Understand arrays in memory

Understand how to access array's element

Understand fixed and variable-size array

Arrays

① Array

② Nested Array

③ Fixed-size Array

④ Variable-size Array

Array Allocation

T A[N];

```
char A[12];
```

```
int B[6];
```

```
double C[3];
```

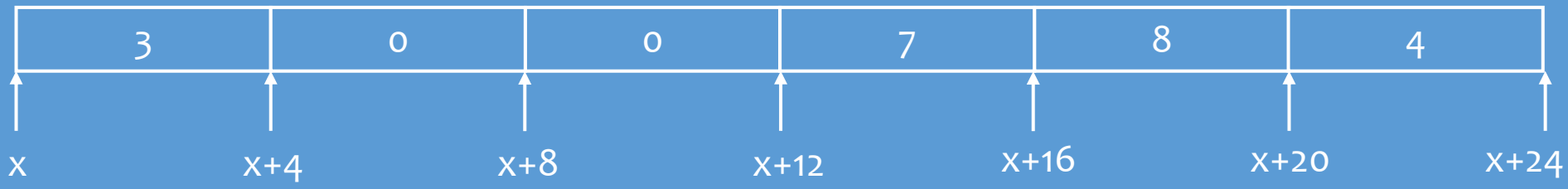
```
char* D[3];
```

```
double* E[3];
```

Array Allocation

T A[N];

```
int val[6];
```



Reference	Type	Value
val[4]	int	?
val	int*	?
val+1	int*	?
&val[2]	int*	?
val[6]	int	?
*(val+1)	int	?
val+i	int*	?

```
short  S[7];
short  *T[3];
short  **U[6];
int    V[8];
double *W[4];
```

Array	Element size	Total size	Start address	Element i
S	?	?	x_S	?
T	?	?	x_T	?
U	?	?	x_U	?
V	?	?	x_V	?
W	?	?	x_W	?

Integer array E
Address in %rdx
Index in %rcx
Result in %eax (data), %rax (pointer)

Expression	Type	Value	Assembly Code
E	int*	x_E	?
E[0]	int	$M[x_E]$	?
E[i]	int	$M[x_E+4i]$	?
&E[2]	int*	x_E+8	?
E+i-1	int*	x_E+4i-4	?
*(E+i-3)	int	$M[x_E+4i-12]$	?
&E[i]-E	long	i	?

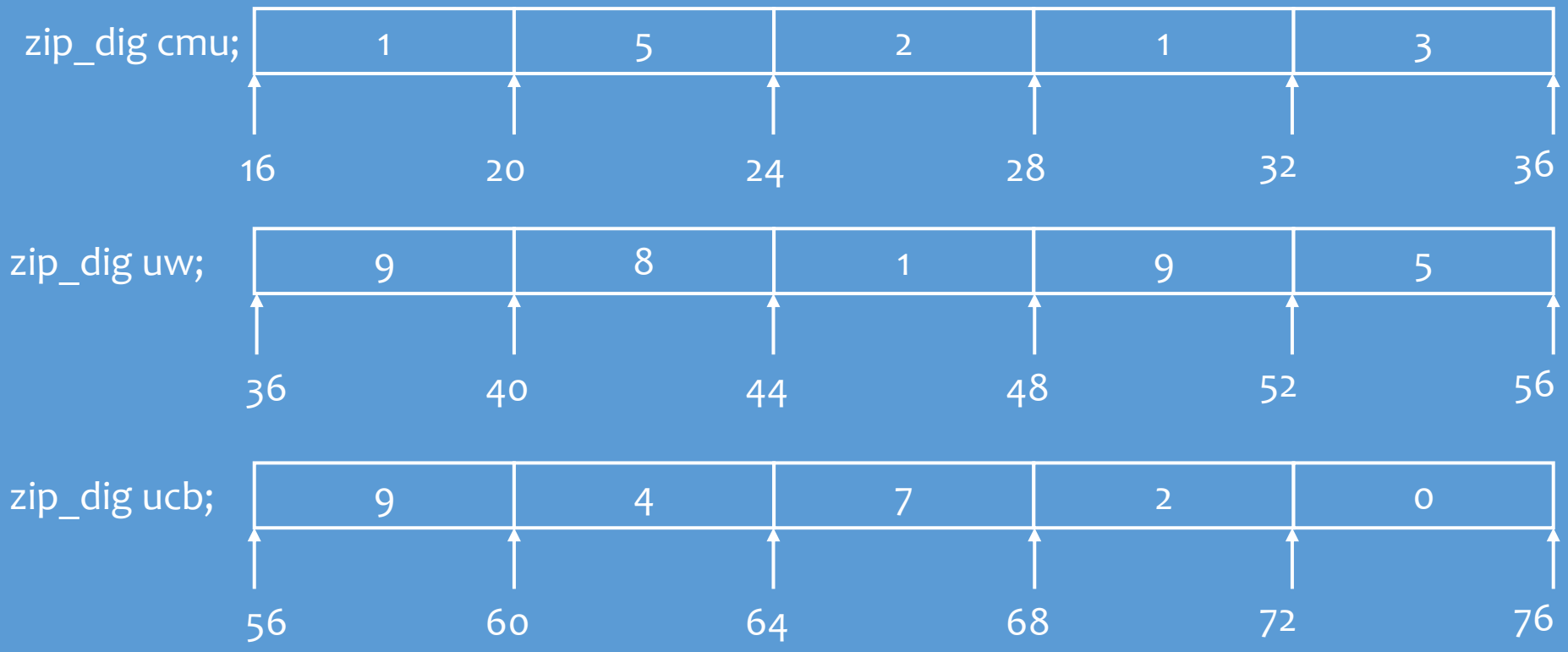
Short integer array S
Address in %rdx
Index in %rcx
Result in %ax (data), %rax (pointer)

