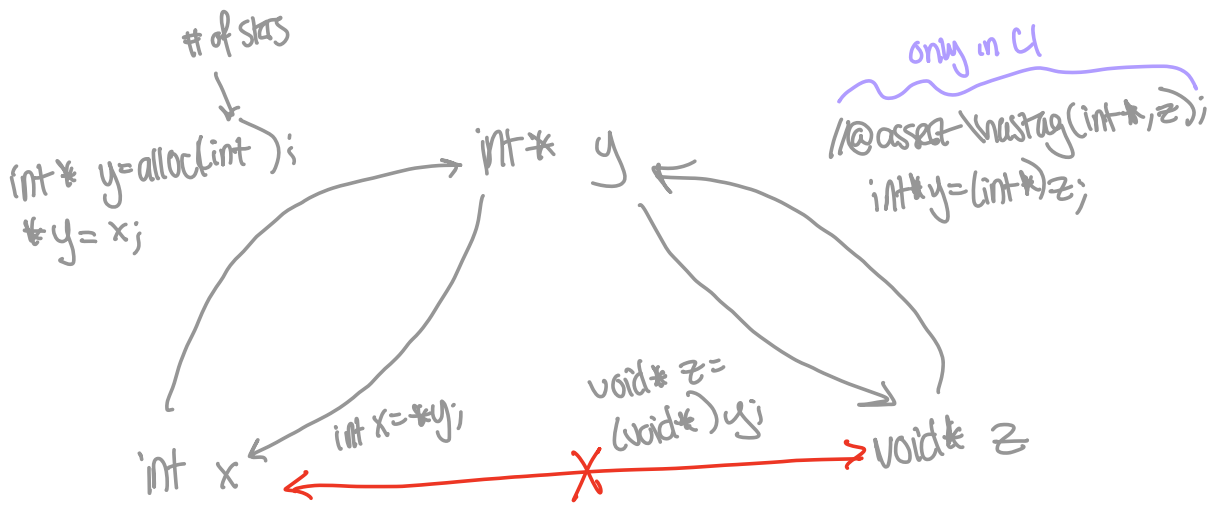
Generic Ptrs & Function Ptrs

Void* → generic pointer, can have any underlying ptee type (int*, char*, struct goose*, etc)

Converting (Cl)



Cl vs C (void*)

Cl:
- only cast to/from same ptee type

C:
- can cast any ptee type to void*
- can cast any ptee type to char*

ex) int *p = xcalloc(n, sizeof(int));
char *c = (char*)p;

FUNCTION POINTERS

Purpose: genericity

chosen, can be anything

Format:

typedef [return type] [name of function type] ([1 or more arguments type])

↳ typedef int hash_fn(void* key);

(types & # of arguments should match, exact parameter names don't)

```
typedef int    ptrs_to_int_fn(int *s, int *t);

int sum(int *s, int *t)
//@ requires s!=NULL && t!=NULL;
{
  return *s + *t;
}

int modulus(int *a, int *b) { }

int div(int *x, int *y) { ... }
```

To use:

to declare a function ptr "address of"

```
ptrs_to_int_fn *F = &sum;
int *a = calloc(int); *a = 100;
int *b = calloc(int); *b = 22;
int result = (*F)( a , b ); //result = 122
free(a);
free(b);
```

to apply the function

PROMPT:

Write a function that computes w/
typedef int compute_fn(int* a, int *b)

```
int add(int*x, int*y)
{
  return *x + *y;
}
```

```
int *compute(void* x, void* y, compute_fn * F)
//@requires x!=NULL && \hastag(int *, x);
//@requires y!=NULL && \hastag(int *, y);
//@requires F!=NULL
{
    int *res = xmalloc(sizeof(int));
    *res = (*F)((int*)x, (int *)y));
    return res;
}
```

* note: \hastag checks validity in C1! *