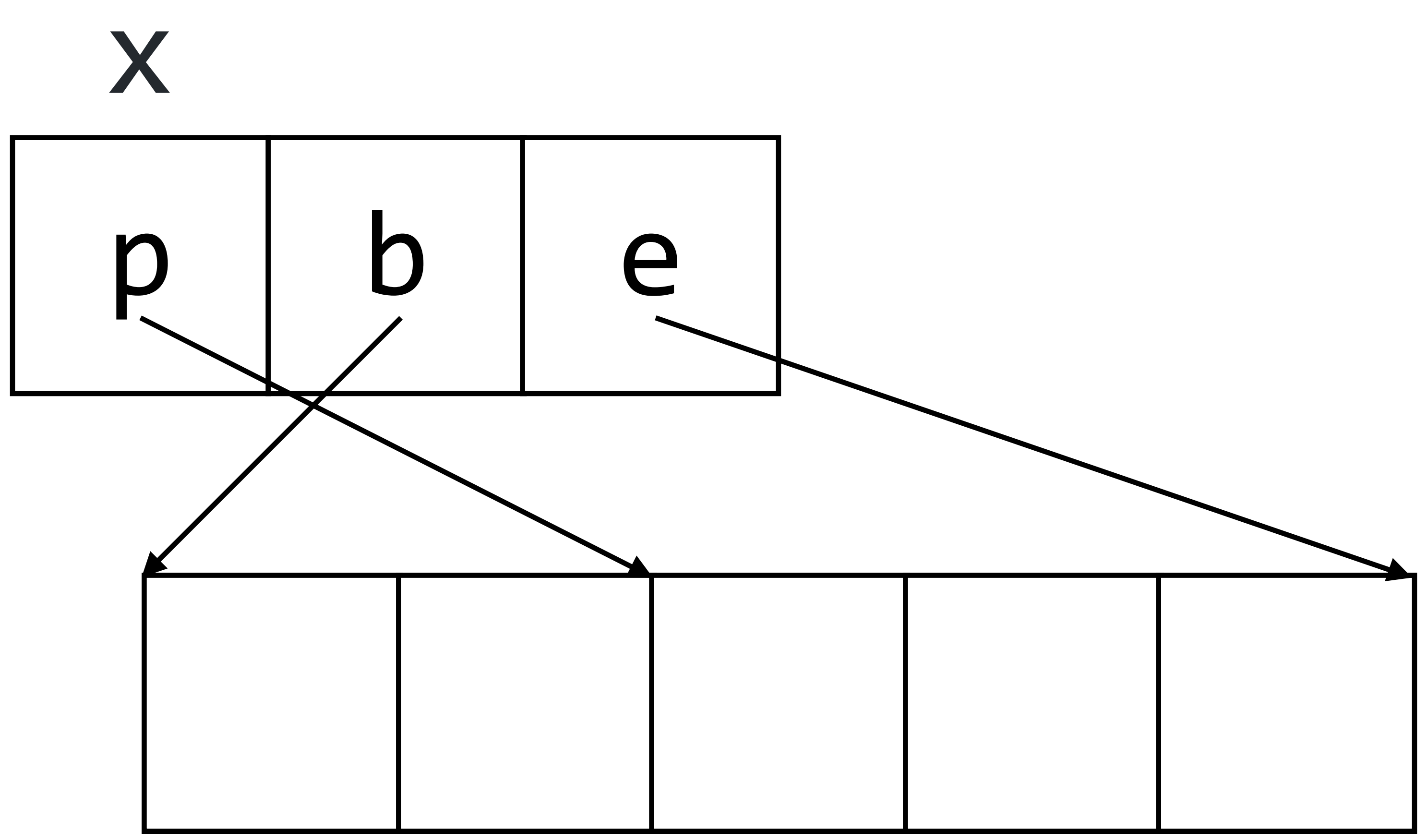
$$Rep(\tau \; * \; \mathtt{SEQ}) = \mathtt{struct}\{Rep(\tau) \; * \; p, b, e; \}$$




$$\frac{x : \tau \; * \; \mathtt{SEQ} \qquad y : \tau}{*x = y \leadsto \quad *x.p = y}$$

# Sequence pointers need bounds checks.

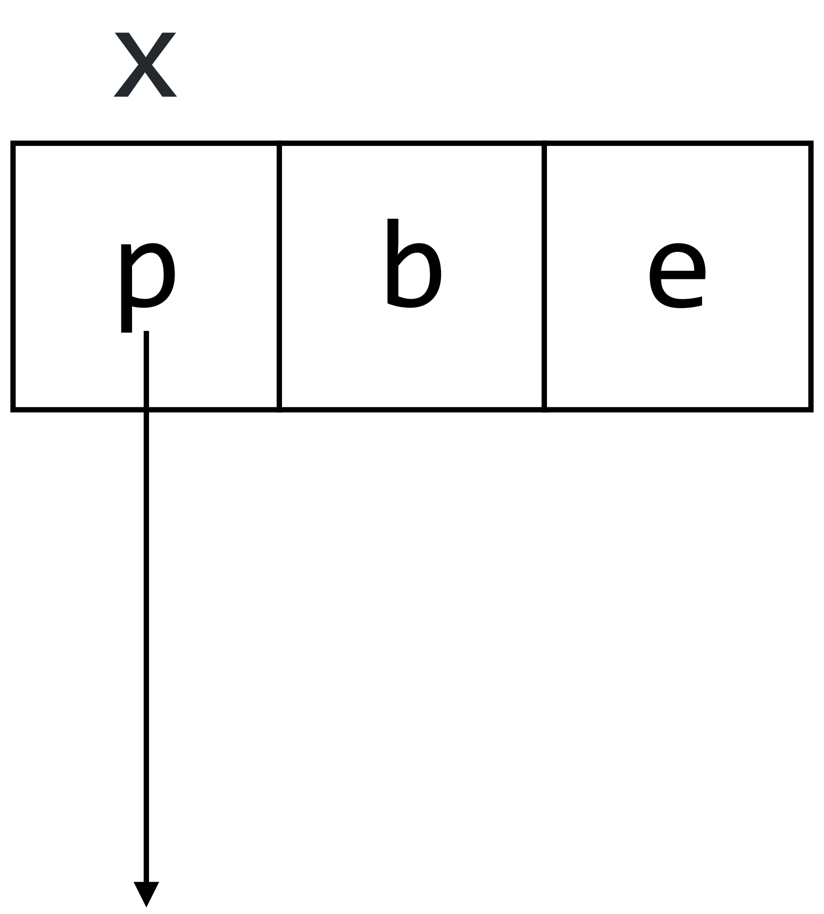
$$\mathit{Rep}(\tau \ * \ \mathtt{SEQ}) = \mathtt{struct}\{\mathit{Rep}(\tau) \ * \ p, b, e;\}$$




$$\frac{x : \tau \ * \ \mathtt{SEQ} \qquad y : \tau}{*x = y \leadsto \begin{array}{l} \mathtt{assert}(x.b \leq x.p \leq x.e - \mathtt{sizeof}(\tau)); \\ *x.p = y \end{array}}$$

# Integers can be casted to sequence pointers.

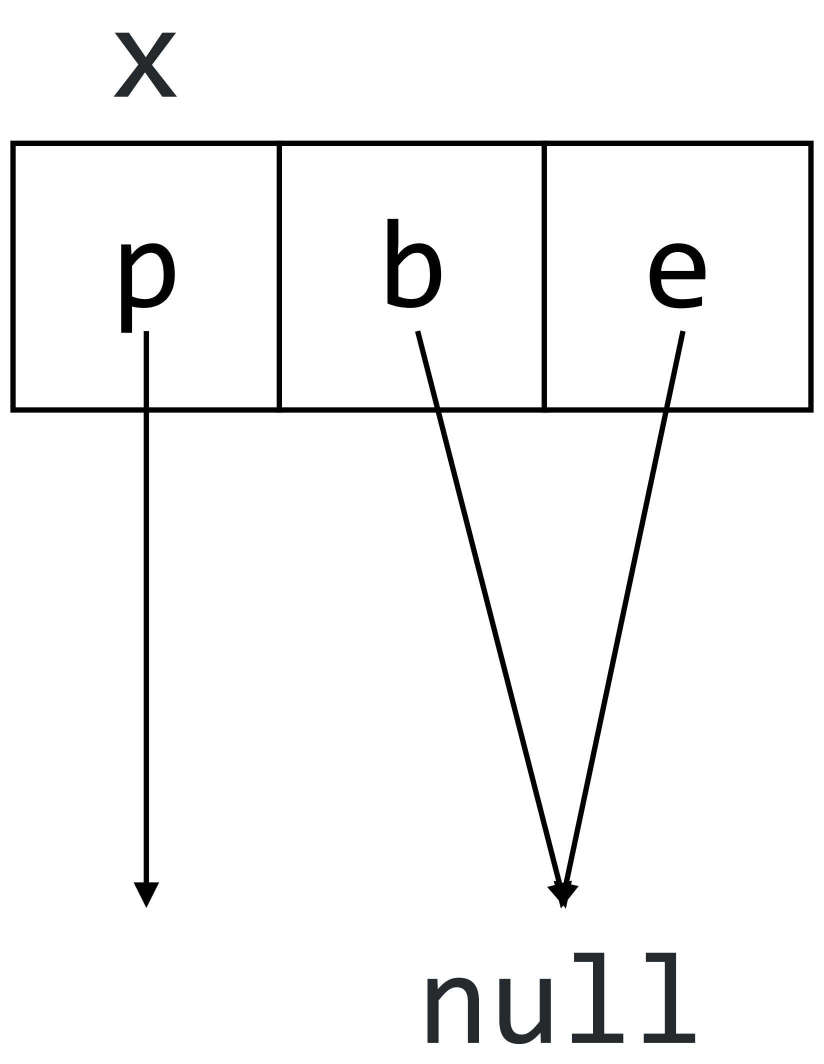
$$Rep(\tau \;*\; \mathtt{SEQ}) = \mathtt{struct}\{Rep(\tau) \;*\; p, b, e;\}$$




$$\frac{x : \mathtt{int}}{(\tau \;*\; \mathtt{SEQ})x \rightsquigarrow \{p = x \qquad\qquad \}}$$

# Integers can be casted to sequence pointers.

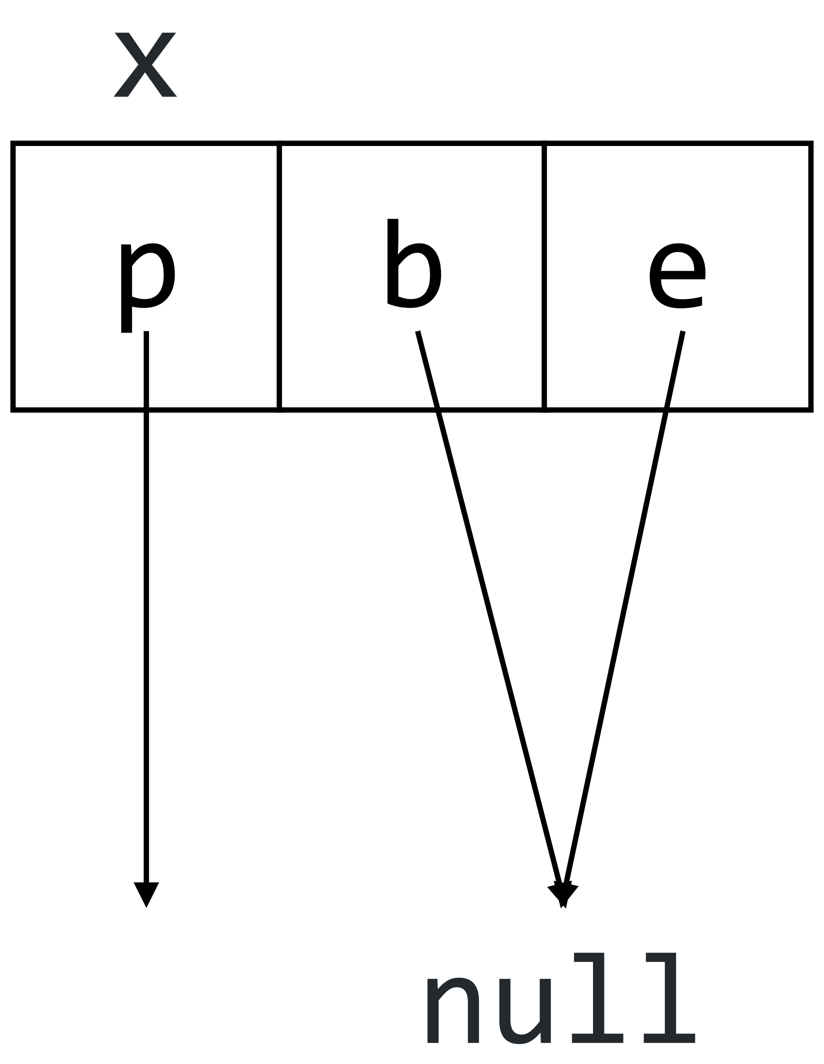
$$Rep(\tau \ * \ \texttt{SEQ}) = \texttt{struct}\{Rep(\tau) \ * \ p, b, e;\}$$




$$\frac{x : \texttt{int}}{(\tau \ * \ \texttt{SEQ})x \rightsquigarrow \{p = x, b = \texttt{null}, e = \texttt{null}\}}$$

# Integers can be casted to sequence pointers.

$$Rep(\tau * \mathsf{SEQ}) = \mathtt{struct}\{Rep(\tau) * p, b, e;\}$$

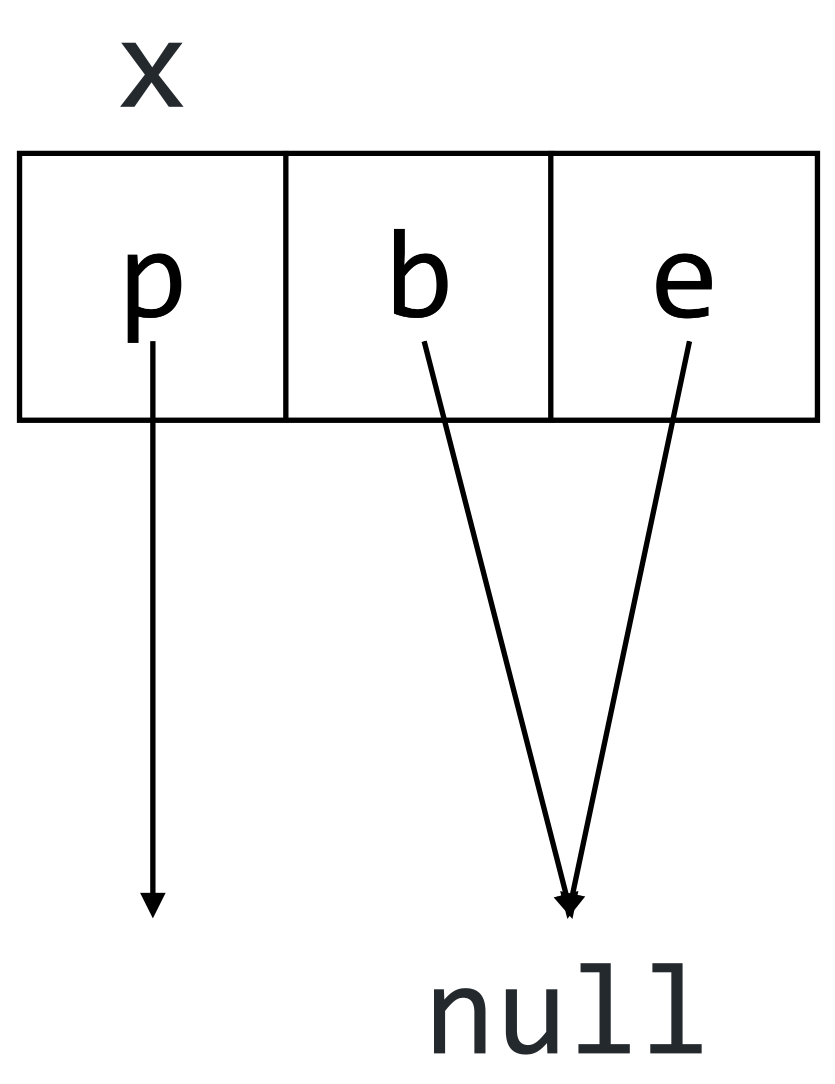



$$\frac{x : \mathtt{int}}{(\tau * \mathsf{SEQ})x \leadsto \{p = x, b = \mathtt{null}, e = \mathtt{null}\}}$$

```
int n = N;
int * SEQ x = (int * SEQ) n;
*x = M;
```

# Integers can be casted to sequence pointers.

$$Rep(\tau \; * \; \texttt{SEQ}) = \texttt{struct}\{Rep(\tau) \; * \; p, b, e;\}$$



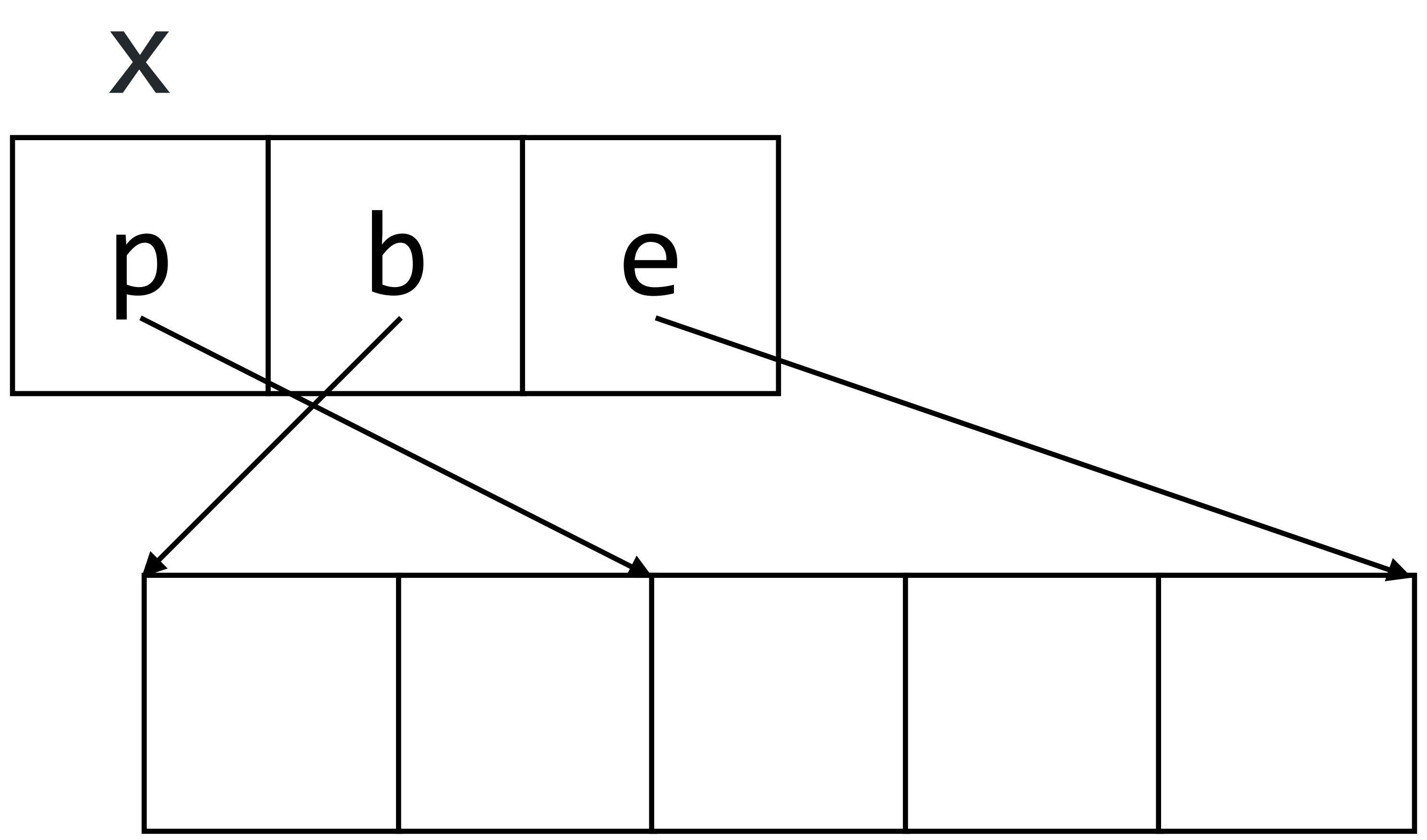

$$\frac{x : \texttt{int}}{(\tau \; * \; \texttt{SEQ})x \rightsquigarrow \{p = x, b = \texttt{null}, e = \texttt{null}\}}$$

```
int n = N;
int * SEQ x = (int * SEQ) n;
int m = (int) x;
```

