C Language
Data and Pointers

Challenges

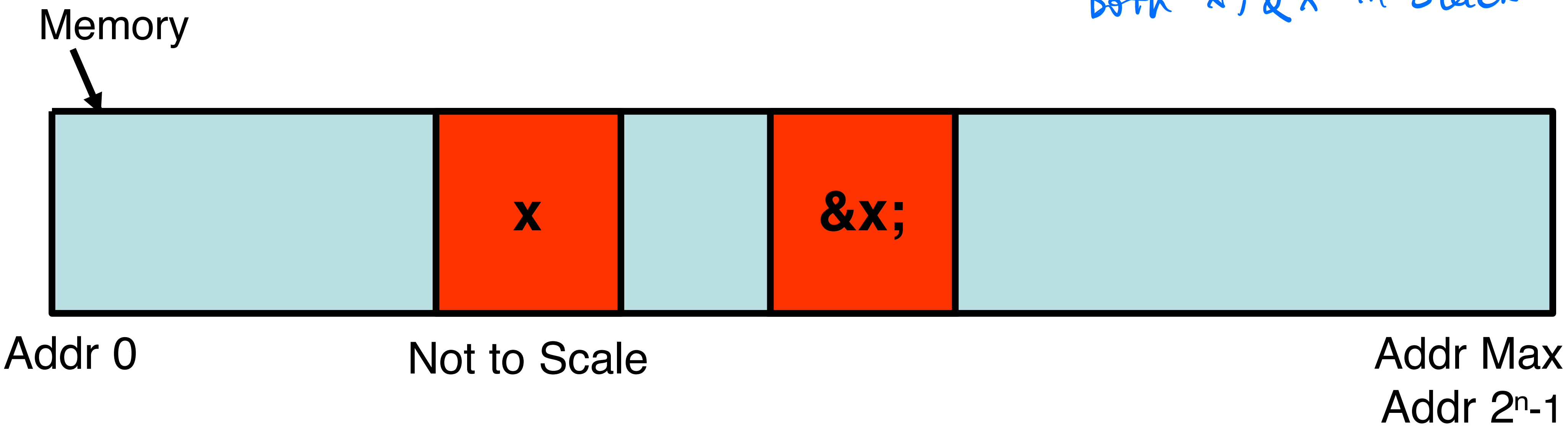
- Your program must operate on data objects and memory references (pointers) in concert
- And there is not really any required binding between the two (i.e., up to the programmer to keep it right)
- And if there is an error due to a pointer, it can be hard to visualize

Objects and Pointers

- **How primitive objects are accessed using pointers is controlled by the programmer**

```
void main(int argc, char *argv[])
  { int x; fib(argv[1], &x); printf("%d", x); }
```

