

Say we write a program that only *ever* uses 7 variables:

```
double x1;
```

```
double x2;
```

```
double x3;
```

```
double x4;
```

```
double x5;
```

```
double x6;
```

```
double x7;
```

Each variable can only store *one* value at a time:

```
x1 = 5.2;
```

```
x2 = 9.3;
```

```
x3 = -10.1;
```

```
cin >> x4;
```

```
cin >> x5;
```

```
cin >> x6;
```

```
cin >> x7;
```

Suppose we want to process a stream of data...

-11.35

39.32

28.56

-5.30

-29.34

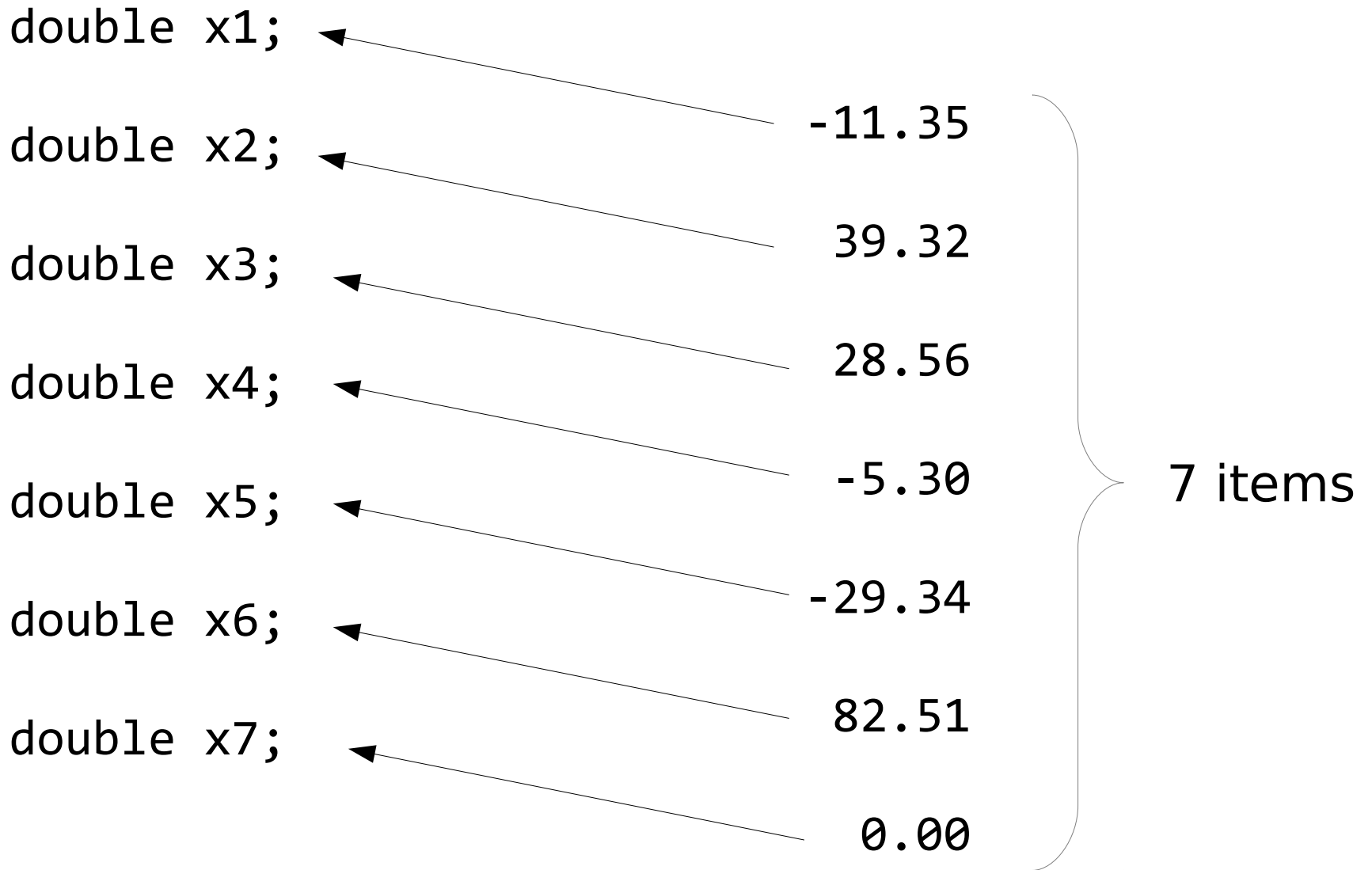
82.51

0.00



7 items

No problem. We have 7 variables.