$$\frac{x : \tau \; * \; \texttt{SEQ}}{(\texttt{int})x \rightsquigarrow x.p}$$

# Pointer arithmetic is allowed for sequence pointers.

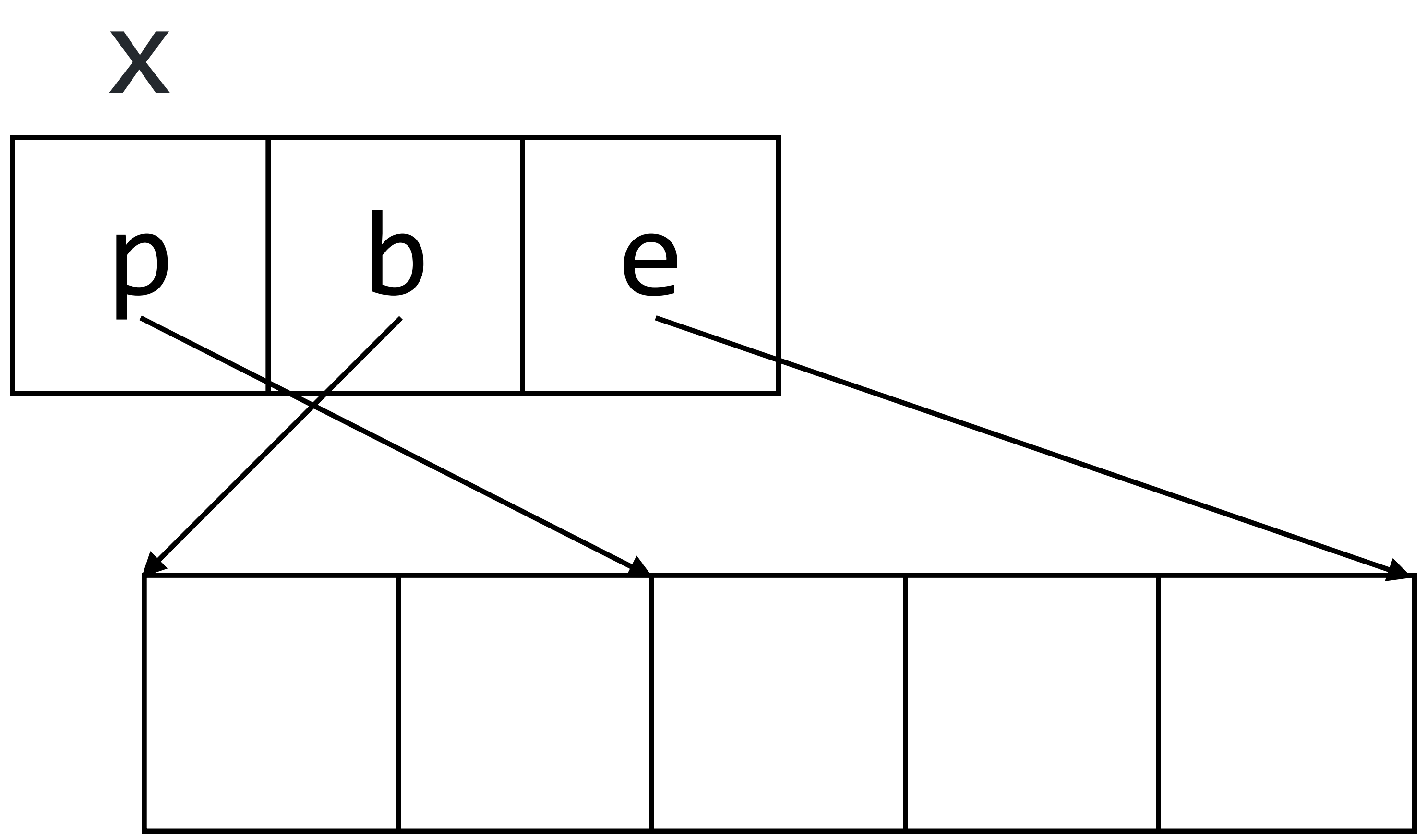
$$Rep(\tau\ *\ \mathtt{SEQ}) = \mathtt{struct}\{Rep(\tau)\ *\ p, b, e;\}$$




$$\frac{x : \tau\ *\ \mathtt{SEQ} \qquad y : \mathtt{int}}{x + y \rightsquigarrow \{p = x.p + y \times \mathtt{sizeof}(\tau), b = x.b, e = x.e\}}$$

# Sequence pointers can be casted to safe pointers.

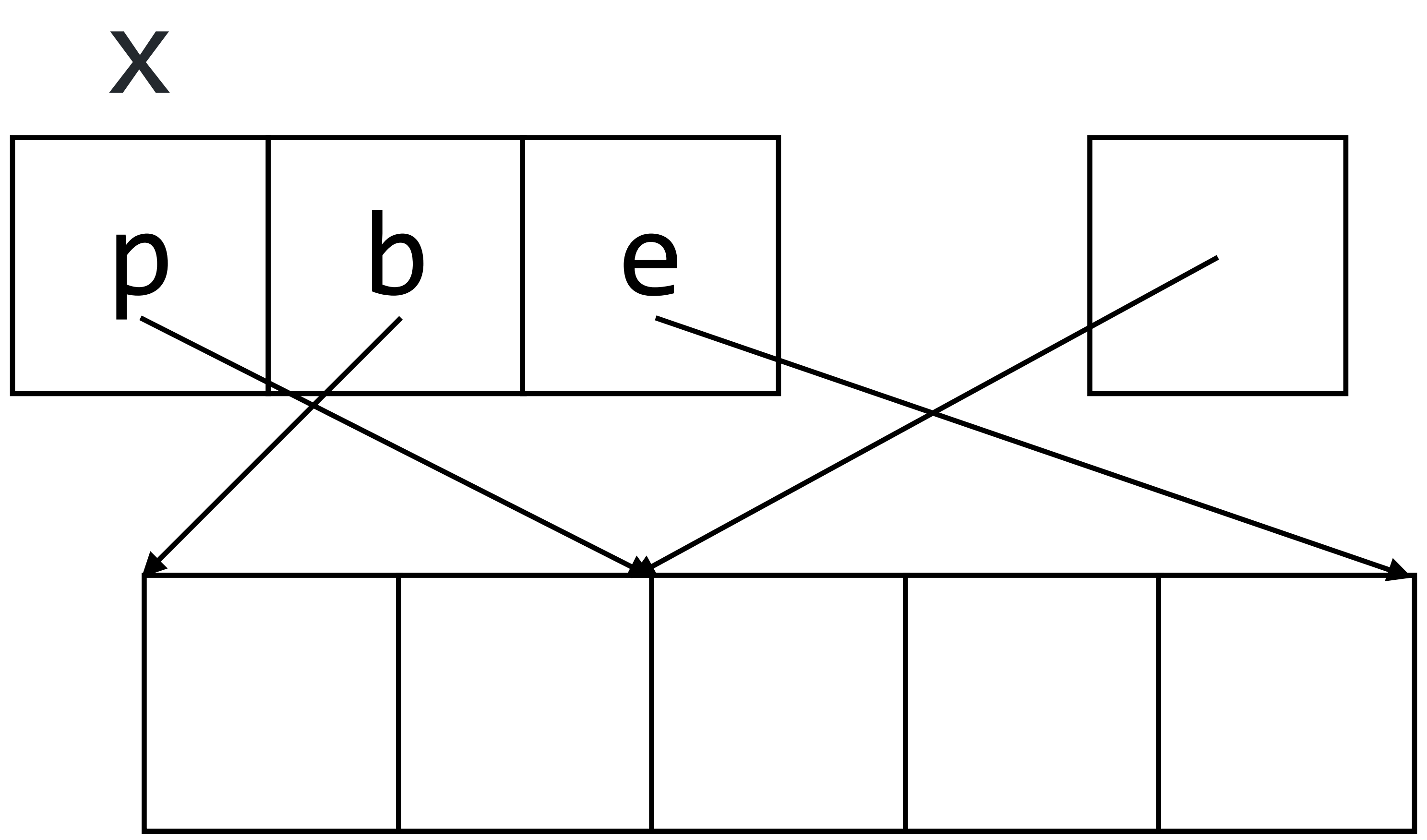
$$Rep(\tau\ *\ \mathtt{SEQ}) = \mathtt{struct}\{Rep(\tau)\ *\ p, b, e;\}$$




$$\frac{x : \tau\ *\ \mathtt{SEQ}}{(\tau\ *\ \mathtt{SAFE})x \rightsquigarrow}$$

# Sequence pointers can be casted to safe pointers.

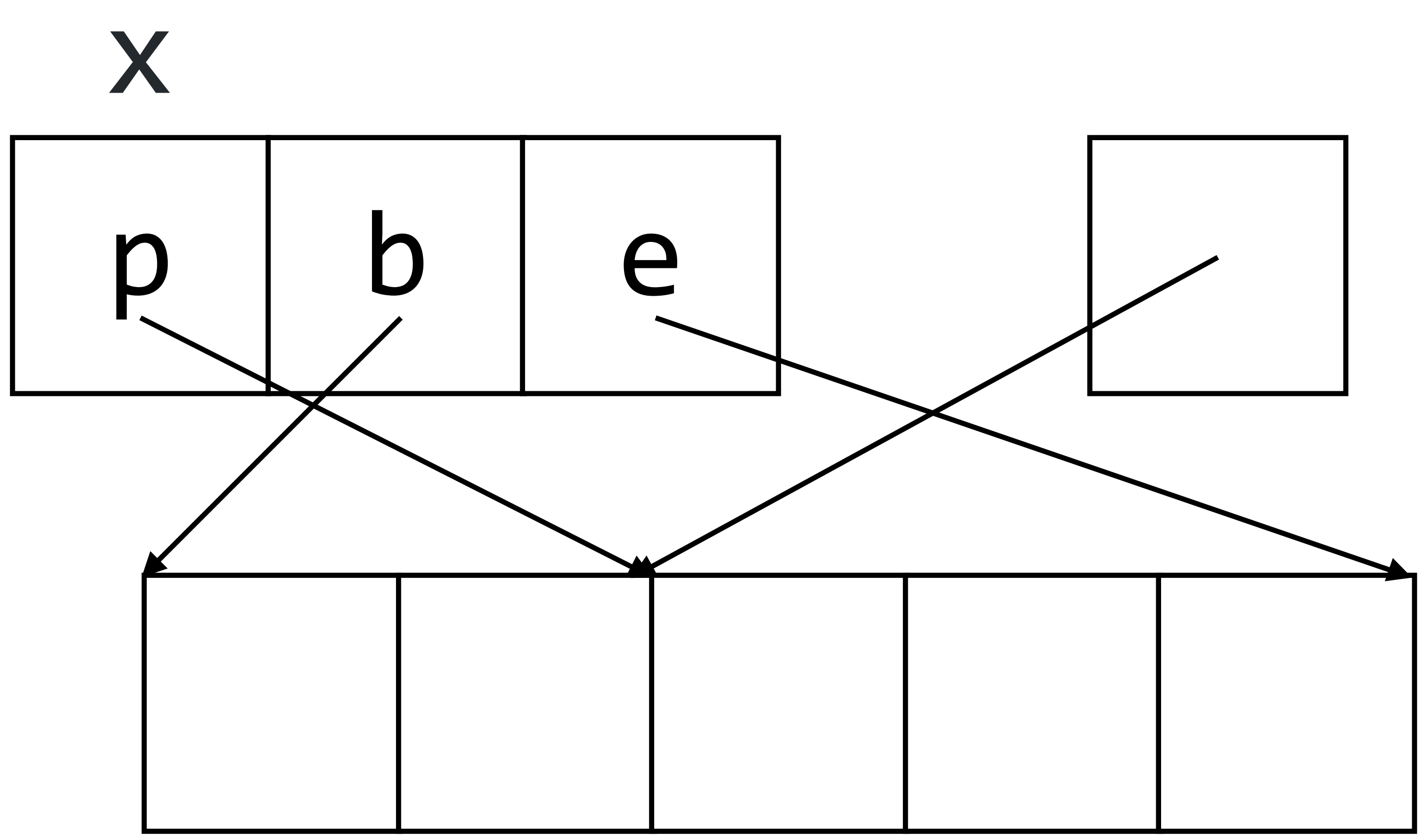
$$Rep(\tau\ *\ \mathtt{SEQ}) = \mathtt{struct}\{Rep(\tau)\ *\ p, b, e;\}$$




$$\frac{x : \tau\ *\ \mathtt{SEQ}}{(\tau\ *\ \mathtt{SAFE})x \rightsquigarrow\ x.p}$$

# Sequence pointers can be casted to safe pointers.

$$Rep(\tau \ * \ \mathtt{SEQ}) = \mathtt{struct}\{Rep(\tau) \ * \ p, b, e;\}$$




$$\frac{x : \tau \ * \ \mathtt{SEQ}}{(\tau \ * \ \mathtt{SAFE})x \rightsquigarrow \begin{array}{l}\mathtt{assert}(x.p = \mathrm{null} \lor x.b \leq x.p \leq x.e - \mathtt{sizeof}(\tau)); \\ x.p\end{array}}$$

# Wild pointers can refer to any values.

```
struct a { int x, y; };
struct b { int *p, n; };

struct b sb;
struct a *sa = (struct a *) &sb;
sa->y = N;
sb.n;
```

# Wild pointers can refer to any values.

```c
struct a { int x, y; };
struct b { int *p, n; };

struct b sb;
struct a *sa = (struct a *) &sb;
sa->y = N;
sb.n;
```

$$\frac{x : \tau \ * \ \mathsf{SAFE} \qquad \tau \neq \tau'}{(\tau' \ * \ \mathsf{SAFE})x : \text{not ok}}$$

$$\frac{x : \tau \ * \ \mathsf{SEQ} \qquad \tau \neq \tau'}{(\tau' \ * \ \mathsf{SEQ})x : \text{not ok}}$$

# Wild pointers can refer to any values.

```
struct a { int x, y; };
struct b { int *p, n; };

struct b sb;
struct a *sa = (struct a *) &sb;
sa->y = N;
sb.n;
```

$$\frac{x : \tau \;*\; \mathsf{SAFE} \qquad \tau \neq \tau'}{(\tau' \;*\; \mathsf{SAFE})x : \text{not ok}}$$

$$\frac{x : \tau \;*\; \mathsf{SEQ} \qquad \tau \neq \tau'}{(\tau' \;*\; \mathsf{SEQ})x : \text{not ok}}$$

$$\frac{x : \tau \;*\; \mathtt{WILD}}{(\tau' \;*\; \mathtt{WILD})x \rightsquigarrow x}$$

# Wild pointers can refer to any values.

```c
struct a { int x, y; };
struct b { int *p, n; };

struct b sb;
struct a *sa = (struct a *) &sb;
sa->x = N;
*sb.p = M;
```

$$\frac{x : \tau \ast \mathsf{SAFE} \qquad \tau \neq \tau'}{(\tau' \ast \mathsf{SAFE})x : \text{not ok}}$$

$$\frac{x : \tau \ast \mathsf{SEQ} \qquad \tau \neq \tau'}{(\tau' \ast \mathsf{SEQ})x : \text{not ok}}$$

$$\frac{x : \tau \ast \mathtt{WILD}}{(\tau' \ast \mathtt{WILD})x \rightsquigarrow x}$$

# Wild pointers can refer to any values.

```
int * SEQ foo;
int * SEQ * WILD p = &foo;
bool * SEQ * WILD q =
    (bool * SEQ * WILD) p;
bool * SEQ bar = *q;
```

$$\frac{x : \tau \ * \ \mathsf{SAFE} \qquad \tau \neq \tau'}{(\tau' \ * \ \mathsf{SAFE})x : \text{not ok}}$$

$$\frac{x : \tau \ * \ \mathsf{SEQ} \qquad \tau \neq \tau'}{(\tau' \ * \ \mathsf{SEQ})x : \text{not ok}}$$

$$\frac{x : \tau \ * \ \mathtt{WILD}}{(\tau' \ * \ \mathtt{WILD})x \rightsquigarrow x}$$

# Wild pointers can refer to any values.

```
int * SEQ foo;
int * SEQ * WILD p = &foo;
bool * SEQ * WILD q =
    (bool * SEQ * WILD) p;
bool * SEQ bar = *q;
```

$$\frac{x : \tau \ * \ \mathsf{SAFE} \qquad \tau \neq \tau'}{(\tau' \ * \ \mathsf{SAFE})x : \text{not ok}}$$

$$\frac{x : \tau \ * \ \mathsf{SEQ} \qquad \tau \neq \tau'}{(\tau' \ * \ \mathsf{SEQ})x : \text{not ok}}$$

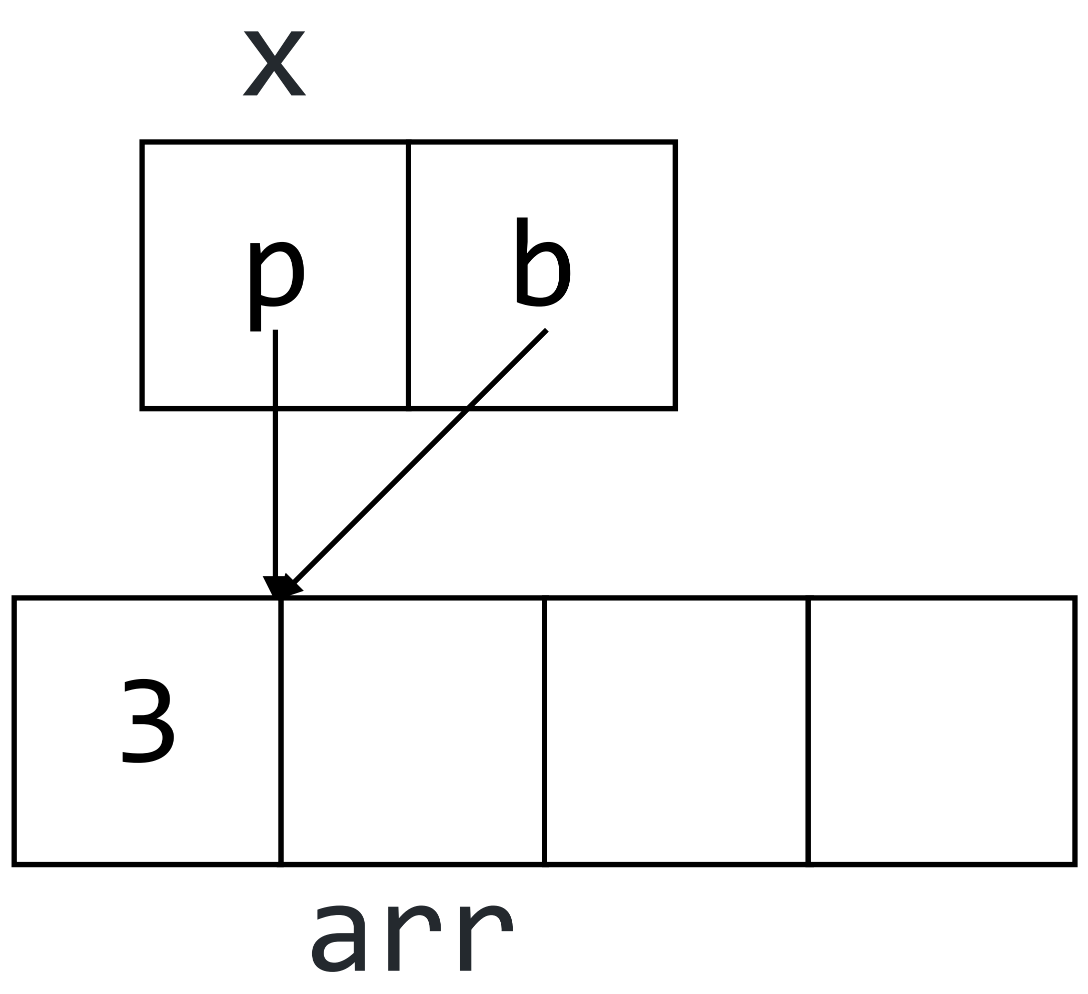
$$\frac{q \neq \mathtt{WILD}}{\tau \ * \ q \ * \ \mathtt{WILD} : \text{wrong type}}$$

$$\frac{x : \tau \ * \ \mathtt{WILD}}{(\tau' \ * \ \mathtt{WILD})x \rightsquigarrow x}$$