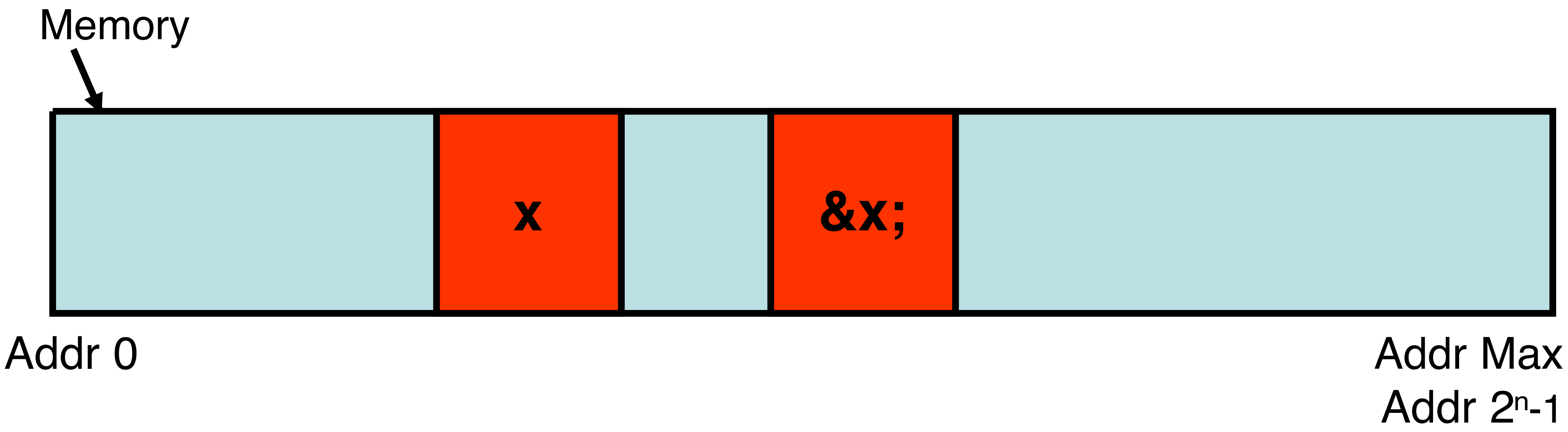
both x, &x in stack



Objects and Pointers

- **What value is &x?**

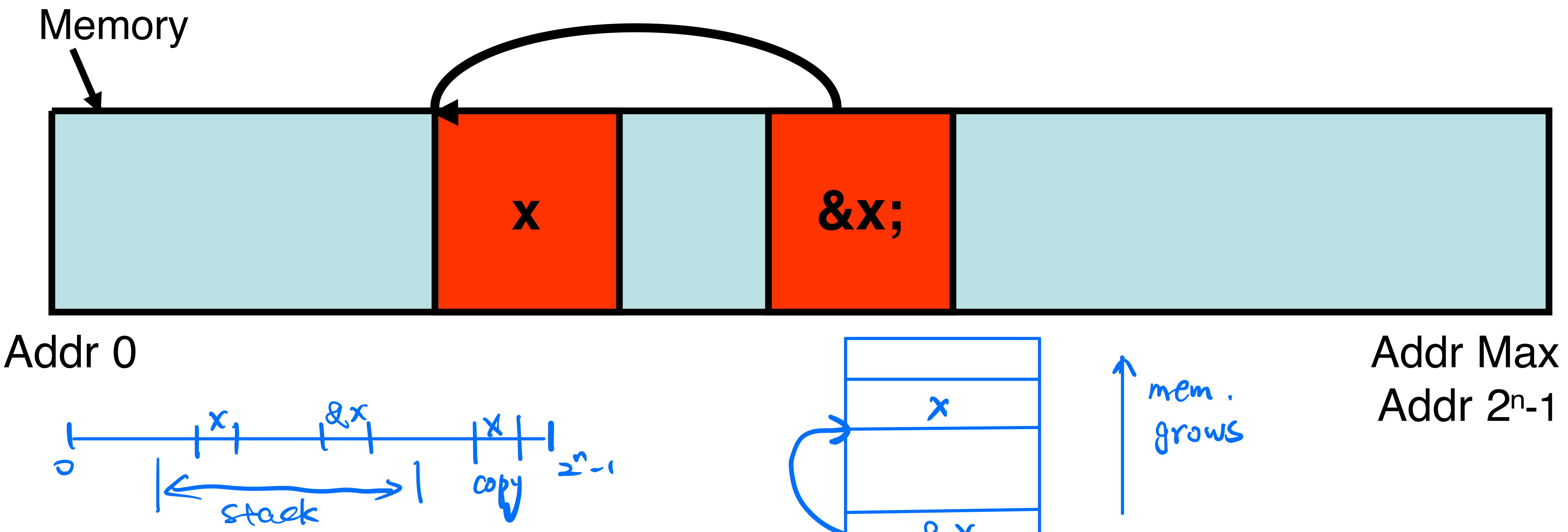
```
void main(int argc, char *argv[])
  { int x; fib(argv[1], &x); printf("%d", x); }
```