What if we have 8 items?

-11.35

39.32

28.56

-5.30

-29.34

82.51

0.00

2.03



8 items

We don't have enough variables...

double x1;	←	
double x2;	←	-11.35
double x3;	←	39.32
double x4;	←	28.56
double x5;	←	-5.30
double x6;	←	-29.34
double x7;	←	82.51
???	←	0.00
		2.03

8 items

The solution is *not* to add another variable...

double x1;	←		
double x2;	←	-11.35	}
double x3;	←	39.32	
double x4;	←	28.56	
double x5;	←	-5.30	
double x6;	←	-29.34	
double x7;	←	82.51	
double x8;	←	0.00	
→		2.03	

Because what if we actually have 9 items?


double x1;		
double x2;		-11.35
double x3;		39.32
double x4;		28.56
double x5;		-5.30
double x6;		-29.34
double x7;		82.51
double x8;		0.00
???		2.03
		10.55

9 items

The diagram shows a list of 9 items. The first item is 'double x1;' with an arrow pointing to the right. The next eight items are 'double x2;' through 'double x8;', each with an arrow pointing to a numerical value. The ninth item is '???' with an arrow pointing to the value '2.03'. Below '2.03' is the value '10.55'. A large curly bracket on the right side groups the values from '-11.35' to '2.03' and is labeled '9 items'.

Because what if we actually have 10 items?

double x1;	
double x2;	-11.35
double x3;	39.32
double x4;	28.56
double x5;	-5.30
double x6;	-29.34
double x7;	82.51
double x8;	0.00
???	2.03
???	10.55
	-2.60



10 items

Because what if we actually have 1 million items?

double x1;	
double x2;	-11.35
double x3;	39.32
double x4;	28.56
double x5;	-5.30
double x6;	-29.34
double x7;	82.51
double x8;	0.00
???	2.03
???	10.55
???	-2.60
???	7.98
???	12.01
???	-72.77

1 million items

Our programs can't grow once they're compiled, put in a machine, and the machine is deployed...

(At least, not C++ programs.)

So if we *ever* (anywhere in our program, even inside loops) mention only 7 variables, we can only store 7 values at once.

So how do we process *arbitrary* streams of data, whose size cannot be known ahead of time?

If we are only summing the data, we don't need to keep all of it.

We can just add every item to a sum, then forget that item and move on to the next.

But what if we wanted to find the median, the mode, the standard deviation, or sort the data?

In these cases...

We have to store *all* the data *somewhere*.

We have big "memories" (say, 4GB)...

Can't we put data in there?

But we still have this issue of a fixed number of variables; once we compile our code, we can't add more variables.

Here is one solution.

Reserve two variables.

(for this solution, we always need *only* two variables)

Now, we want to prepare to "store" (in memory) the first item from the stream.

??? ← -11.35

Before we get that item, we ask for a chunk of memory to be "reserved" for our use.

```
double* first = (give me a memory location...)
```

We save this location in `first` (which is a pointer).

location: 2096



Before we get that item, we ask for a chunk of memory to be "reserved" for our use.

```
double* first = (give me a memory location...)
```

We save this location in `first` (which is a pointer).