# Wild pointers can refer to any values.

$$Rep(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \ * \ p, b;\}$$

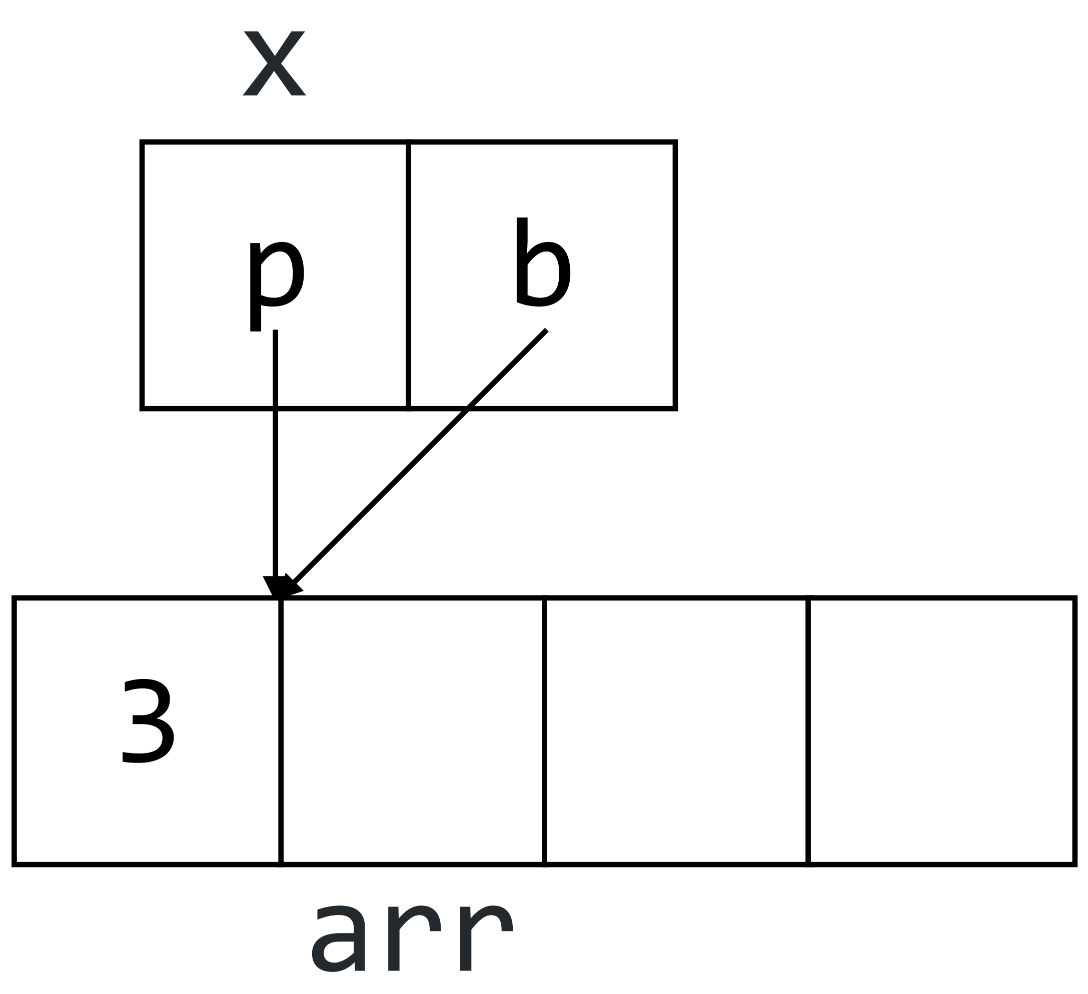
```
int arr[3];
int * WILD x = arr;
```



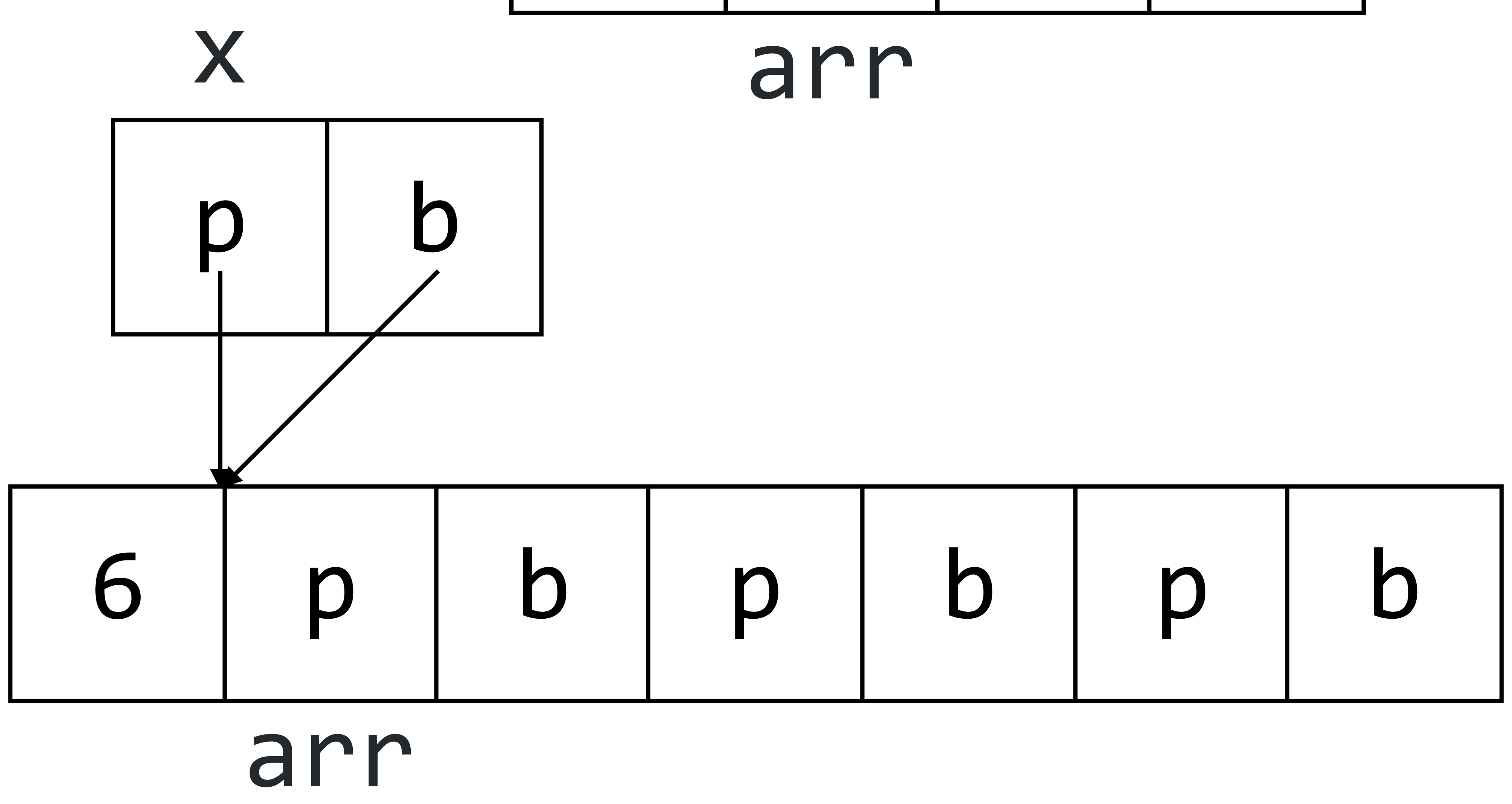
# Wild pointers can refer to any values.

$$Rep(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \ * \ p, b;\}$$

```
int arr[3];
int * WILD x = arr;
```



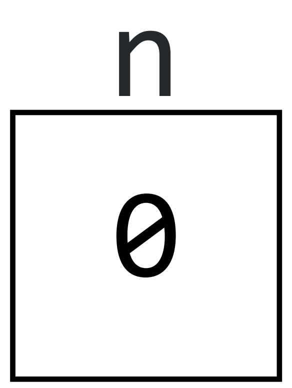

```
int (* WILD arr)[3];
int * WILD * WILD x = arr;
```



# Wild pointers can refer to any values.

$$Rep(\tau \; * \; \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \; * \; p, b;\}$$

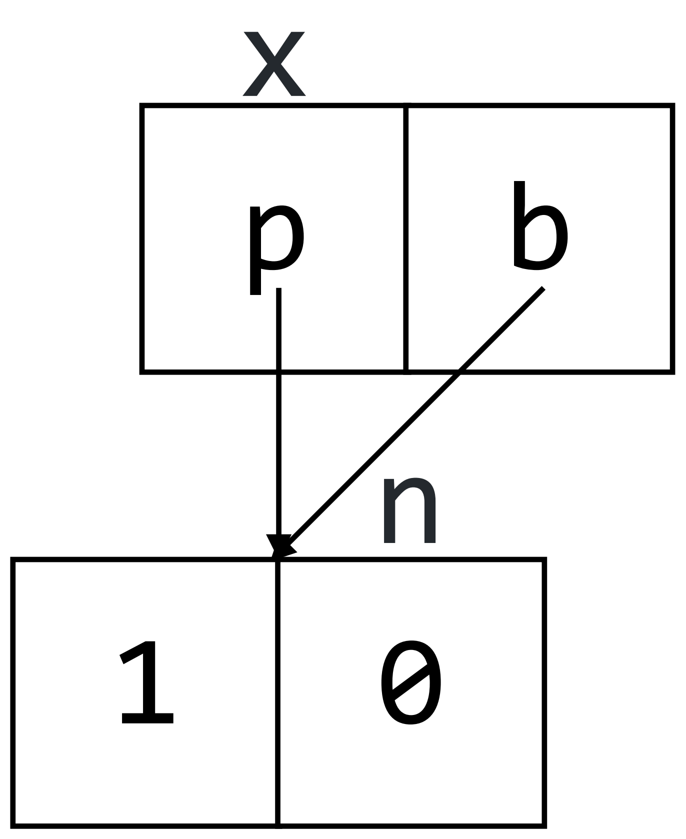
```c
int n = 0;
```



# Wild pointers can refer to any values.

$$Rep(\tau \ * \ \mathtt{WILD}) = \mathtt{struct}\{Rep(\tau) \ * \ p, b;\}$$

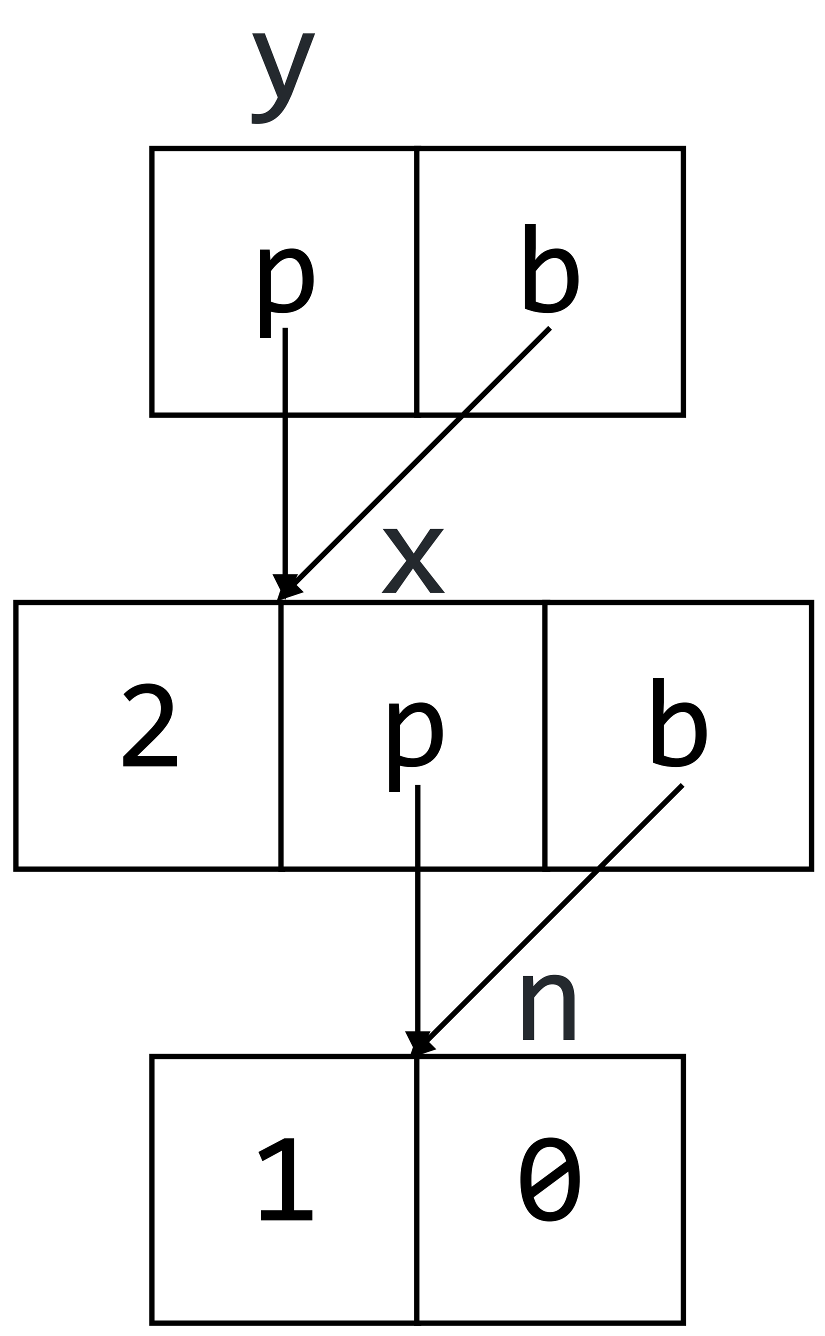
```c
int n = 0;
int * WILD x = &n;
```



# Wild pointers can refer to any values.

$$Rep(\tau * \texttt{WILD}) = \texttt{struct}\{Rep(\tau) * p, b;\}$$

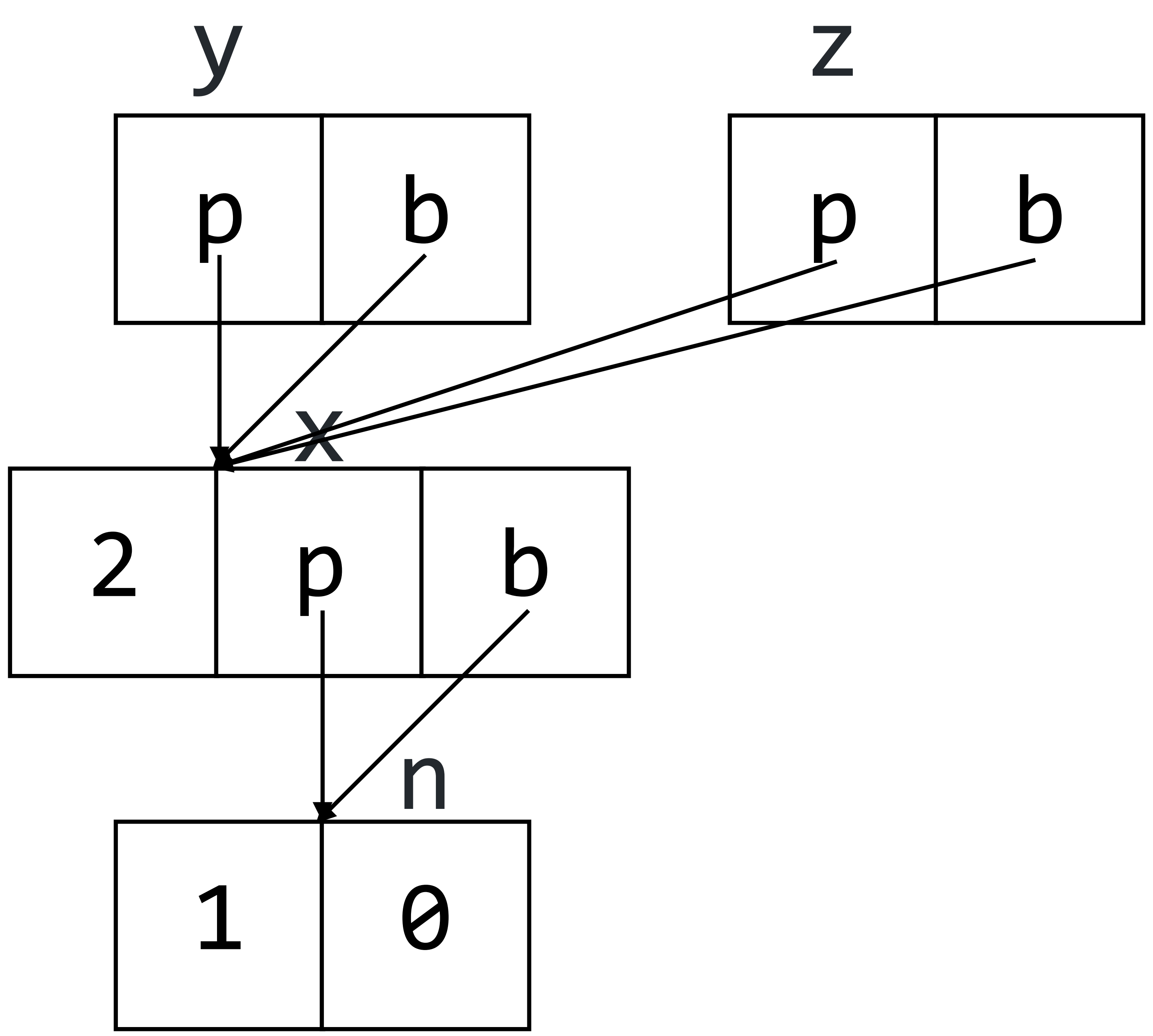
```
int n = 0;
int * WILD x = &n;
int * WILD * WILD y = &x;
```



# Wild pointers can refer to any values.

$$Rep(\tau\ *\ \texttt{WILD}) = \texttt{struct}\{Rep(\tau)\ *\ p, b;\}$$

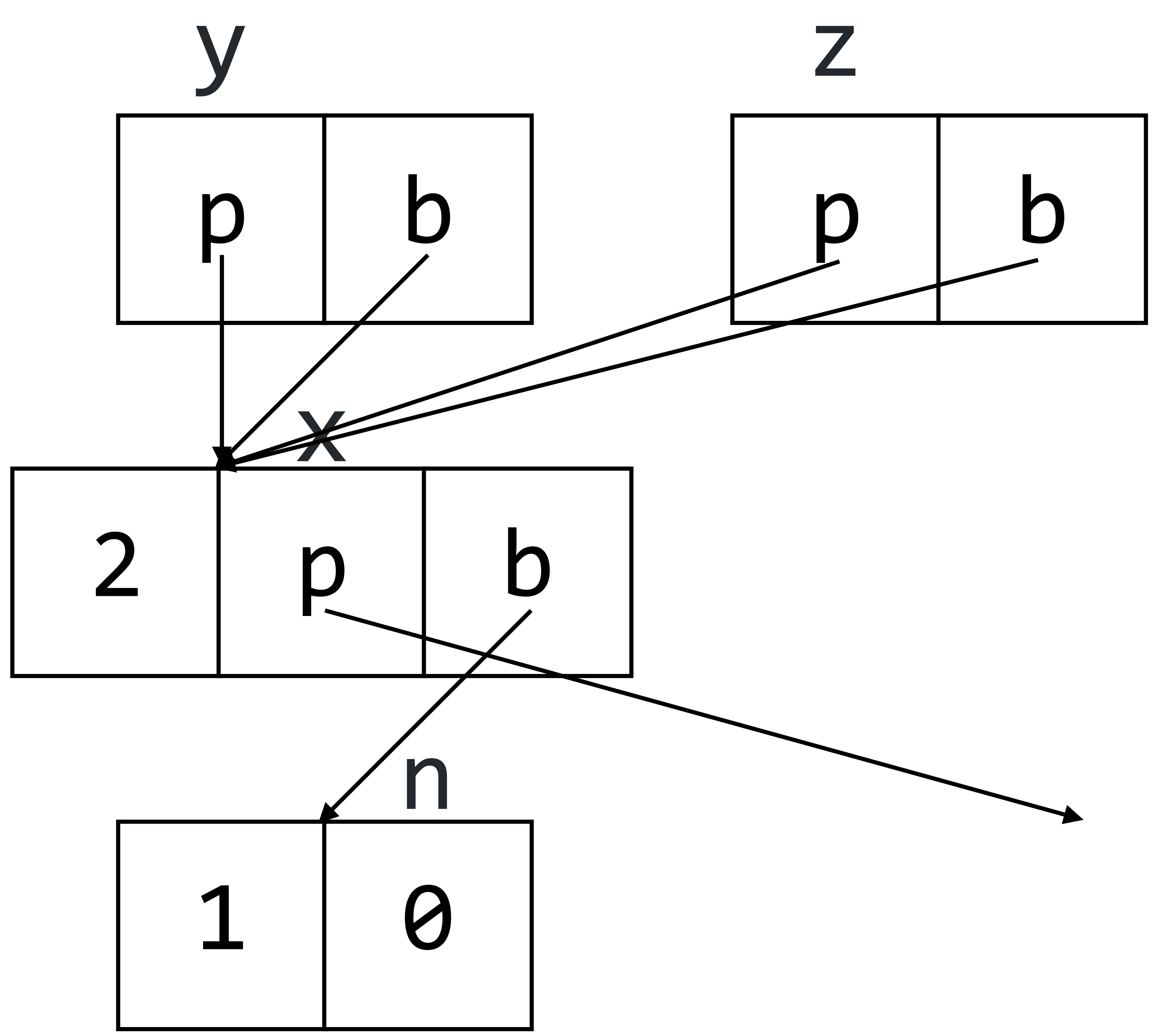
```
int n = 0;
int * WILD x = &n;
int * WILD * WILD y = &x;
int * WILD z = (int * WILD) y;
```