Objects and Pointers

- **Why do we pass &x to the callee (fib)?**

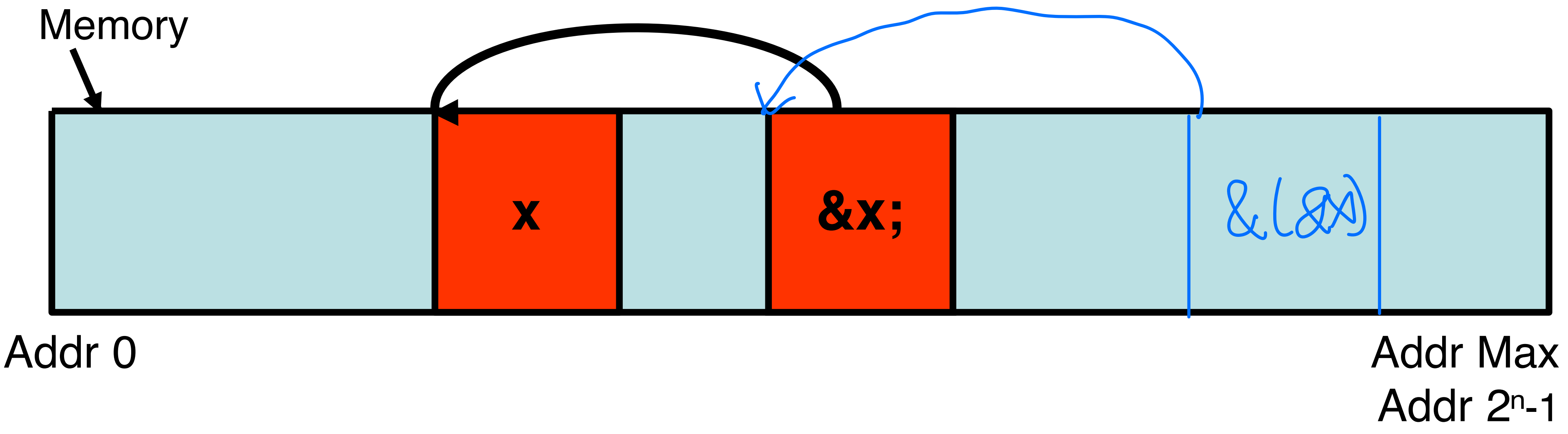
```
void main(int argc, char *argv[])
  { int x; fib(argv[1], &x); printf("%d", x); }
```



Objects and Pointers

- **What value is &(&x)?** address of pointer of x

```
void main(int argc, char *argv[])
  { int x; fib(argv[1], &x); printf("%d", x); }
```