Now, put the item from the stream in that memory chunk.

```
cin >> *first; // * means "go to that location"
```

location: 2096 -11.35 ← -11.35

Before we get the *second* item, we have to reserve another memory chunk. Let's reserve the *next* chunk:

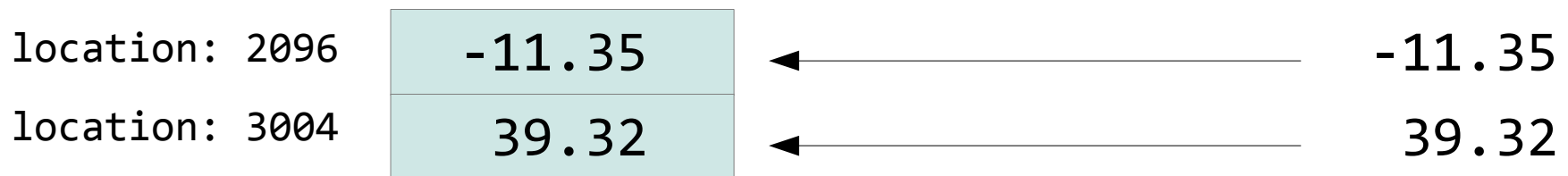
(reserve the chunk starting at "first + 1"...)

We don't save this location because, well, we know where it is:

`first + 1`

Save the item in that location:

```
cin >> *(first + 1);
```



Do it again for the third item:

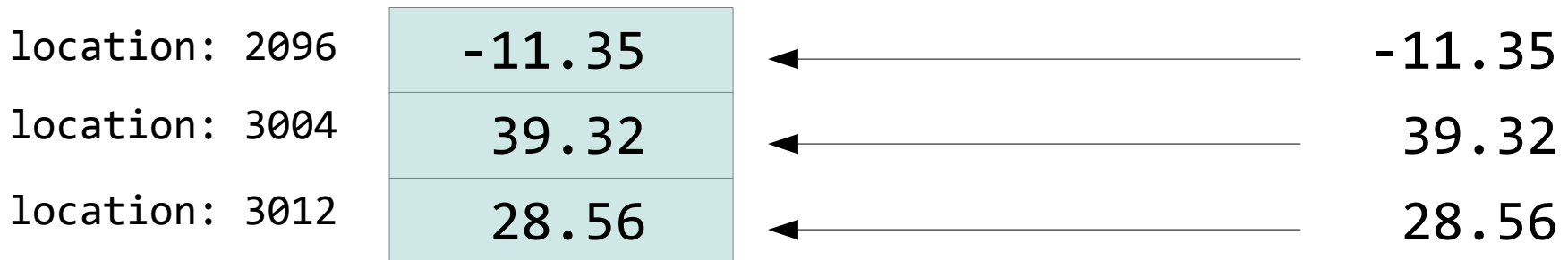
(reserve the chunk starting at "first + 2"...)

We don't save this location because, well, we know where it is:

`first + 2`

Save the item in that location:

```
cin >> *(first + 2);
```



Do it again for the fourth item:

(reserve the chunk starting at "first + 3"...)

We don't save this location because, well, we know where it is:

`first + 3`

Save the item in that location:

```
cin >> *(first + 3);
```

location: 2096	-11.35	←	-11.35
location: 3004	39.32	←	39.32
location: 3012	28.56	←	28.56
location: 3020	-5.30	←	-5.30

And so on...

± 2 3 million times...

location: 2096	-11.35	←	-11.35
location: 3004	39.32	←	39.32
location: 3012	28.56	←	28.56
location: 3020	-5.30	←	-5.30
location: 3028	-29.34	←	-29.34
location: 3032	82.51	←	82.51
location: 3040	0.00	←	0.00
location: 3048	2.03	←	2.03

etc...

This entire time, we've only used our two reserved variables:

```
double* first
```

and which other?

We kept adding to `first`; we added the number of data items we had already seen. So we need a count variable:

```
int count
```

That makes our calculations before look like this:

```
(reserve the chunk starting at "first + count"...)
```

```
cin >> *(first + count);
```

So the problem of storing an *arbitrary* amount of data with a *fixed-size* program has been solved.

We created the concept of an "array."

But consider this slight variation:

We don't know how big each item is. Perhaps they are words, not numbers.

Some words are tiny, some are huge...

So how much memory we need to reserve is different for each item.

`(first + count)` is not going to suffice.