# Wild pointers can refer to any values.

$$Rep(\tau\ *\ \texttt{WILD}) = \texttt{struct}\{Rep(\tau)\ *\ p, b;\}$$

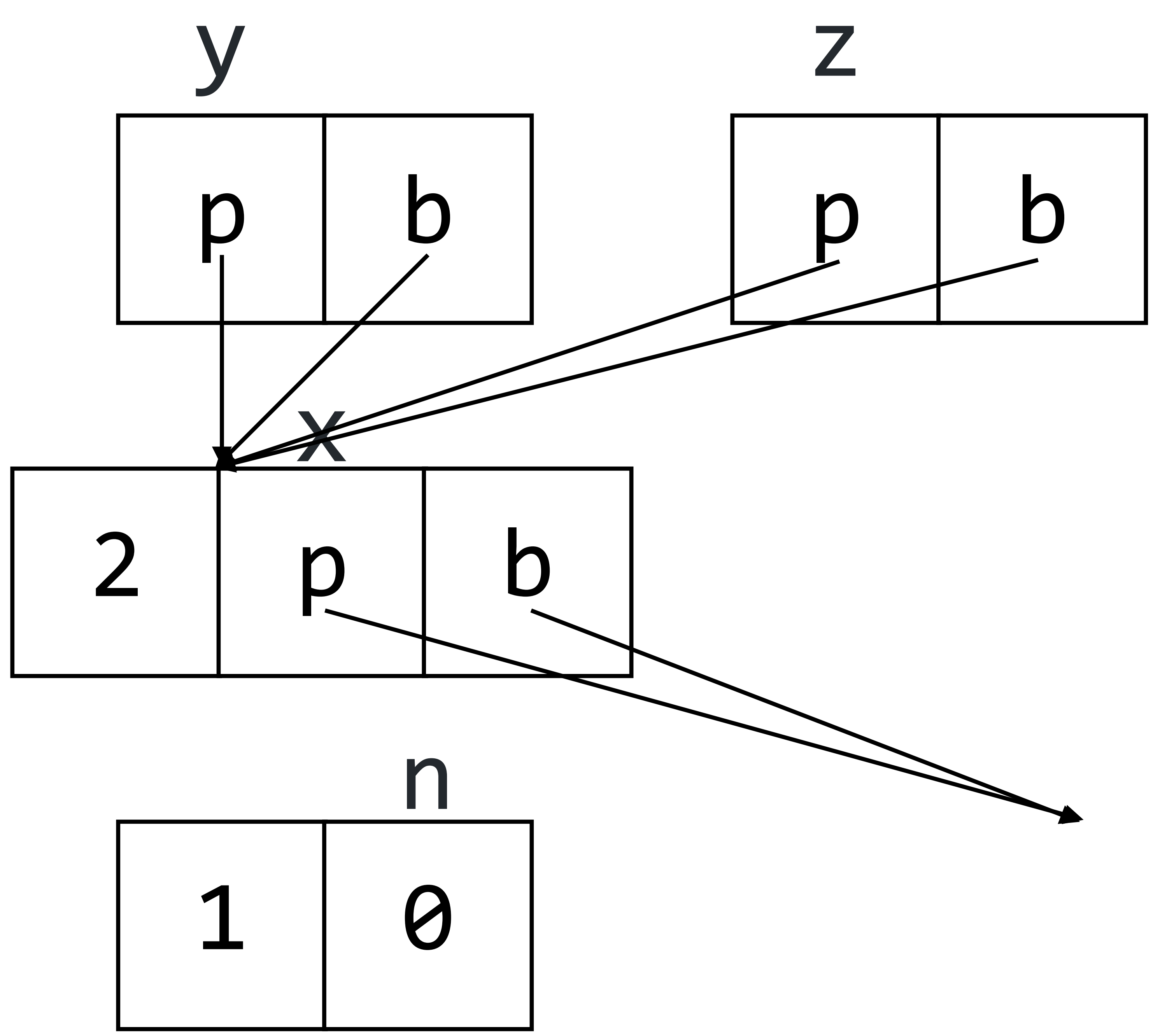
```c
int n = 0;
int * WILD x = &n;
int * WILD * WILD y = &x;
int * WILD z = (int * WILD) y;
*z = N;
```



# Wild pointers can refer to any values.

$$Rep(\tau * \texttt{WILD}) = \texttt{struct}\{Rep(\tau) * p, b;\}$$

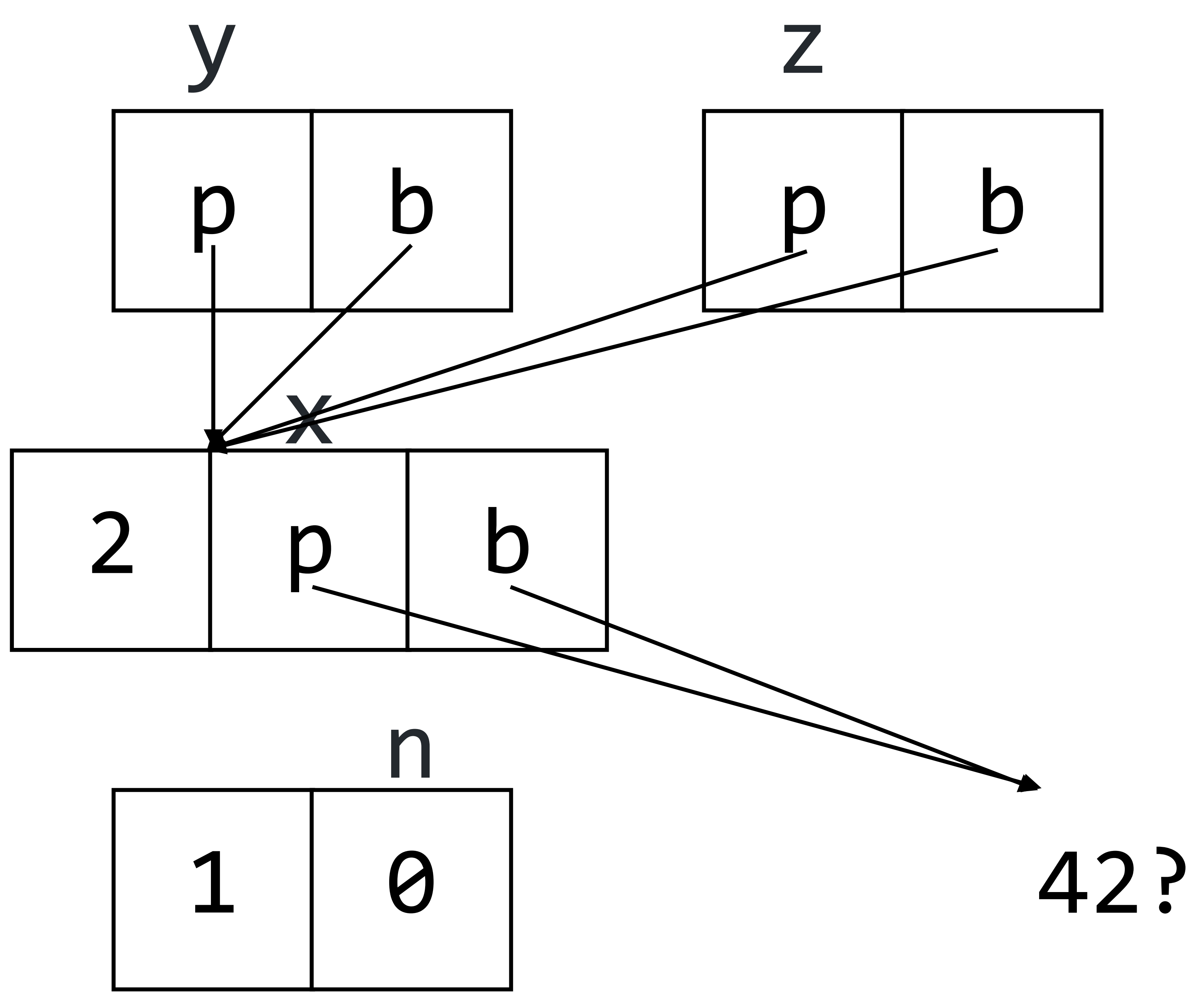
```
int n = 0;
int * WILD x = &n;
int * WILD * WILD y = &x;
int * WILD z = (int * WILD) y;
*z = N;
*(z + 1) = N;
```



# Wild pointers can refer to any values.

$$Rep(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \ * \ p, b;\}$$

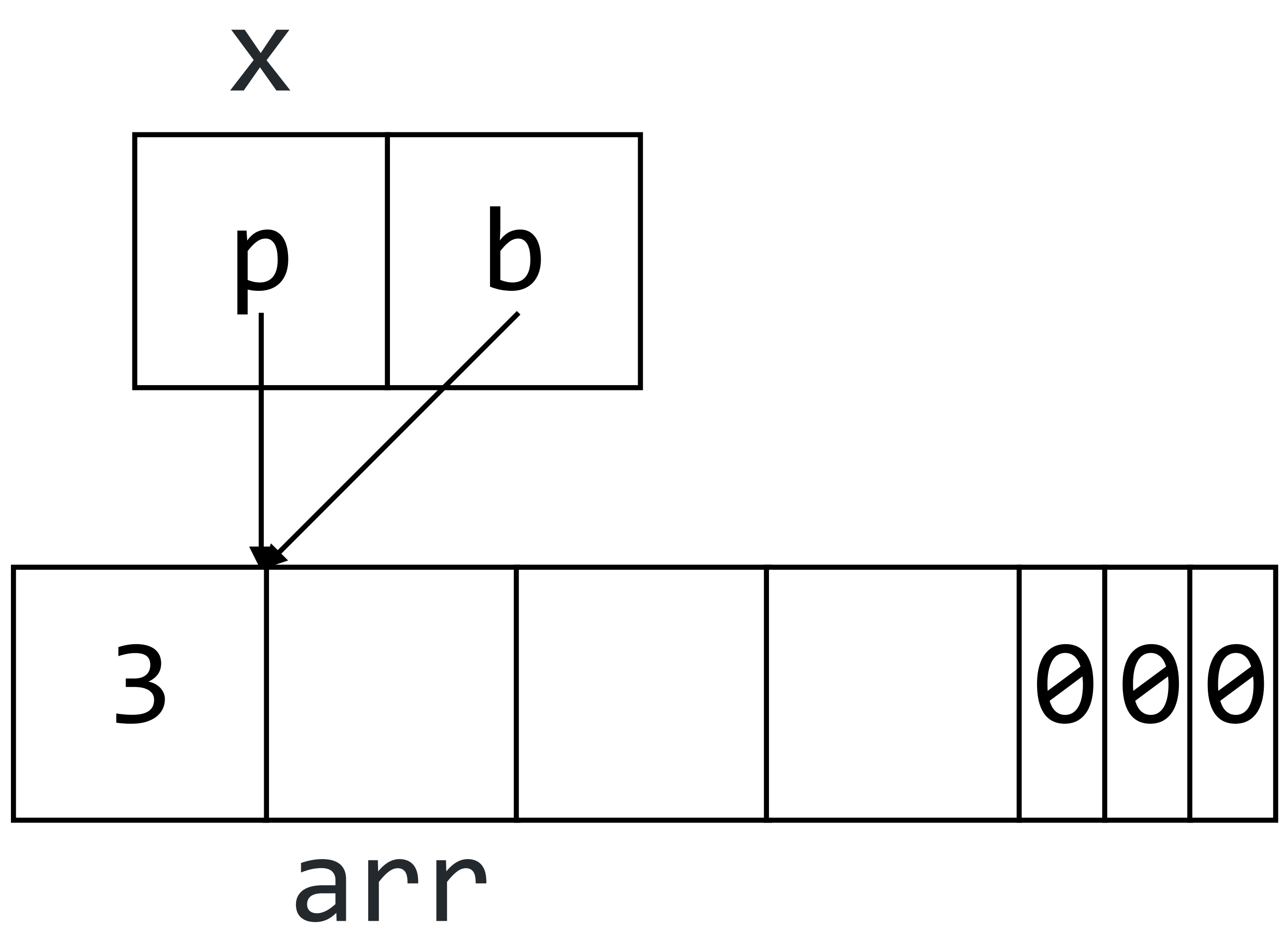
```
int n = 0;
int * WILD x = &n;
int * WILD * WILD y = &x;
int * WILD z = (int * WILD) y;
*z = N;
*(z + 1) = N;
**y = 42;
```



# Wild pointers need tag bits.

$$\mathit{Rep}(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{\mathit{Rep}(\tau) \ * \ p, b;\}$$

```
int arr[3];
int * WILD x = arr;
```

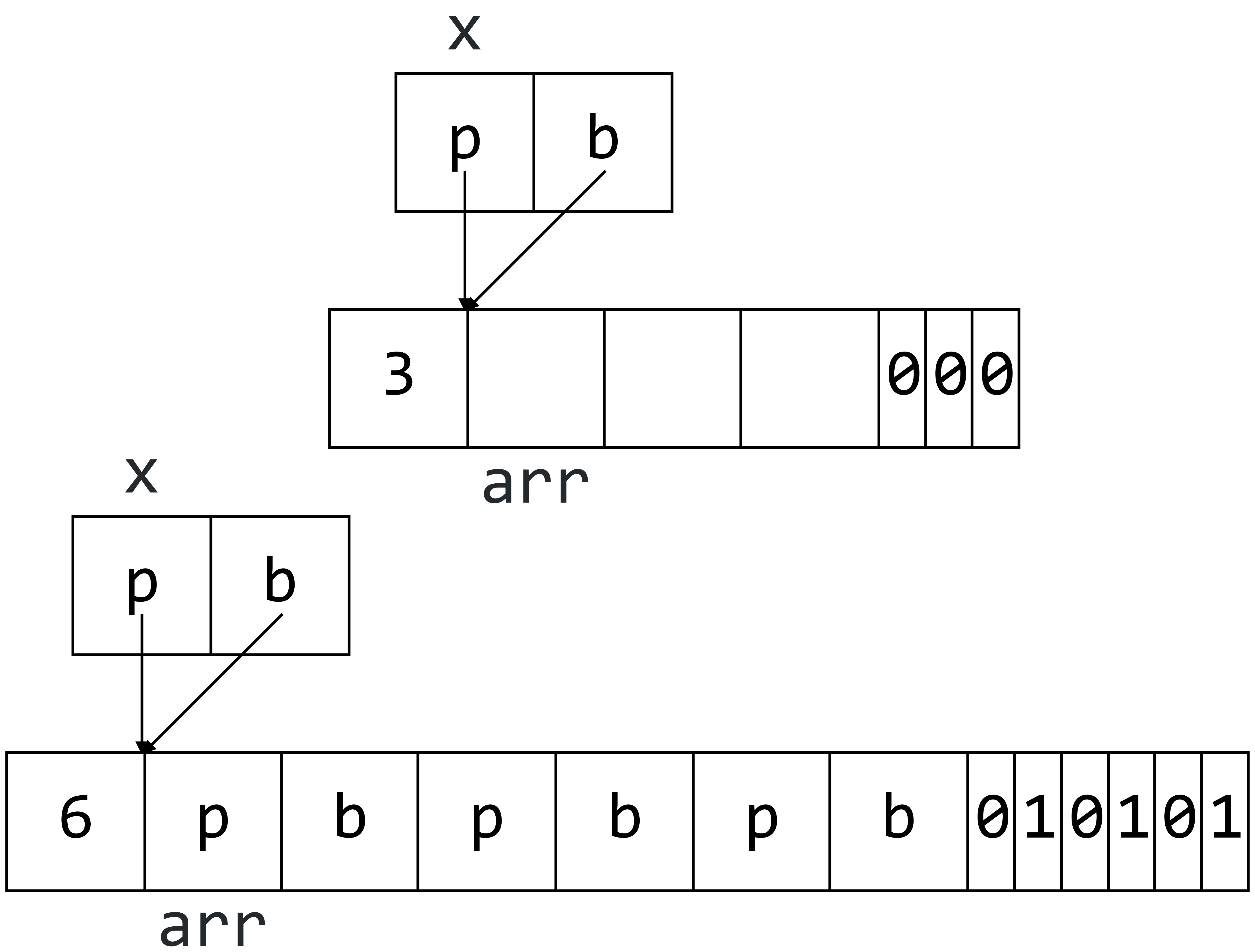


# Wild pointers need tag bits.

$$Rep(\tau\ *\ \mathtt{WILD}) = \mathtt{struct}\{Rep(\tau)\ *\ p, b;\}$$

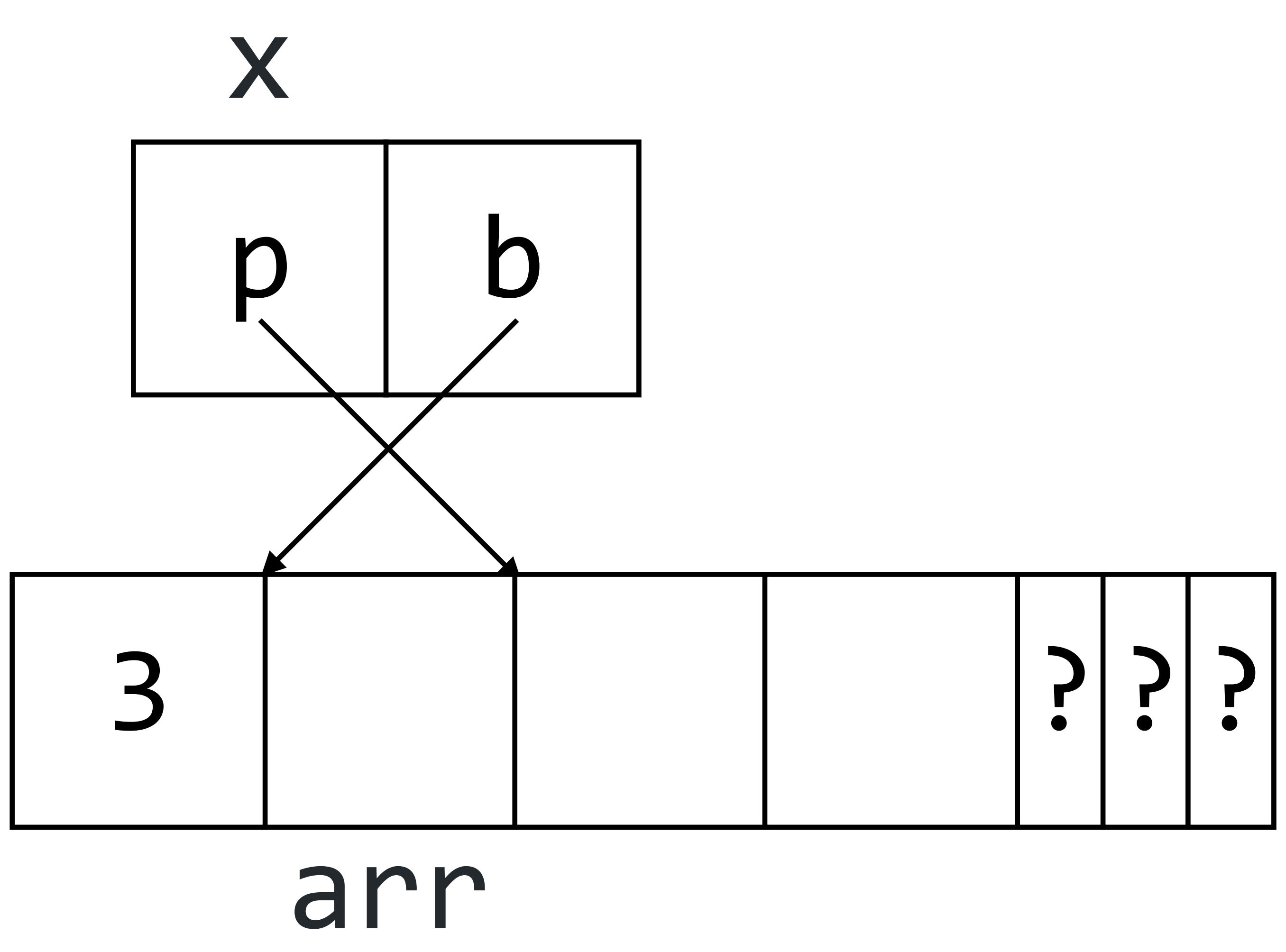
```
int arr[3];
int * WILD x = arr;
```

```
int (* WILD arr)[3];
int * WILD * WILD x = arr;
```