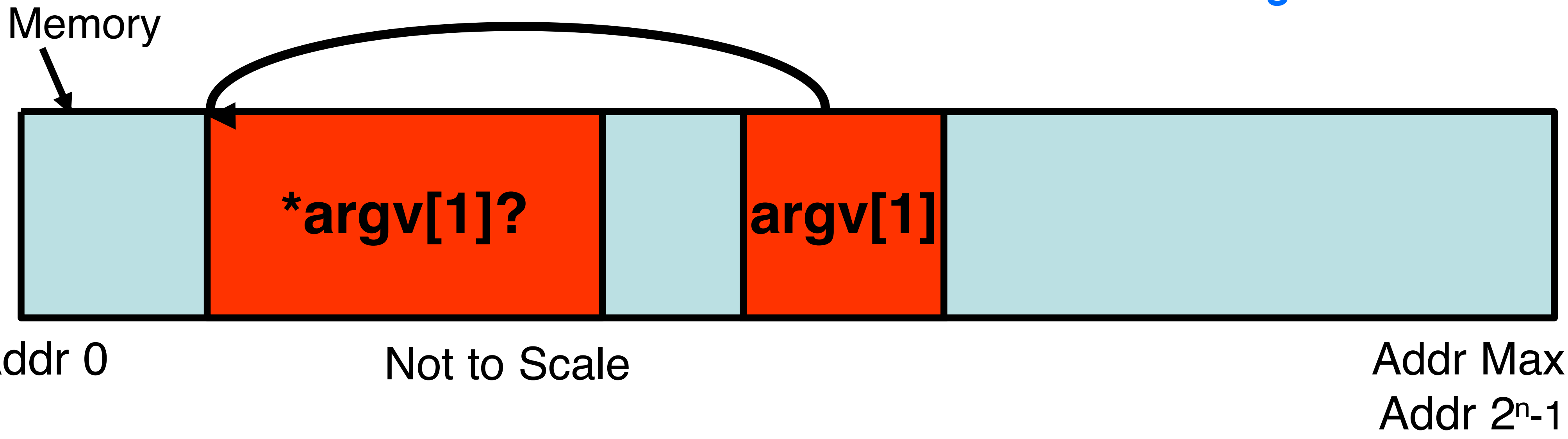


"Strings"

- **And C fakes strings as a sequence of bytes ending in a null byte (terminator)**

```
void main(int argc, char *argv[])
  { printf("%s", argv[1]); }
```

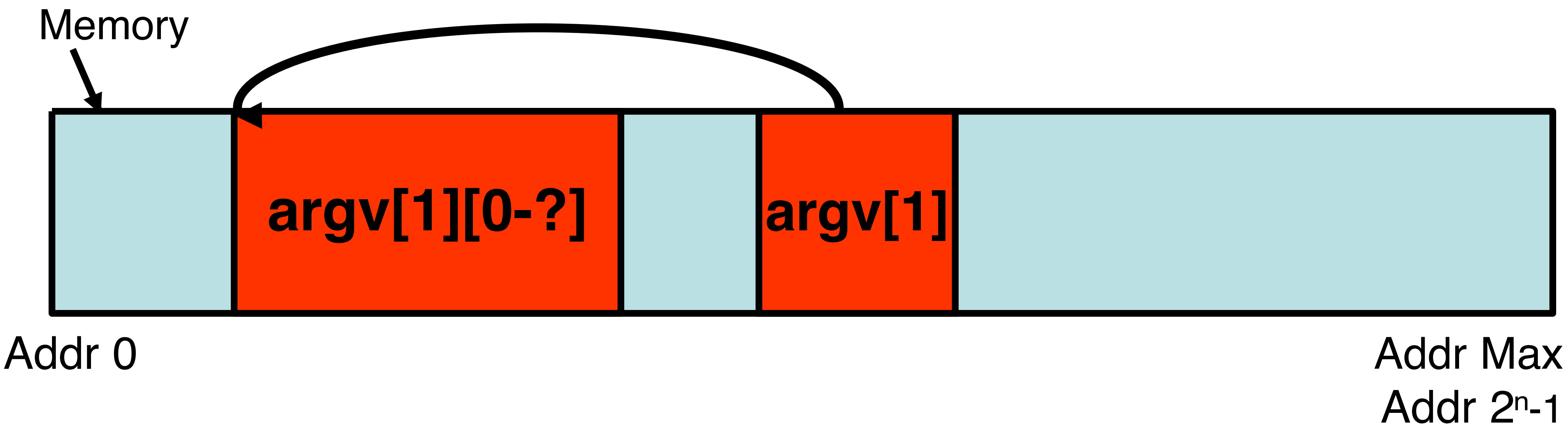
argv contains command line arguments



“Strings”