Expression	Type	Value	Assembly Code
S+1	?	x_S+2	?
S[3]	?	$M[x_S+6]$	?
&S[i]	?	x_S+2i	?
S[4*i+1]	?	$M[x_S+8i+2]$	?
S+i-5	?	$x_S+2i-10$	?

```
typedef int zip_dig[5];

zip_dig cmu = { 1, 5, 2, 1, 3 };
zip_dig uw  = { 9, 8, 1, 9, 5 };
zip_dig ucb = { 9, 4, 7, 2, 0 };
```

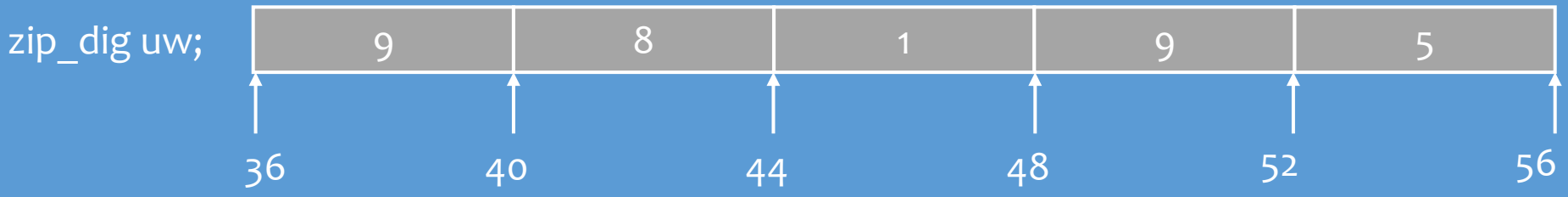
uw[3] = ?
uw[6] = ?
uw[-1] = ?
cmu[15] = ?



```
int get_digit
  (zip_dig z, int dig)
{
  return z[dig];
}
```

```
# %edx = z
# %eax = dig
movl (%edx,%eax,4),%eax
```

Register %edx contains starting address of array
Register %eax contains array index



```
int zd2int(zip_dig z)
{
  int i;
  int zi = 0;
  for (i = 0; i < 5; i++) {
    zi = 10 * zi + z[i];
  }
  return zi;
}
```

```
int zd2int(zip_dig z)
{
  int zi = 0;
  int *zend = z + 4;
  do {
    zi = 10 * zi + *z;
    z++;
  } while (z <= zend);
  return zi;
}
```

```
   xorl %eax,%eax
   leal 16(%ecx),%ebx
.L59:
   leal (%eax,%eax,4),%edx
   movl (%ecx),%eax
   addl $4,%ecx
   leal (%eax,%edx,2),%eax
   cmpl %ebx,%ecx
   jle .L59
```

Registers

%ecx	z
%eax	zi
%ebx	zend

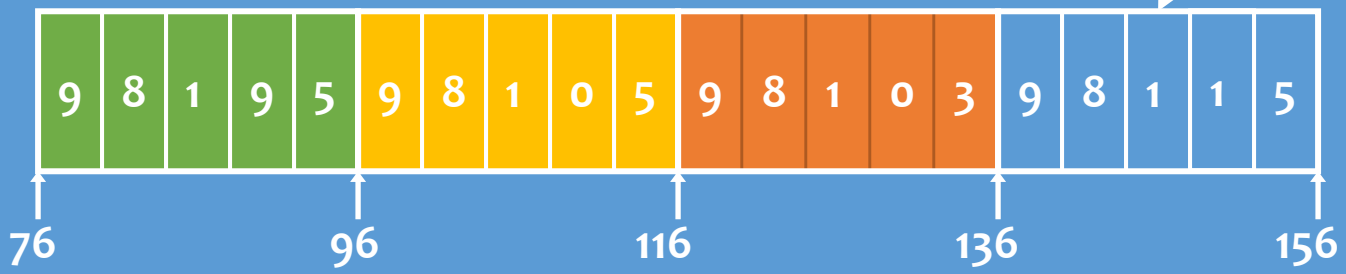
```
#define PCOUNT 4
zip_dig sea[PCOUNT] =
  {{ 9, 8, 1, 9, 5 },
   { 9, 8, 1, 0, 5 },
   { 9, 8, 1, 0, 3 },
   { 9, 8, 1, 1, 5 }};
```

Address

A + i * (C * K) + j * K

= A + (i * C + j)* K

sea[3][2];



sea[3][3] =

sea[2][5] =

sea[2][-1] =

sea[4][-1] =

sea[0][19] =

sea[0][-1] =

Nested array

```
T  D[R][C];
```

`D[i][j]`

$\&D[i][j] = x_D + L(Ci+j)$

int A[5][3];

Address x_A in %rdi

i in %rsi

j in %rdx

Copy A[i][j] to %eax:

```
leaq    (%rsi,%rsi,2),%rax
leaq    (%rdi,%rax,4),%rax
movl    (%rax,%rdx,4),%eax
```

```
#define:
long P[M][N];
long Q[N][M];

long sum_element(long i, long j) {
   return P[i][j] + Q[j][i];
}
```

M=5
N=7

&D[i][j]
$=x_D+L(Ci+j)$

i in %rdi
j in %rsi

```
sum_element:
   leaq  0(,%rdi,8),%rdx
   subq  %rdi,%rdx
   addq  %rsi,%rdx
   leaq  (%rsi,%rsi,4),%rax
   addq  %rax,%rdi
   movq  Q(,%rdi,8),%rax
   addq  P(,%rdx,8),%rax
   ret
```

Summary

- Array
- Nested Array
- Multi-level Array

Charles Petzold

American programmer, Microsoft MVP

> Programming in machine code is like eating with a toothpick.