# Wild pointers need tag bits.

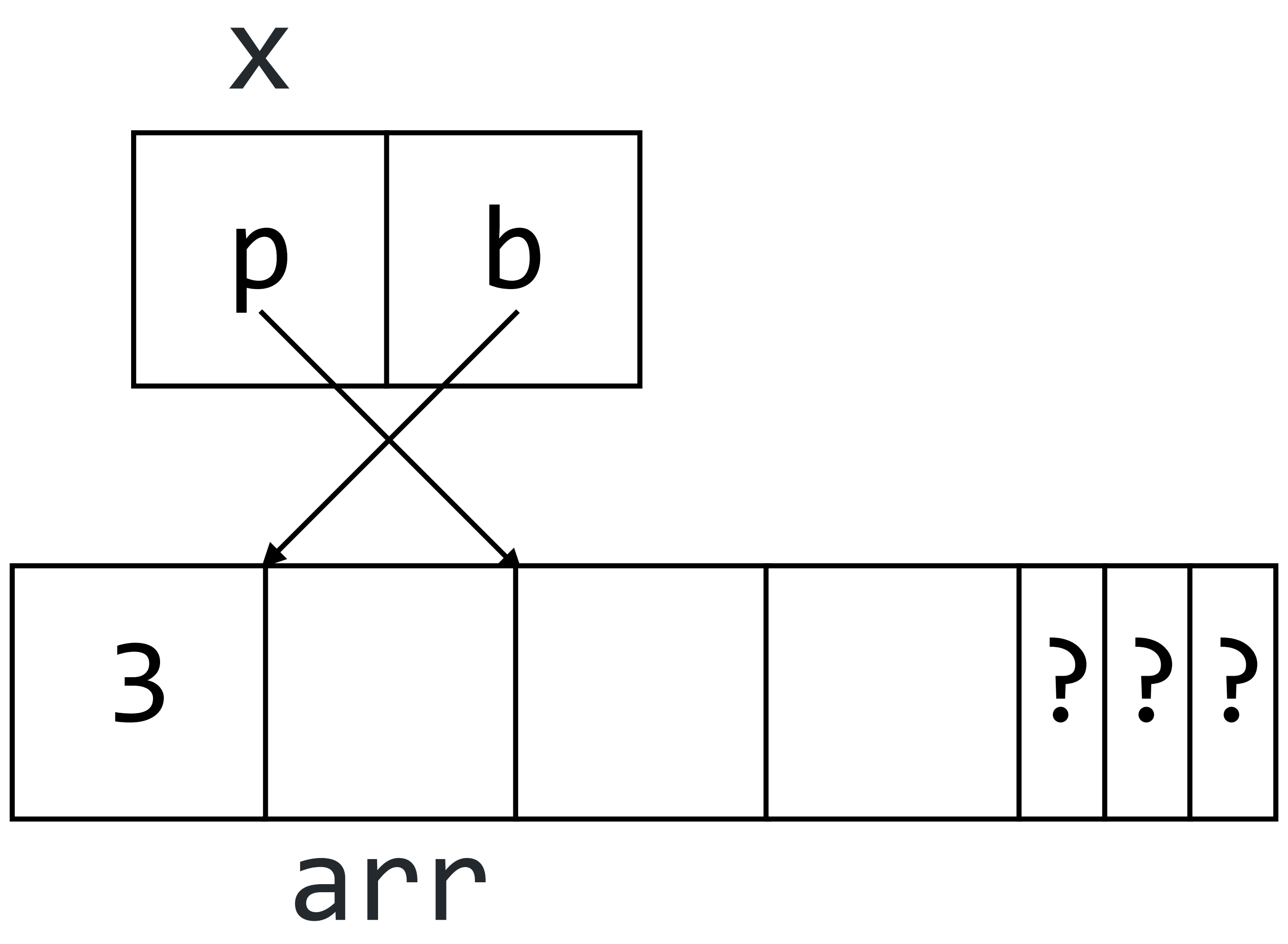
$$Rep(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \ * \ p, b;\}$$




$$\frac{x : \texttt{int} \ * \ \texttt{WILD} \qquad y : \texttt{int}}{*x = y \rightsquigarrow \begin{array}{l} \texttt{assert}(x.b \neq \texttt{null}); \\ \texttt{assert}(x.b \leq x.p \leq \texttt{len}(x.b) - \texttt{sizeof}(\texttt{int})); \\ \\ *x.p = y \end{array}}$$

# Wild pointers need tag bits.

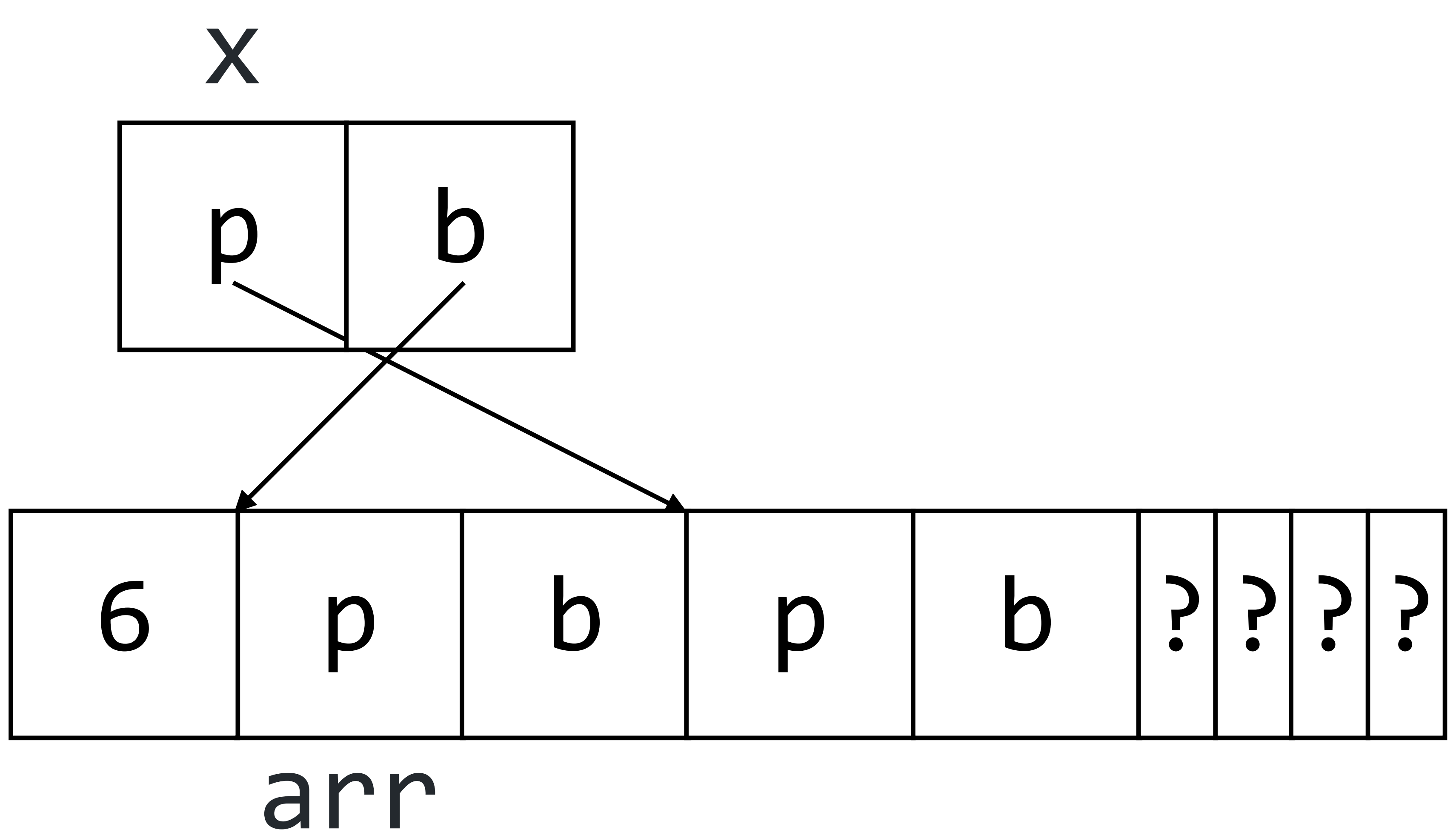
$$Rep(\tau \; * \; \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \; * \; p, b;\}$$




$$\frac{x : \texttt{int} \; * \; \texttt{WILD} \qquad y : \texttt{int}}{*x = y \rightsquigarrow \begin{array}{l} \texttt{assert}(x.b \neq \texttt{null}); \\ \texttt{assert}(x.b \leq x.p \leq \texttt{len}(x.b) - \texttt{sizeof}(\texttt{int})); \\ \texttt{tag}(x.b, x.p) = 0; \\ *x.p = y \end{array}}$$

# Wild pointers need tag bits.

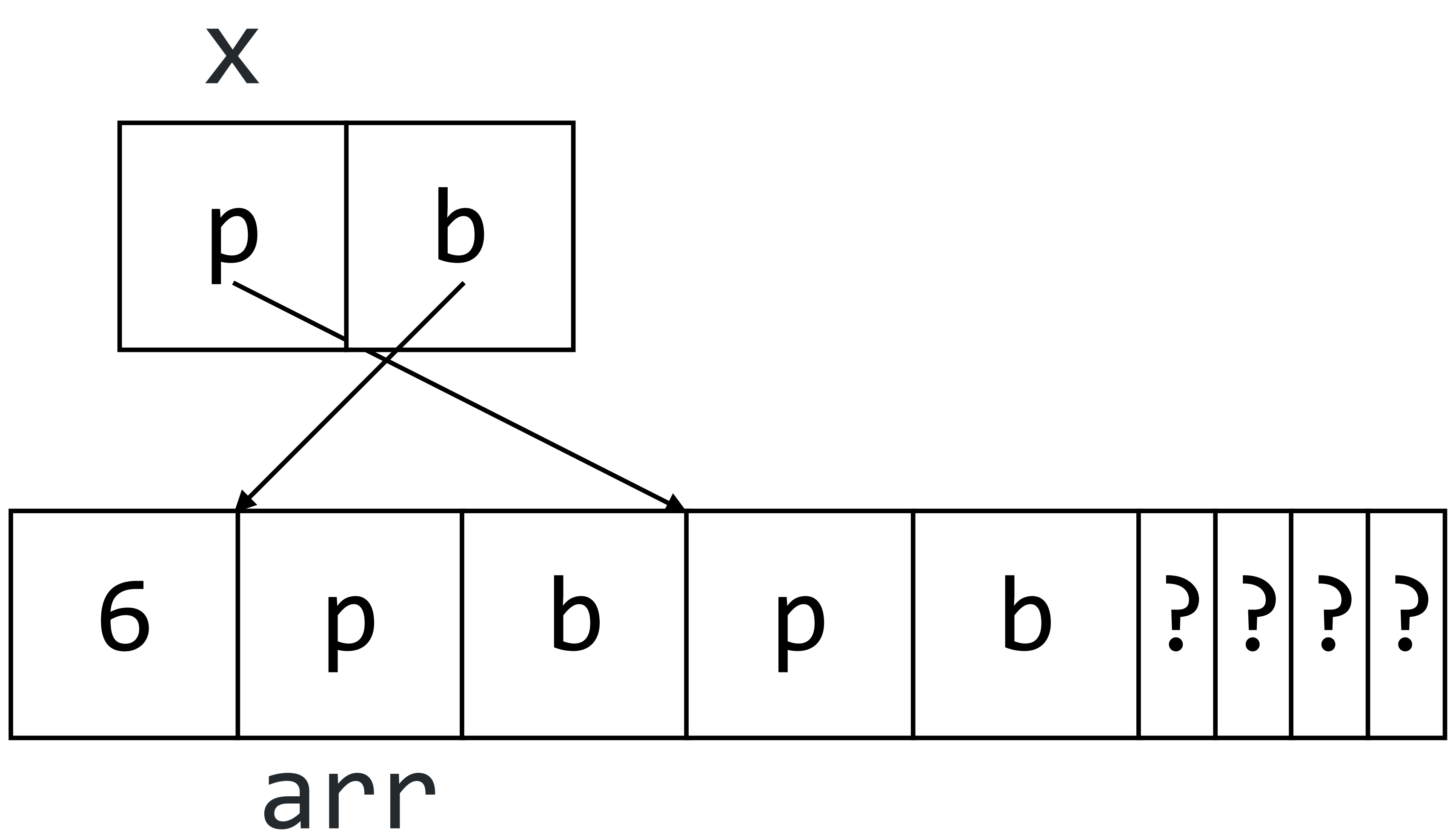
$$Rep(\tau\ *\ \mathtt{WILD}) = \mathtt{struct}\{Rep(\tau)\ *\ p, b;\}$$




$$\frac{x : \tau\ *\ \mathtt{WILD}\ *\ \mathtt{WILD} \qquad y : \tau\ *\ \mathtt{WILD}}{*x = y \rightsquigarrow \begin{array}{l} \mathtt{assert}(x.b \neq \mathtt{null}); \\ \mathtt{assert}(x.b \leq x.p \leq \mathtt{len}(x.b) - \mathtt{sizeof}(\tau\ *\ \mathtt{WILD})); \\ \mathtt{assert}(\mathtt{tag}(x.b, x.p + 1) = 1); \\ \\ \\ *x.p = y \end{array}}$$

# Wild pointers need tag bits.

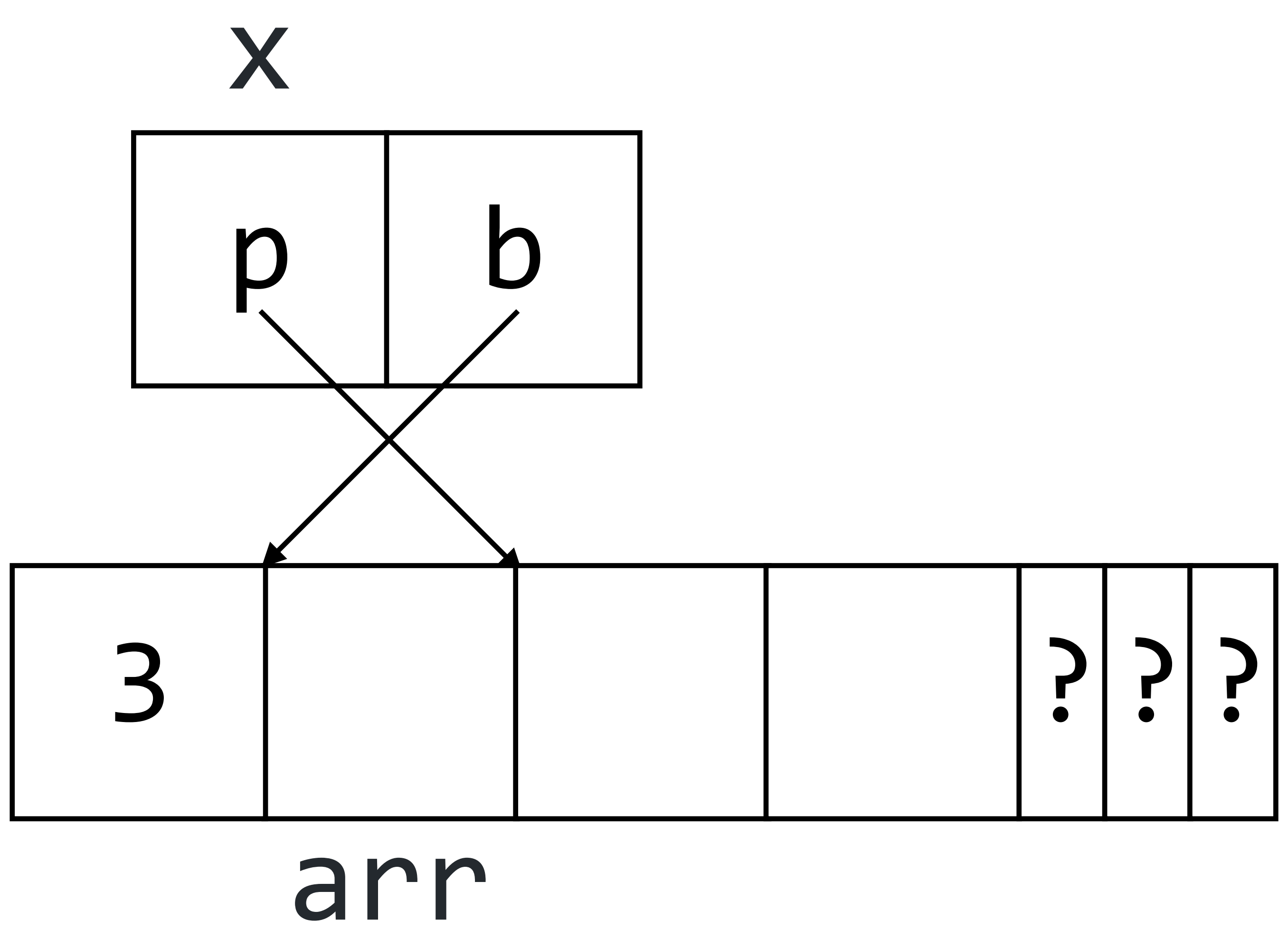
$$Rep(\tau\ *\ \texttt{WILD}) = \texttt{struct}\{Rep(\tau)\ *\ p, b;\}$$




$$\frac{x : \tau\ *\ \texttt{WILD}\ *\ \texttt{WILD} \qquad y : \tau\ *\ \texttt{WILD}}{*x = y \rightsquigarrow \begin{array}{l} \texttt{assert}(x.b \neq \texttt{null}); \\ \texttt{assert}(x.b \leq x.p \leq \texttt{len}(x.b) - \texttt{sizeof}(\tau\ *\ \texttt{WILD})); \\ \texttt{assert}(\texttt{tag}(x.b, x.p + 1) = 1); \\ \texttt{tag}(x.b, x.p) = 0; \\ \texttt{tag}(x.b, x.p + 1) = 1; \\ *x.p = y \end{array}}$$