- **Causes no end of grief for programmers**

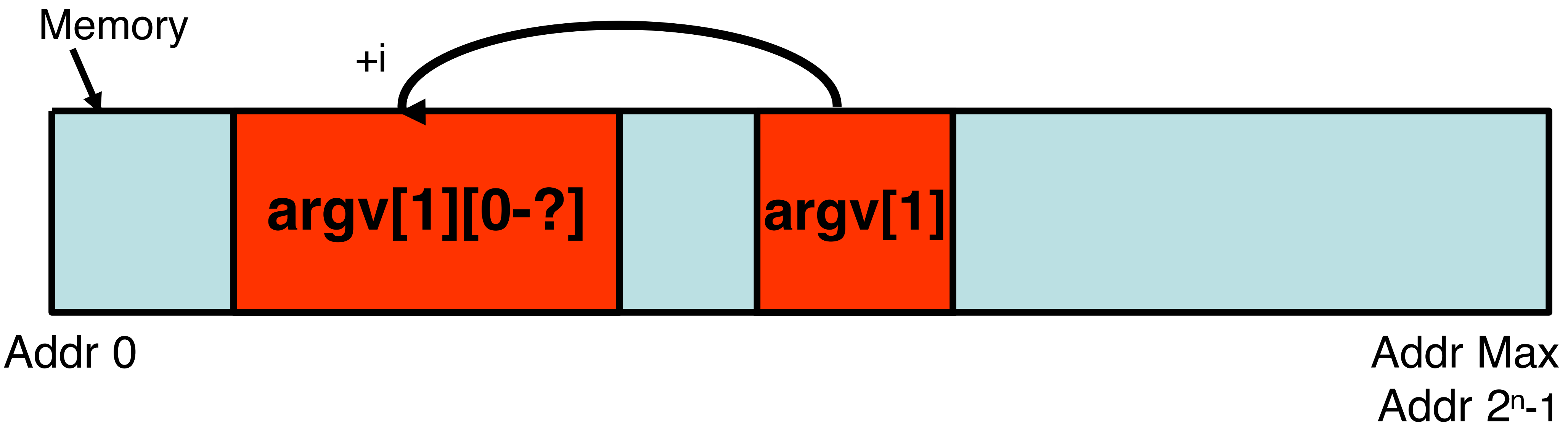
```
void main(int argc, char *argv[])
  { printf(“%s”, argv[1]); }
```



"Strings"

- **Character at a time**

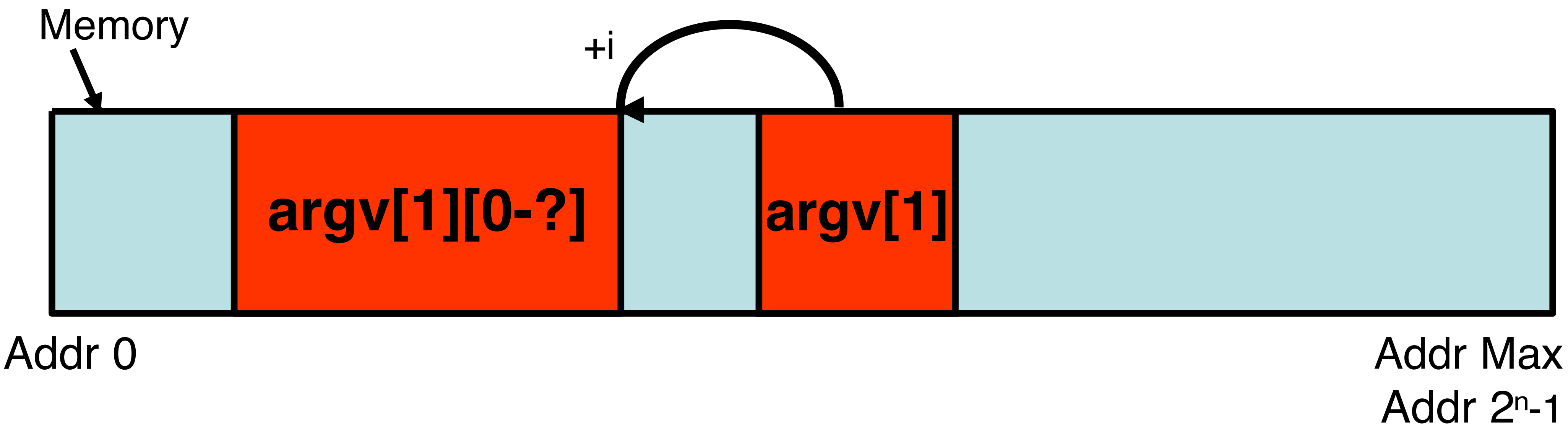
```
void main(int argc, char *argv[])
  { int i; for(i = 0; i++; i<strlen(argv[1])) {
   printf("%c", argv[1][i]); } }
```



“Strings”