# Wild pointers need tag bits.

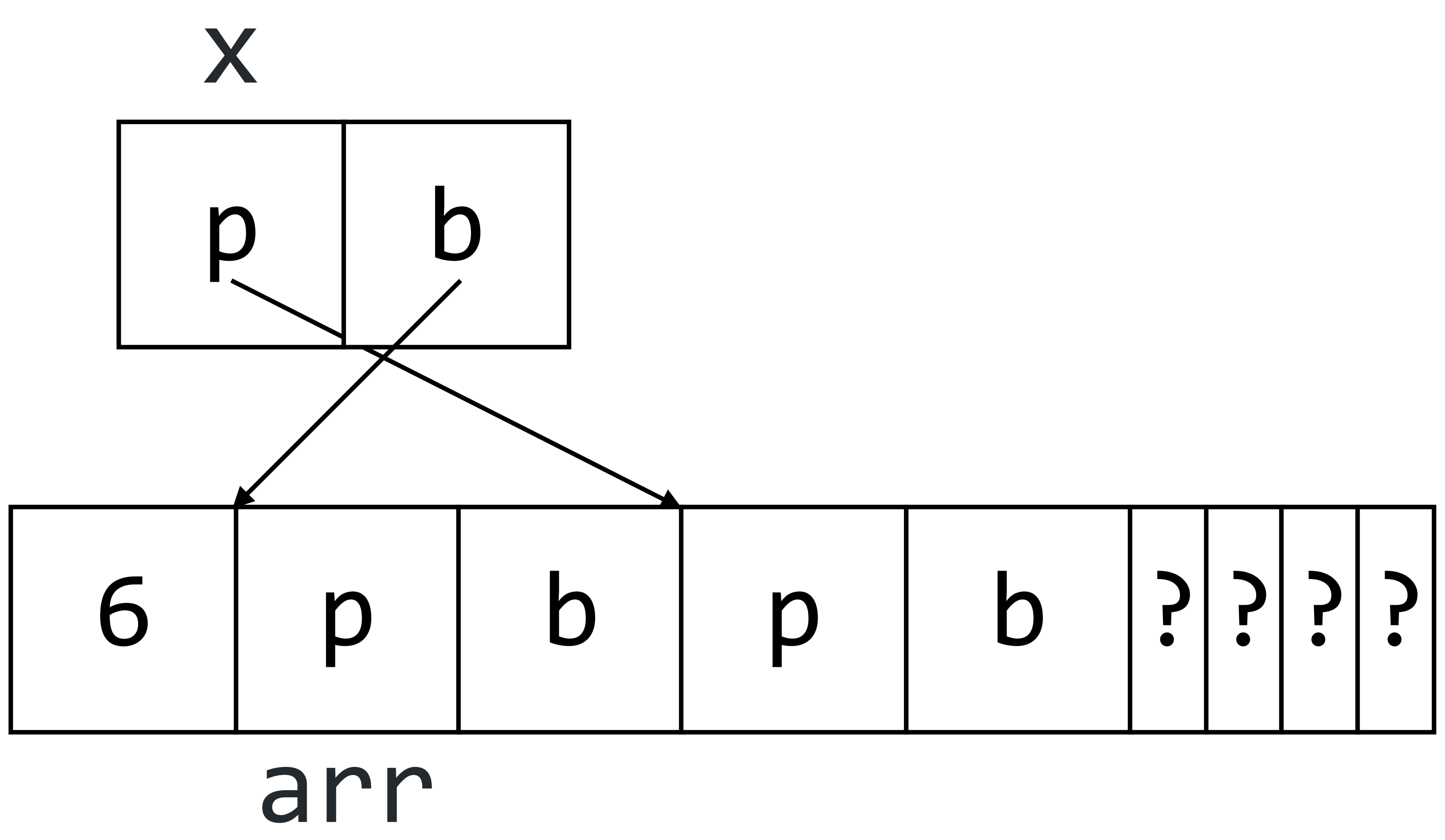
$$Rep(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \ * \ p, b;\}$$




$$\frac{x : \texttt{int} \ * \ \texttt{WILD}}{*x \rightsquigarrow \begin{array}{l} \texttt{assert}(x.b \neq \texttt{null}); \\ \texttt{assert}(x.b \leq x.p \leq \texttt{len}(x.b) - \texttt{sizeof}(\texttt{int})); \\ *x.p \end{array}}$$

# Wild pointers need tag bits.

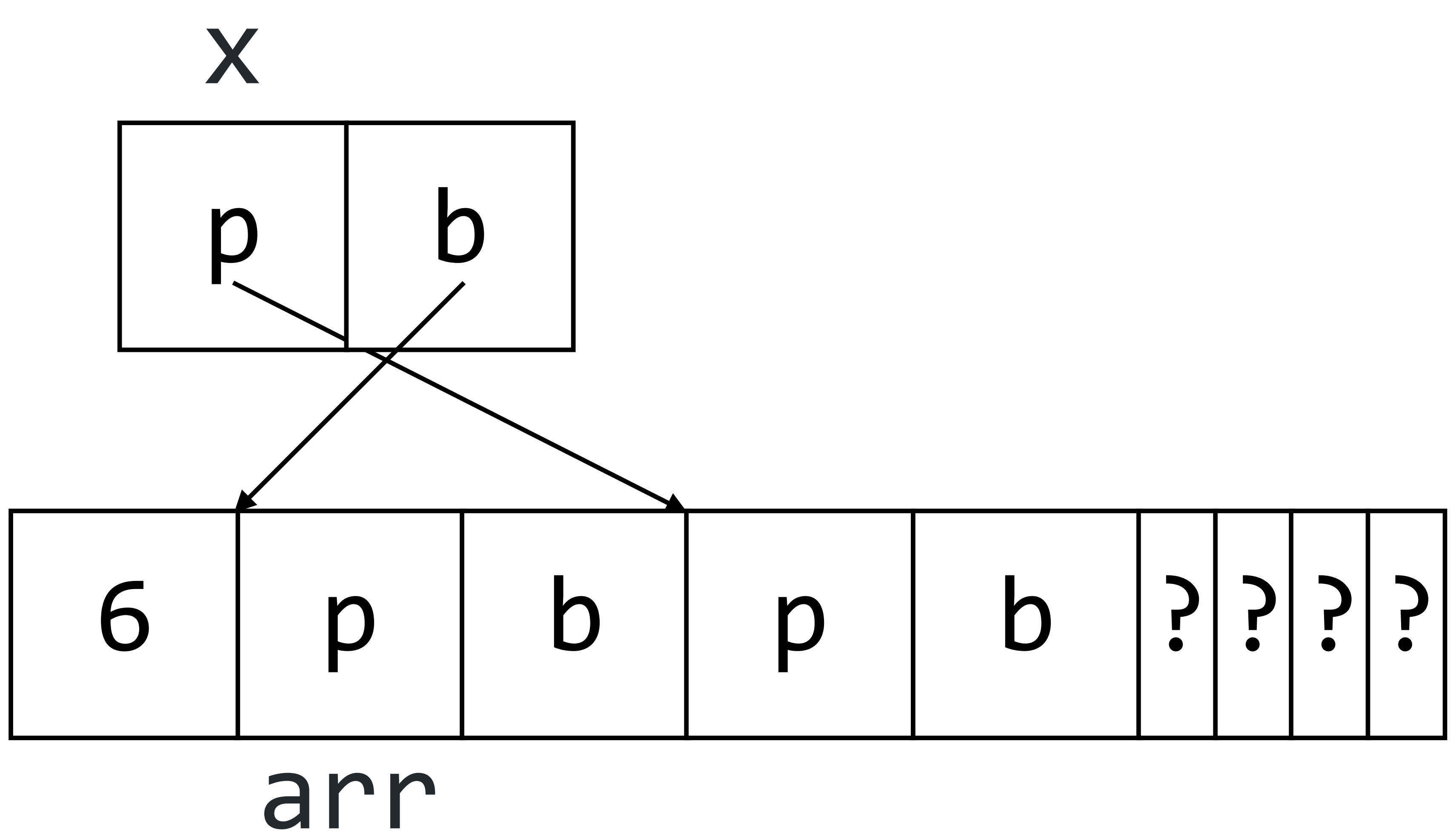
$$Rep(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \ * \ p, b;\}$$




$$\frac{x : \tau \ * \ \texttt{WILD} \ * \ \texttt{WILD}}{*x \rightsquigarrow \begin{array}{l} \texttt{assert}(x.b \neq \texttt{null}); \\ \texttt{assert}(x.b \leq x.p \leq \texttt{len}(x.b) - \texttt{sizeof}(\tau \ * \ \texttt{WILD})); \\ \\ *x.p \end{array}}$$

# Wild pointers need tag bits.

$$Rep(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \ * \ p, b; \}$$

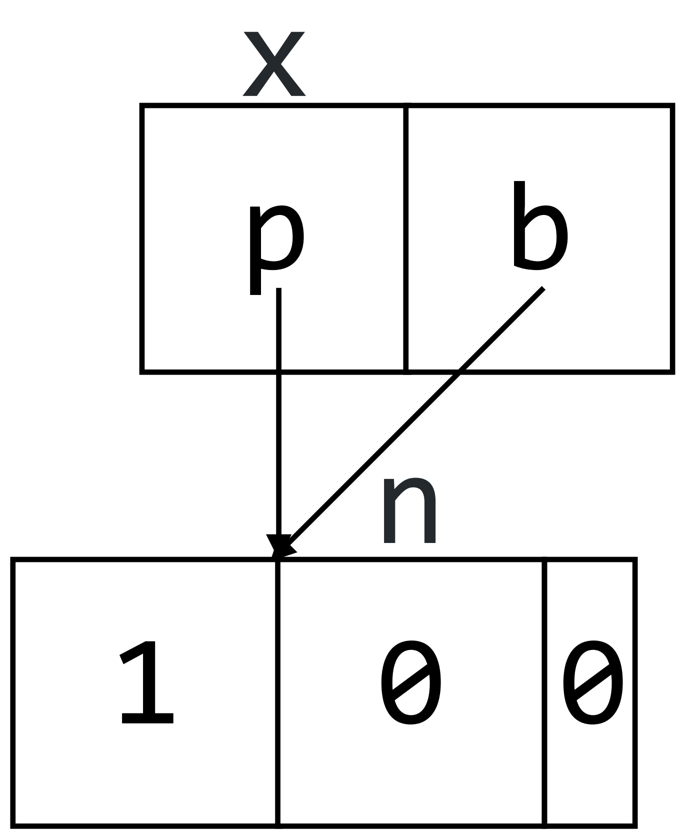



$$\frac{x : \tau \ * \ \texttt{WILD} \ * \ \texttt{WILD}}{*x \leadsto \begin{array}{l} \texttt{assert}(x.b \neq \texttt{null}); \\ \texttt{assert}(x.b \leq x.p \leq \texttt{len}(x.b) - \texttt{sizeof}(\tau \ * \ \texttt{WILD})); \\ \texttt{assert}(\texttt{tag}(x.b, x.p + 1) = 1); \\ *x.p \end{array}}$$

# Wild pointers need tag bits.

$$Rep(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \ * \ p, b;\}$$

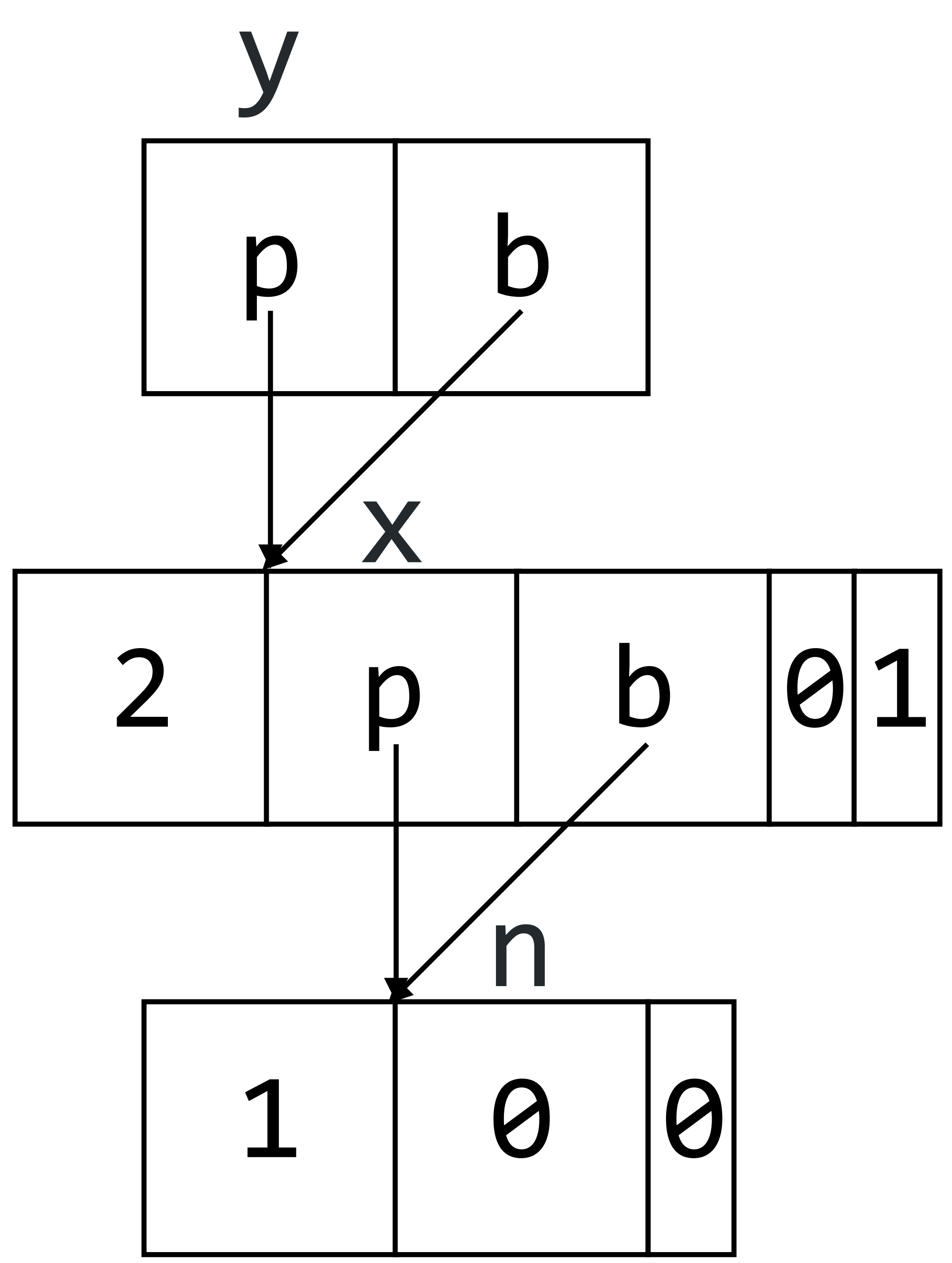
```
int n = 0;
int * WILD x = &n;
```



# Wild pointers need tag bits.

$$Rep(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \ * \ p, b;\}$$

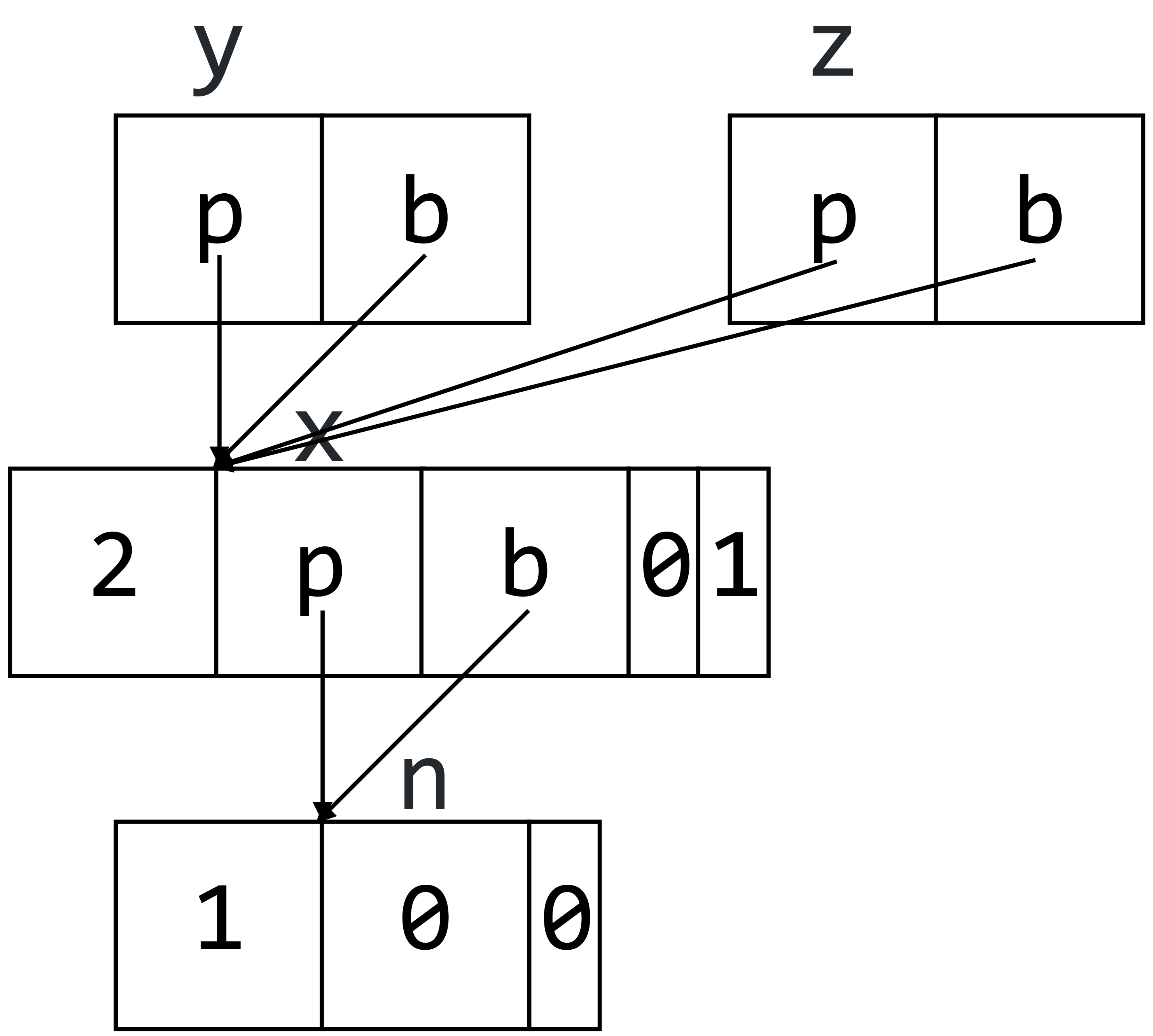
```
int n = 0;
int * WILD x = &n;
int * WILD * WILD y = &x;
```



# Wild pointers need tag bits.

$$Rep(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \ * \ p, b;\}$$

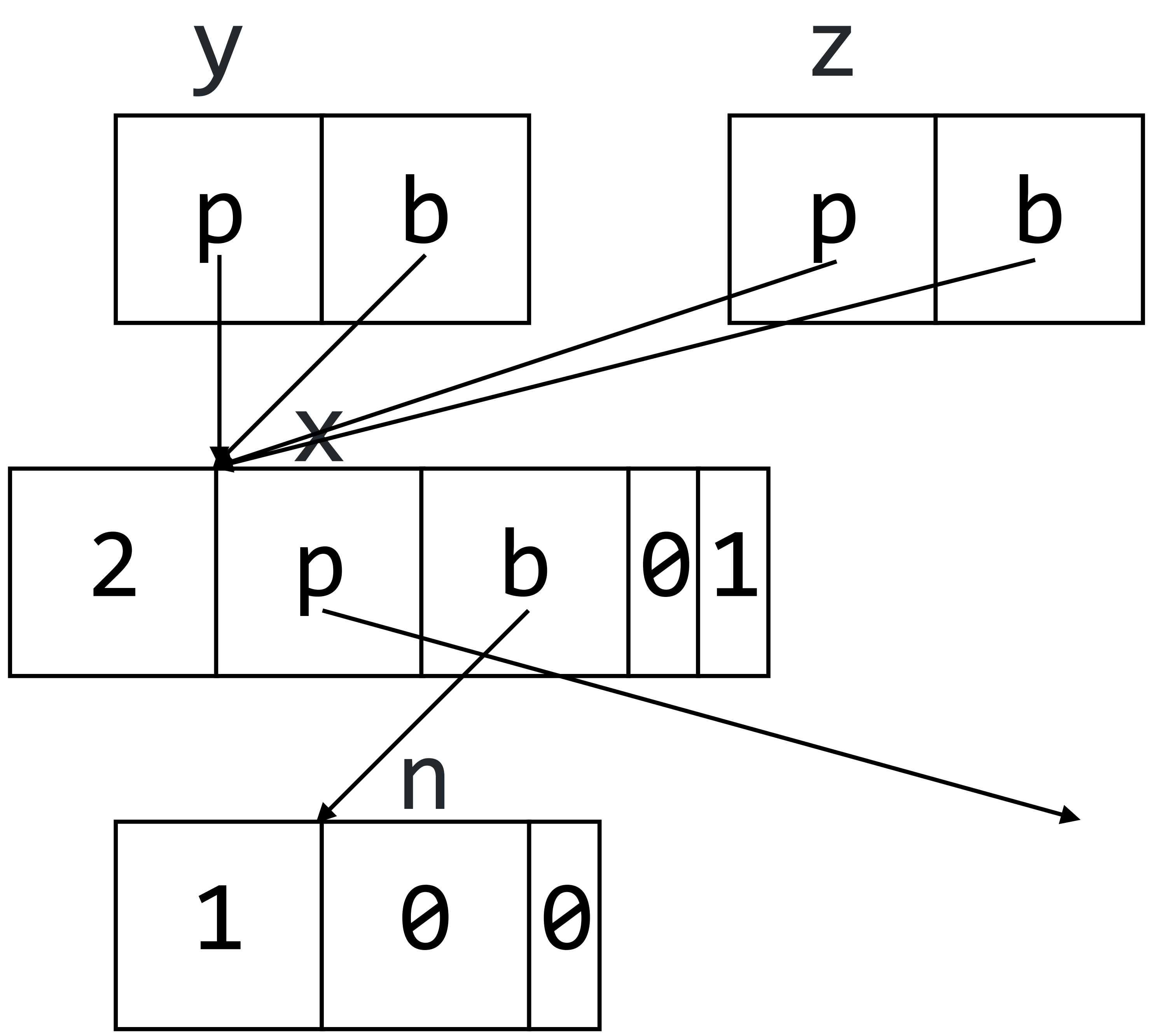
```c
int n = 0;
int * WILD x = &n;
int * WILD * WILD y = &x;
int * WILD z = (int * WILD) y;
```



# Wild pointers need tag bits.

$$Rep(\tau * \texttt{WILD}) = \texttt{struct}\{Rep(\tau) * p, b;\}$$

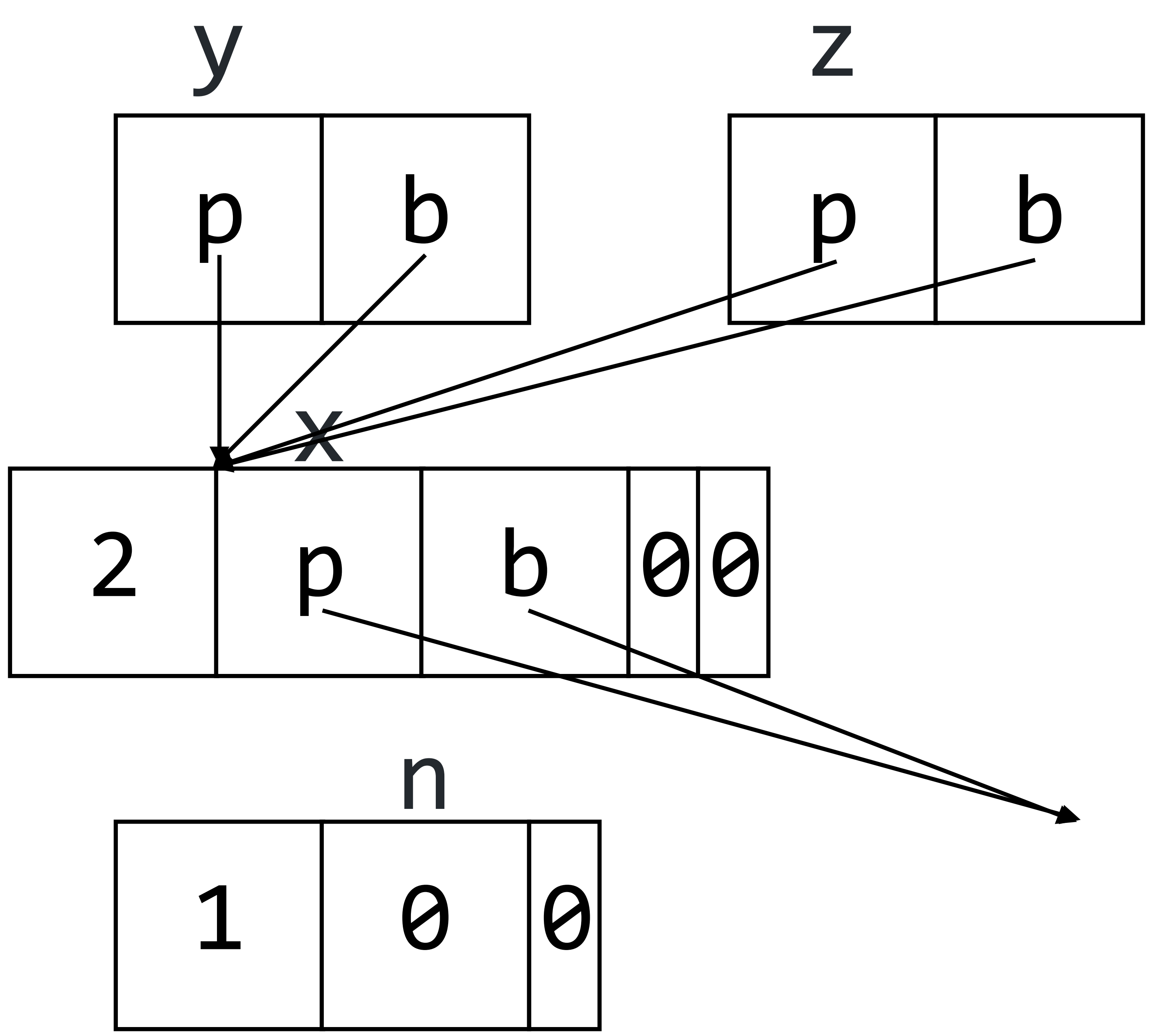
```
int n = 0;
int * WILD x = &n;
int * WILD * WILD y = &x;
int * WILD z = (int * WILD) y;
*z = N;
```



# Wild pointers need tag bits.

$$Rep(\tau \ast \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \ast p, b;\}$$

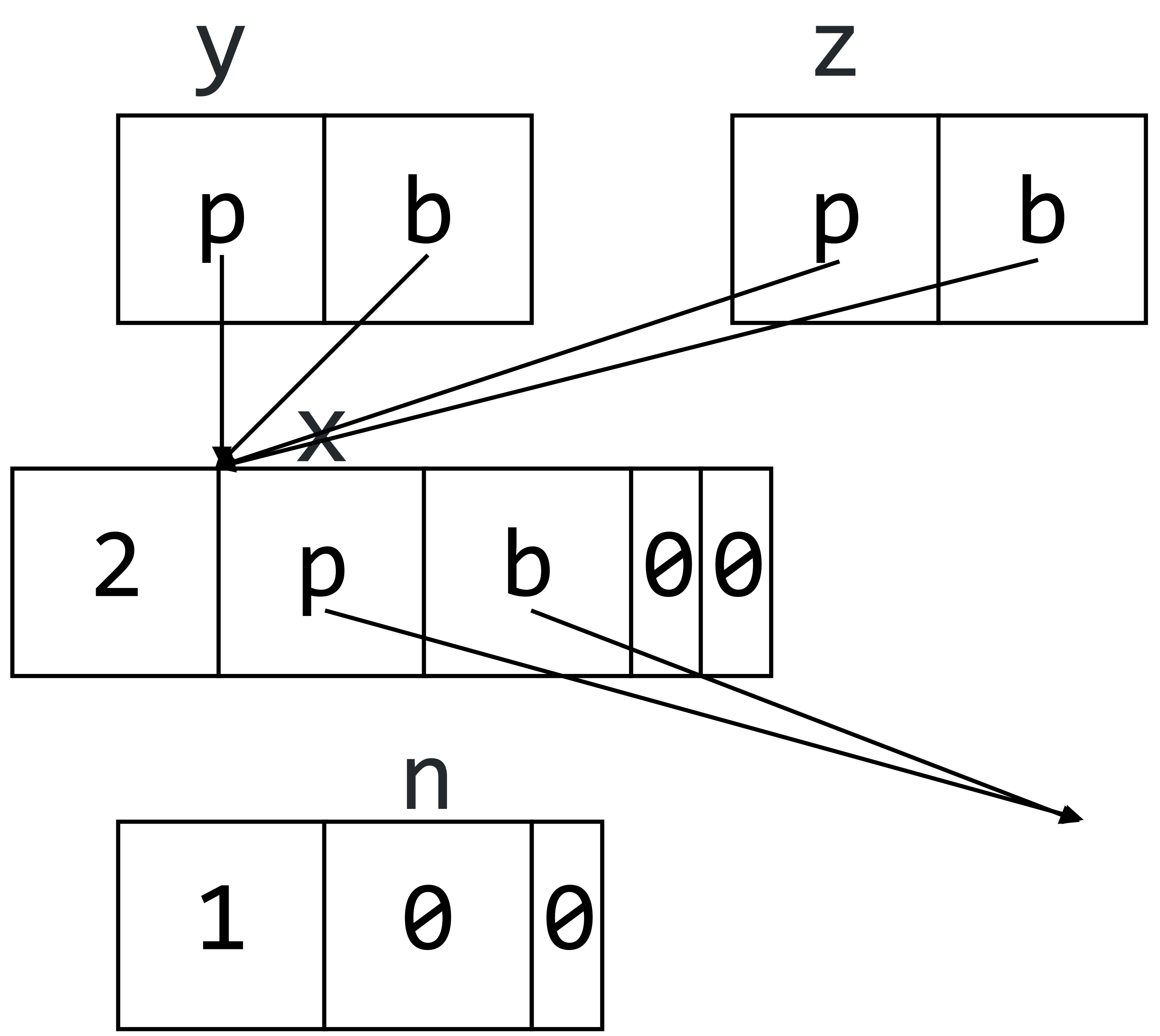
```
int n = 0;
int * WILD x = &n;
int * WILD * WILD y = &x;
int * WILD z = (int * WILD) y;
*z = N;
*(z + 1) = N;
```



# Wild pointers need tag bits.

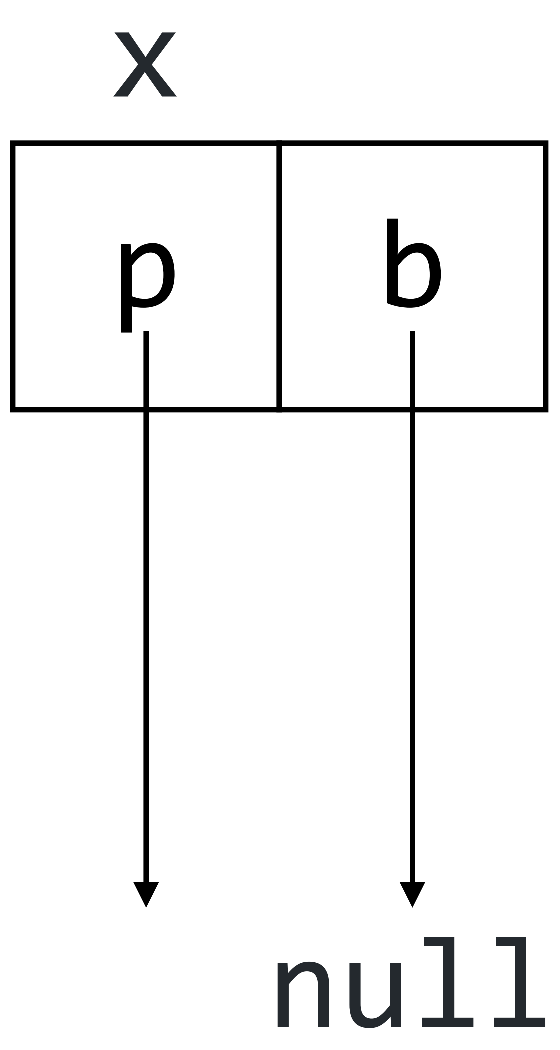
$$Rep(\tau * \texttt{WILD}) = \texttt{struct}\{Rep(\tau) * p, b;\}$$

```
int n = 0;
int * WILD x = &n;
int * WILD * WILD y = &x;
int * WILD z = (int * WILD) y;
*z = N;
*(z + 1) = N;
**y = 42;
```



# Integers can be casted to wild pointers.

$$\mathit{Rep}(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{\mathit{Rep}(\tau) \ * \ p, b;\}$$

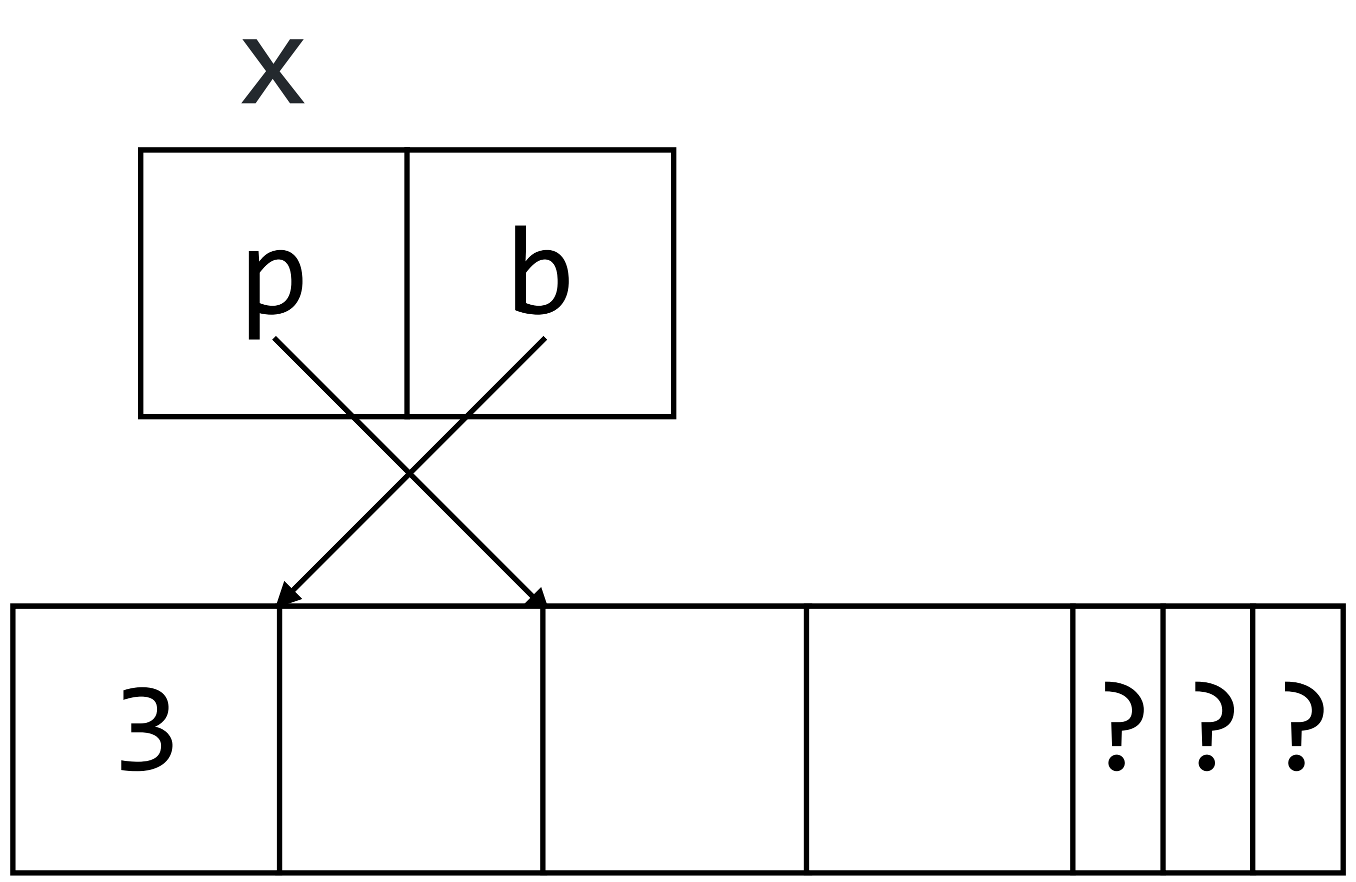



$$\frac{x : \texttt{int}}{(\tau \ * \ \texttt{WILD})x \leadsto \{p = x, b = \texttt{null}\}}$$

$$\frac{x : \tau \ * \ \texttt{WILD}}{(\texttt{int})x \leadsto x.p}$$

# Pointer arithmetic is allowed for wild pointers.

$$Rep(\tau \ * \ \texttt{WILD}) = \texttt{struct}\{Rep(\tau) \ * \ p, b;\}$$




$$\frac{x : \tau \ * \ \texttt{WILD} \qquad y : \texttt{int}}{x + y \rightsquigarrow \{p = x.p + y \times \texttt{sizeof}(\tau), b = x.b\}}$$

# Wild pointers can't be casted to/from the other kinds.

```c
int * WILD * SAFE x;
int * WILD * WILD y =
    (int * WILD * WILD) x;
int * WILD z = (int * WILD) y;
*z = N;
*(z + 1) = N;
**x = 42;
```

$$\frac{x : \tau \ * \ \mathsf{SAFE}}{(\tau' \ * \ \mathsf{WILD})x : \text{not ok}}$$

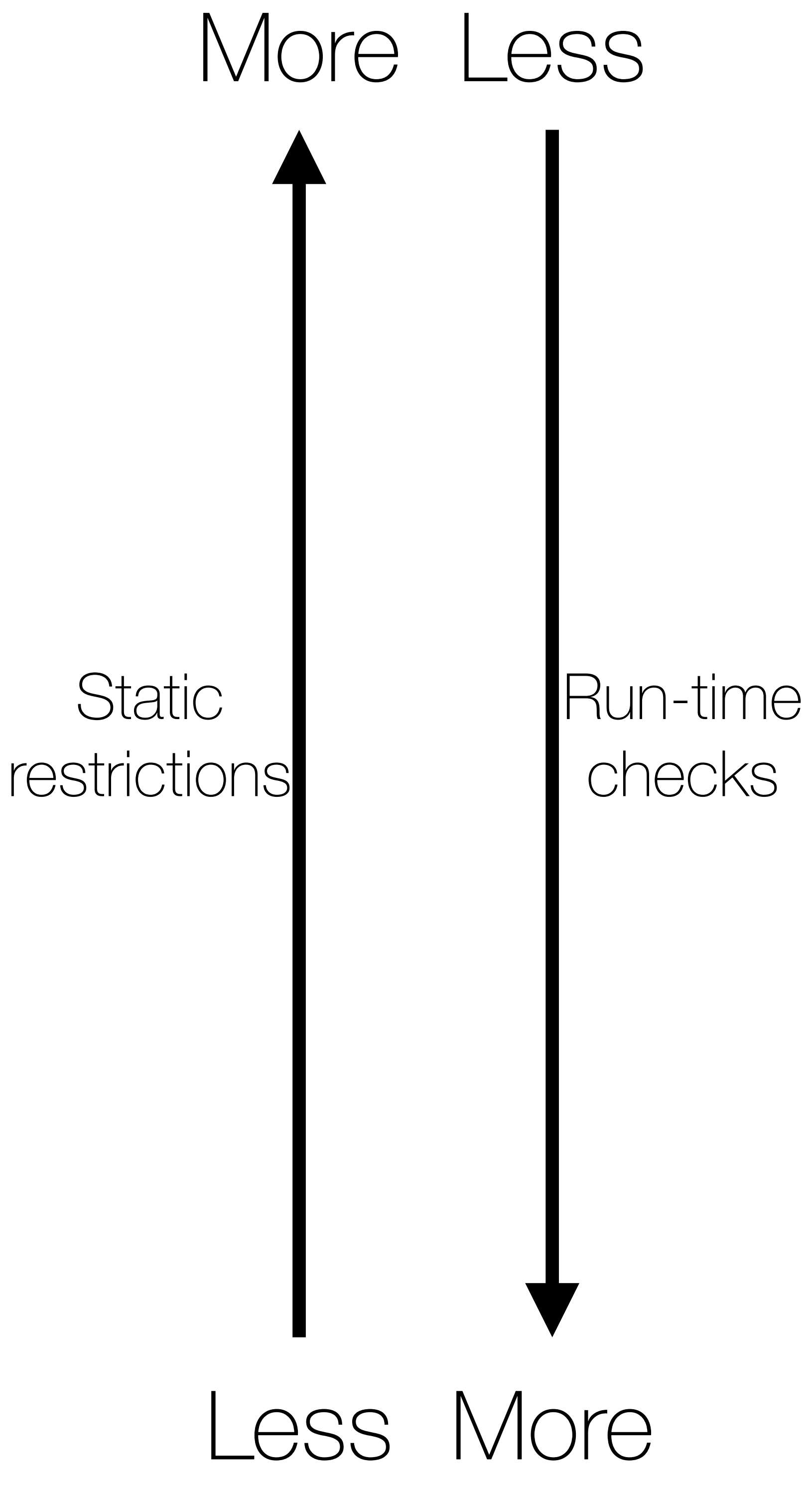
$$\frac{x : \tau \ * \ \mathsf{SEQ}}{(\tau' \ * \ \mathsf{WILD})x : \text{not ok}}$$

$$\frac{x : \tau \ * \ \mathsf{WILD}}{(\tau' \ * \ \mathsf{SAFE})x : \text{not ok}}$$

$$\frac{x : \tau \ * \ \mathsf{WILD}}{(\tau' \ * \ \mathsf{SEQ})x : \text{not ok}}$$

# CCured classifies pointers into 3 kinds.

| | Read/write | Pointer arithmetic | Arbitrary casts | Null checks | Bounds checks | Tags checks |
|---|---|---|---|---|---|---|
| `SAFE` | ✓ | | | ⚙ | | |
| `SEQ` | ✓ | ✓ | | ⚙ | ⚙ | |
| `WILD` | ✓ | ✓ | ✓ | ⚙ | ⚙ | ⚙ |