- **Why does this loop terminate?**

```
void main(int argc, char *argv[])
  { int i; for(i = 0; i++; i<strlen(argv[1])) {
   printf(“%c”, argv[1][i]); } }
```

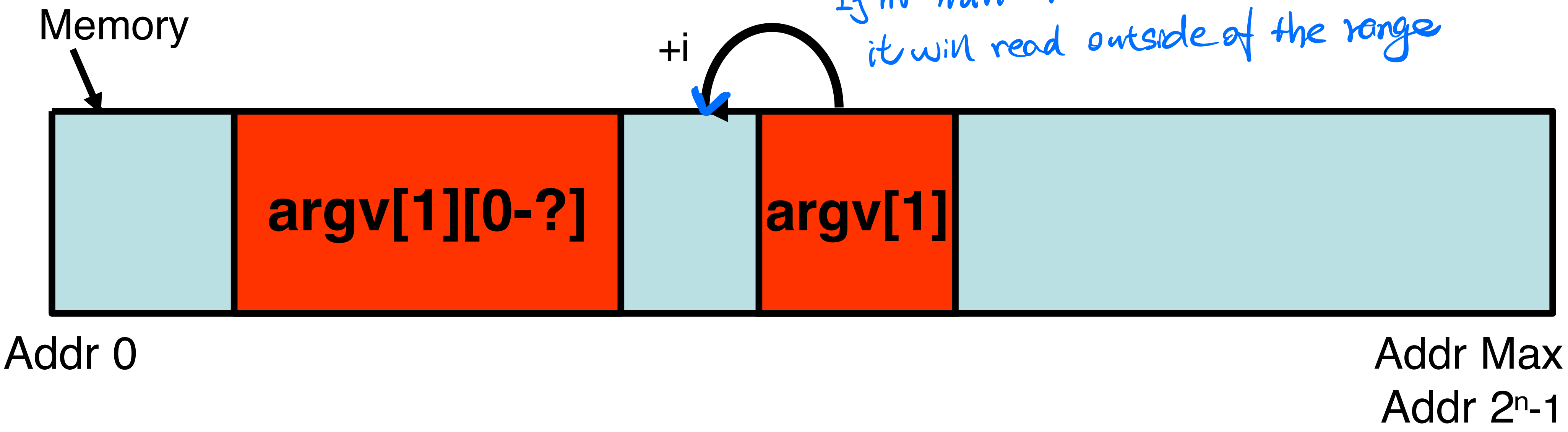


“Strings”

- **Is a null-terminating byte guaranteed?**

```
void main(int argc, char *argv[])
  { int i; for(i = 0; i++; i<strlen(argv[1])) {
   printf(“%c”, argv[1][i]); } }
```

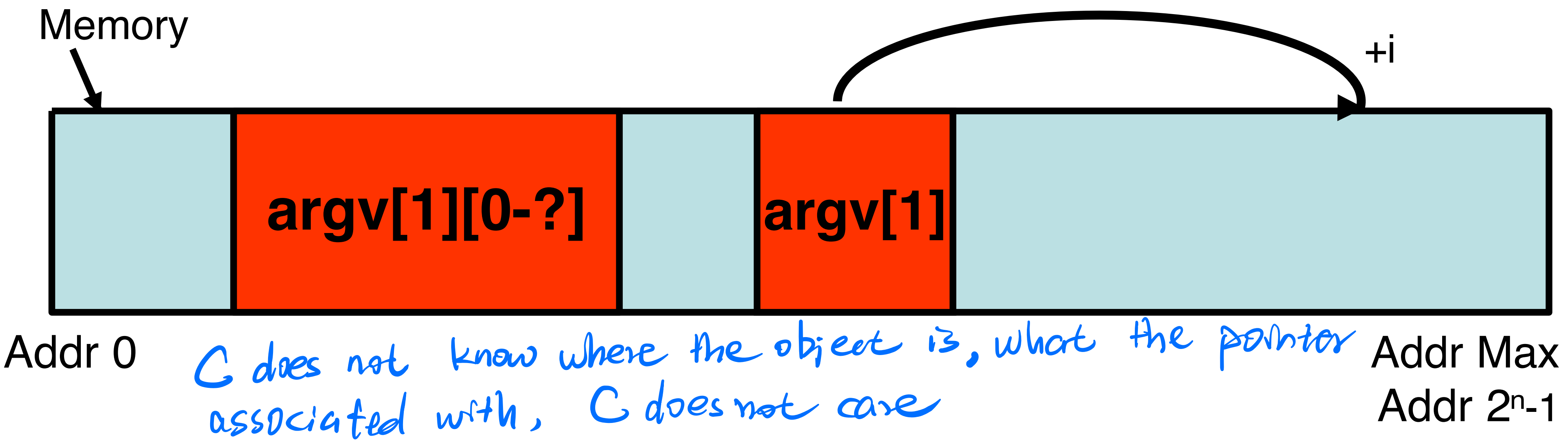
If no null-terminater it will read outside of the range