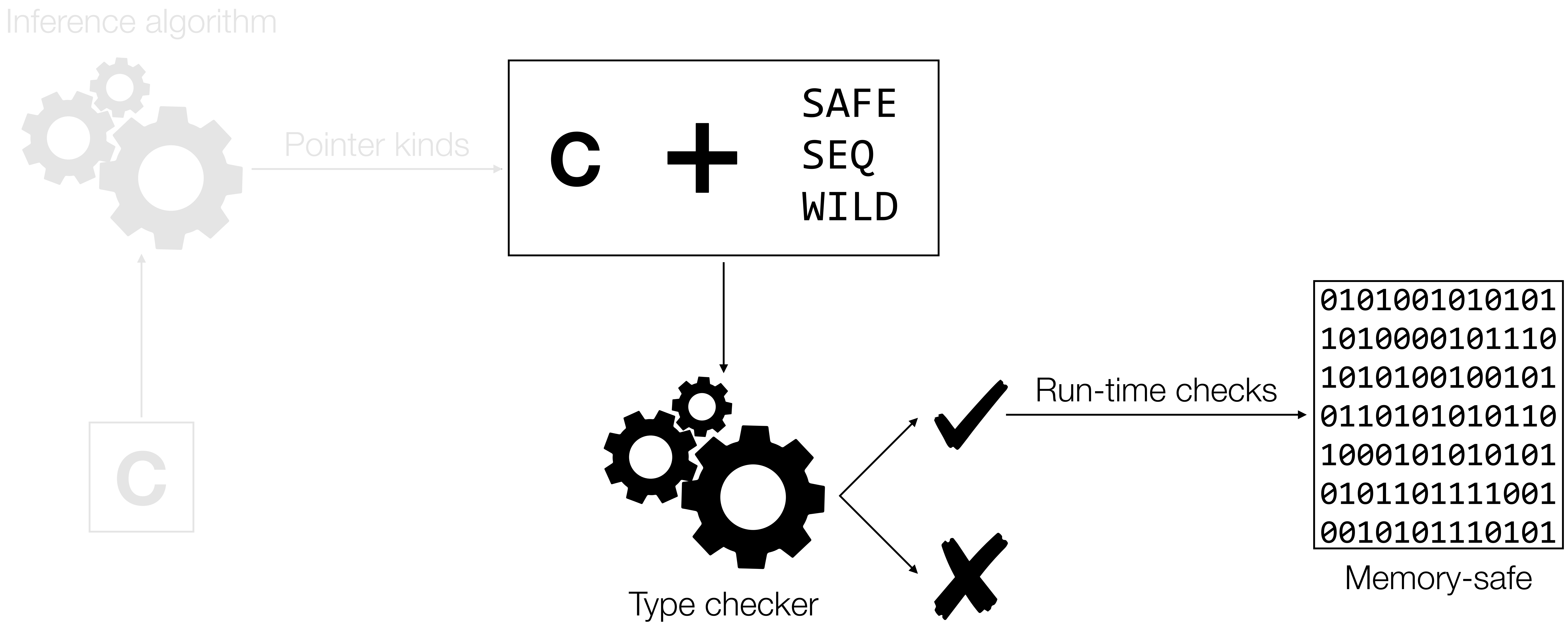


***Theorem.***

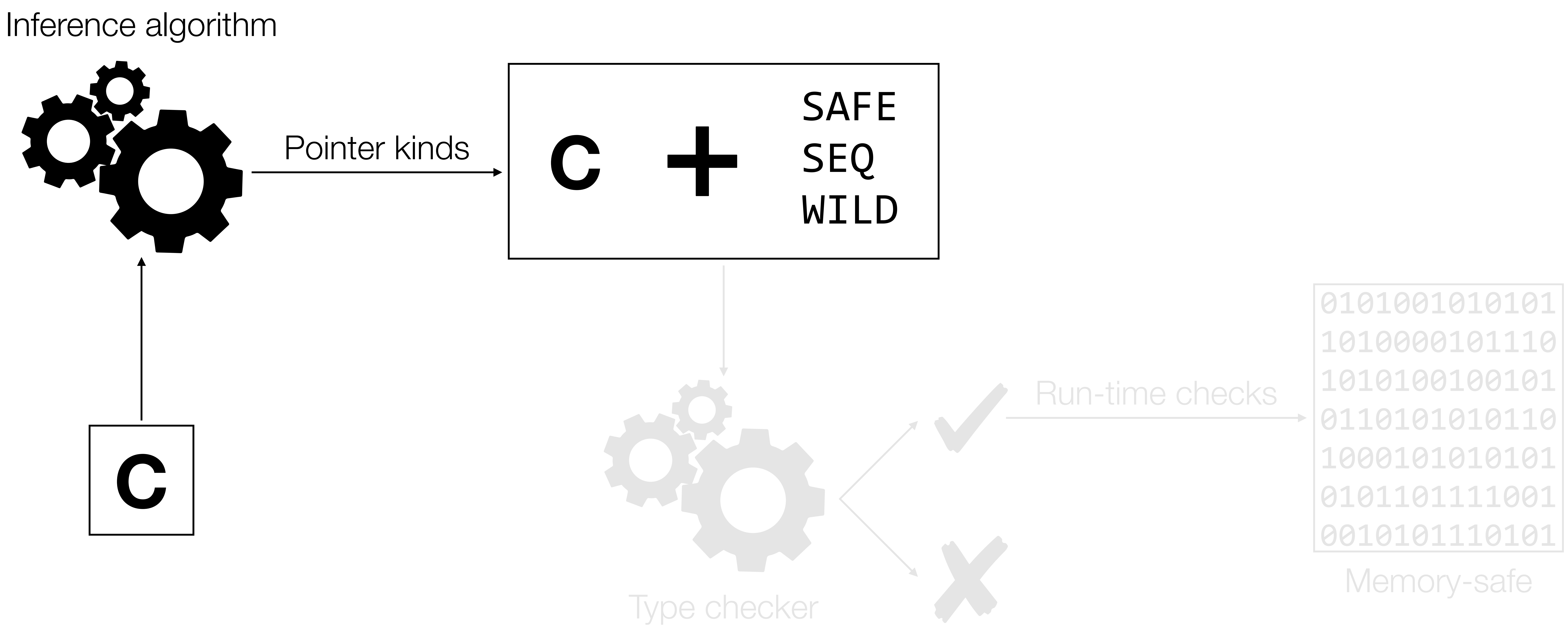
Every well-typed CCured program either

- terminates due to an assertion failure or
- runs without type/memory errors.

# CCured transforms C programs to achieve memory safety guarantees.

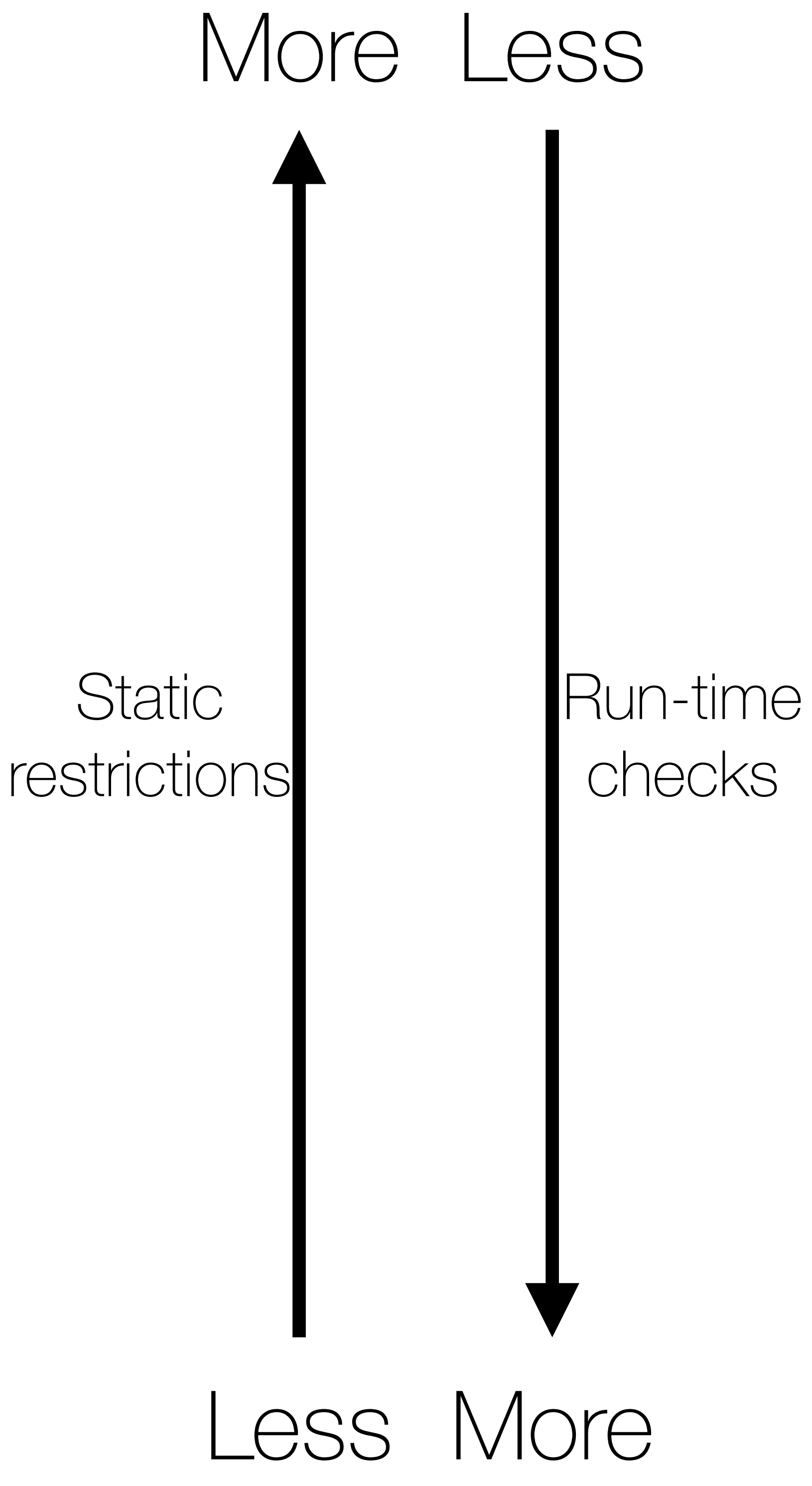


# CCured transforms C programs to achieve memory safety guarantees.



# Programs should use as few wild pointers as possible.

| | Read/write | Pointer arithmetic | Arbitrary casts | Null checks | Bounds checks | Tags checks |
|---|---|---|---|---|---|---|
| `SAFE` | ✓ | | | ⚙ | | |
| `SEQ` | ✓ | ✓ | | ⚙ | ⚙ | |
| `WILD` | ✓ | ✓ | ✓ | ⚙ | ⚙ | ⚙ |



# CCured collects constraints from C programs.

$$\frac{x : \tau\ *\ q \qquad y : \mathtt{int}}{x + y \mapsto \{q \neq \mathsf{SAFE}\}} \qquad \frac{x : \mathtt{int} \qquad x \neq 0}{(\tau\ *\ q)x \mapsto \{q \neq \mathsf{SAFE}\}}$$

$$\frac{x : \tau_1\ *\ q_1}{(\tau_2\ *\ q_2)x \mapsto \begin{array}{l} \{q_1 = q_2 = \mathtt{WILD} \lor \tau_1 = \tau_2\} \cup \\ \{q_1 = \mathtt{WILD} \leftrightarrow q_2 = \mathtt{WILD}\} \cup \\ \{q_2 = \mathsf{SEQ} \rightarrow q_1 = \mathsf{SEQ}\} \end{array}}$$

$$\tau\ *\ q_1\ *\ q_2 \mapsto \{q_2 = \mathtt{WILD} \rightarrow q_1 = \mathtt{WILD}\}$$

# CCured normalizes constraints to 5 sorts.

$$\frac{x : \tau\ *\ q \qquad y : \mathtt{int}}{x + y \mapsto \{q \neq \mathtt{SAFE}\}} \qquad \frac{x : \mathtt{int} \qquad x \neq 0}{(\tau\ *\ q)x \mapsto \{q \neq \mathtt{SAFE}\}}$$

$$\frac{x : \tau_1\ *\ q_1}{(\tau_2\ *\ q_2)x \mapsto \begin{array}{l} \{q_1 = q_2 = \mathtt{WILD} \vee \tau_1 = \tau_2\} \cup \\ \{q_1 = \mathtt{WILD} \leftrightarrow q_2 = \mathtt{WILD}\} \cup \\ \{q_2 = \mathtt{SEQ} \to q_1 = \mathtt{SEQ}\} \end{array}}$$

$$\tau\ *\ q_1\ *\ q_2 \mapsto \{q_2 = \mathtt{WILD} \to q_1 = \mathtt{WILD}\}$$

- $q \neq \mathtt{SAFE}$
- $q_1 = q_2$
- $q = \mathtt{WILD}$
- $q_1 = \mathtt{WILD} \to q_2 = \mathtt{WILD}$
- $q_1 = \mathtt{SEQ} \to q_2 = \mathtt{SEQ}$

# CCured finds all the wild pointers.

$$\frac{x : \tau * q \qquad y : \mathtt{int}}{x + y \mapsto \{q \neq \mathsf{SAFE}\}}$$

$$\frac{x : \mathtt{int} \qquad x \neq 0}{(\tau * q)x \mapsto \{q \neq \mathsf{SAFE}\}}$$

$$\frac{x : \tau_1 * q_1}{(\tau_2 * q_2)x \mapsto \begin{array}{l}\{q_1 = q_2 = \mathtt{WILD} \lor \tau_1 = \tau_2\} \cup \\ \{q_1 = \mathtt{WILD} \leftrightarrow q_2 = \mathtt{WILD}\} \cup \\ \{q_2 = \mathsf{SEQ} \to q_1 = \mathsf{SEQ}\}\end{array}}$$

$$\tau * q_1 * q_2 \mapsto \{q_2 = \mathtt{WILD} \to q_1 = \mathtt{WILD}\}$$

- $q \neq \mathsf{SAFE}$
- $q_1 = q_2$
- $q = \mathtt{WILD}$
- $q_1 = \mathtt{WILD} \to q_2 = \mathtt{WILD}$
- $q_1 = \mathsf{SEQ} \to q_2 = \mathsf{SEQ}$

# CCured finds all the sequence pointers.

$$\frac{x : \tau * q \qquad y : \mathtt{int}}{x + y \mapsto \{q \neq \mathsf{SAFE}\}} \qquad \frac{x : \mathtt{int} \qquad x \neq 0}{(\tau * q)x \mapsto \{q \neq \mathsf{SAFE}\}}$$

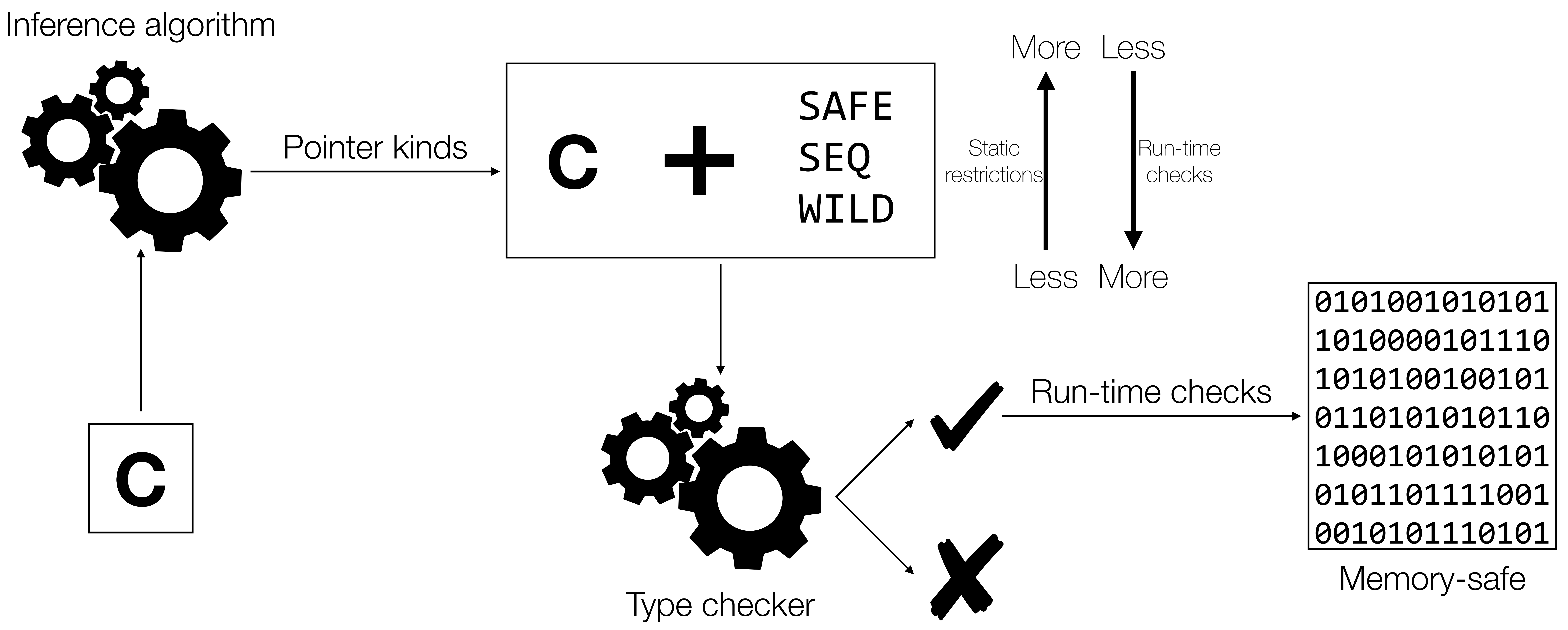
$$\frac{x : \tau_1 * q_1}{(\tau_2 * q_2)x \mapsto \begin{array}{l}\{q_1 = q_2 = \mathsf{WILD} \vee \tau_1 = \tau_2\} \cup \\ \{q_1 = \mathsf{WILD} \leftrightarrow q_2 = \mathsf{WILD}\} \cup \\ \{q_2 = \mathsf{SEQ} \rightarrow q_1 = \mathsf{SEQ}\}\end{array}}$$

$$\tau * q_1 * q_2 \mapsto \{q_2 = \mathsf{WILD} \rightarrow q_1 = \mathsf{WILD}\}$$

- $q \neq \mathsf{SAFE}$
- $q_1 = q_2$
- $q = \mathsf{WILD}$
- $q_1 = \mathsf{WILD} \rightarrow q_2 = \mathsf{WILD}$
- $\boxed{q_1 = \mathsf{SEQ} \rightarrow q_2 = \mathsf{SEQ}}$

# CCured can cure existing C programs.

| Name | Lines of Code | % sf/sq/w/rt | CCured Ratio | Purify Ratio | Memory Ratio | Lines Changed |
|---|---|---|---|---|---|---|
| SPECINT95 | | | | | | |
| compress | 1590 | 90/10/0/0 | 1.17 | 28 | 1.01 | 36 |
| go | 29,315 | 94/06/0/0 | 1.22 | 51 | 1.60 | 117 |
| ijpeg | 31,371 | 80/20/0/1 | 1.50 | 30 | 1.05 | 1103 |
| li | 7761 | 80/20/0/0 | 1.70 | 50 | 2.00 | 600 |
| Olden | | | | | | |
| bh | 2053 | 80/20/0/0 | 1.44 | 94 | 1.55 | 271 |
| bisort | 707 | 93/07/0/0 | 1.09 | 42 | 2.00 | 469 |
| em3d | 557 | 93/06/0/0 | 1.45 | 7 | 1.39 | 22 |
| health | 725 | 93/07/0/0 | 1.07 | 25 | 1.90 | 449 |
| mst | 617 | 97/03/0/0 | 1.87 | 5 | 1.15 | 44 |
| perimeter | 395 | 100/0/0/0 | 1.10 | 544 | 1.97 | 3 |
| power | 763 | 94/06/0/0 | 1.29 | 53 | 1.58 | 8 |
| treeadd | 385 | 96/04/0/0 | 1.15 | 500 | 2.61 | 14 |
| tsp | 561 | 100/0/0/0 | 1.06 | 66 | 2.54 | 7 |
| Ptrdist-1.1 | | | | | | |
| anagram | 661 | 88/12/0/0 | 1.43 | 34 | 1.52 | 37 |
| bc | 7323 | 77/23/0/0 | 9.91 | 100 | 2.18 | 58 |
| ft | 2194 | 98/02/0/0 | 1.03 | 12 | 2.12 | 59 |
| ks | 793 | 88/12/0/0 | 1.11 | 31 | 1.65 | 22 |
| yacr2 | 3999 | 88/12/0/0 | 1.56 | 26 | 1.63 | 30 |




# CCured: Type-Safe Retrofitting of Legacy Software