"Strings"

- **Pointers are independent of data objects**

```
void main(int argc, char *argv[])
  { int i; for(i = 0; i++; i<strlen(argv[1])) {
   printf("%c", argv[1][i]); } }
```

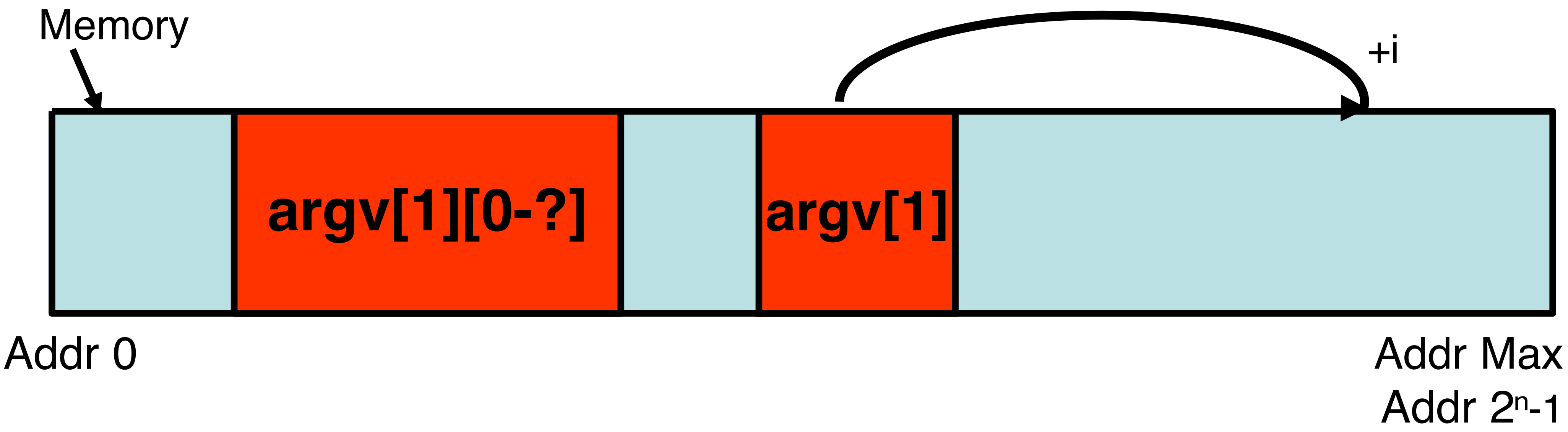


C does not know where the object is, what the pointer associated with, C does not care

“Strings”

- **Programmers must keep them in sync**

```
void main(int argc, char *argv[])
  { int i; for(i = 0; i++; i<strlen(argv[1])) {
   printf(“%c”, argv[1][i]); } }
```



Ints/Strings and Pointers

- C programs have many pointers
- And many bugs are the result of pointer errors
- The debugger helps you look at the pointer values quickly, but you need to know where they belong to find errors
  - Argv[1] value is?
  - How long should argv[1]'s data object (array) be?
  - Is the argv[1] pointer referencing a different data object?

PENNSTATE

1855

Questions?