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];
```

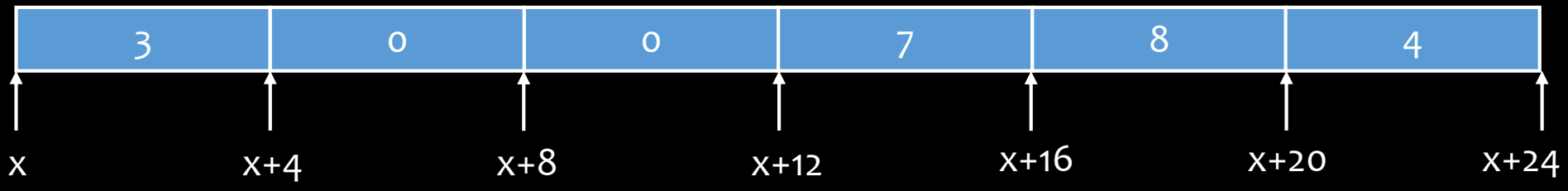
```
char* D[3];
```

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

# Array Allocation

T A[N];

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




| Reference | Type | Value |
|---|---|---|
| val[4] | int | 8 |
| val | int* | x |
| val+1 | int* | x+4 |
| &val[2] | int* | x+8 |
| val[6] | int | ?? |
| *(val+1) | int | 0 |
| val+i | int* | x+4i |

```
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 | 2 | 14 | $x_S$ | $x_S+2i$ |
| T | 8 | 24 | $x_T$ | $x_T+8i$ |
| U | 8 | 48 | $x_U$ | $x_U+8i$ |
| V | 4 | 32 | $x_V$ | $x_V+4i$ |
| W | 8 | 32 | $x_W$ | $x_W+8i$ |

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

| Expression | Type | Value | Assembly Code |
|---|---|---|---|
| E | int* | $x_E$ | movl %rdx,%rax |
| E[0] | int | M[$x_E$] | movl (%rdx),%eax |
| E[i] | int | M[$x_E$+4i] | movl (%rdx,%rcx,4),%eax |
| &E[2] | int* | $x_E$+8 | leaq 8(%rdx),%rax |
| E+i-1 | int* | $x_E$+4i-4 | leaq -4(%rdx,%rcx,4),%rax |
| *(E+i-3) | int | M[$x_E$+4i-12] | movl -12(%rdx,%rcx,4),%eax |
| &E[i]-E | long | i | movq %rcx,%rax |

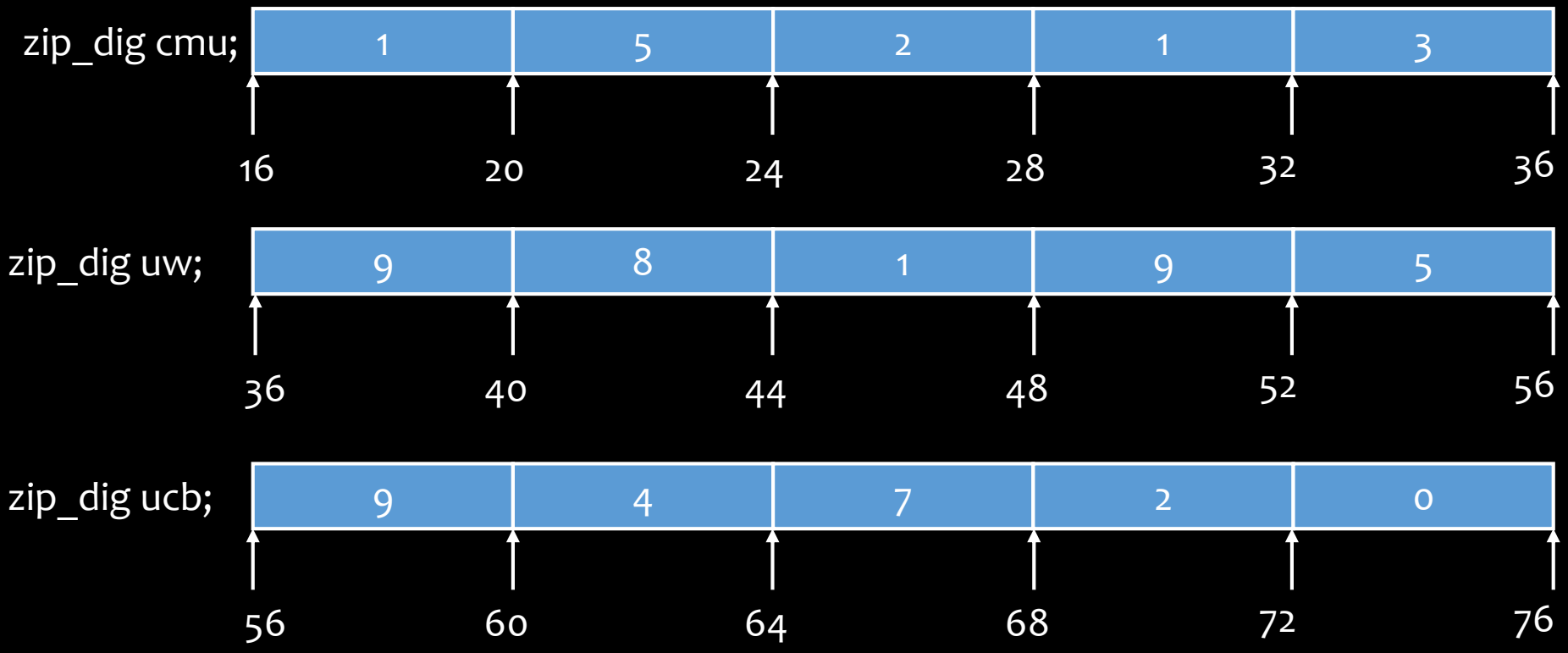
Short integer array S
Address in %rdx
Index in %rcx
Result in %ax (data), %rax (pointer)

| Expression | Type | Value | Assembly Code |
|---|---|---|---|
| S+1 | short* | $x_S$+2 | leaq 2(%rdx),%rax |
| S[3] | short | M[$x_S$+6] | movw 6(%rdx),%ax |
| &S[i] | short* | $x_S$+2i | leaq (%rdx,%rcx,2),%rax |
| S[4*i+1] | short | M[$x_S$+8i+2] | movw 2(%rdx,%rcx,8),%ax |
| S+i-5 | short* | $x_S$+2i-10 | leaq -10(%rdx,%rcx,2),%rax |

```
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] = 9
uw[6] = 4
uw[-1] = 3
cmu[15] = ?

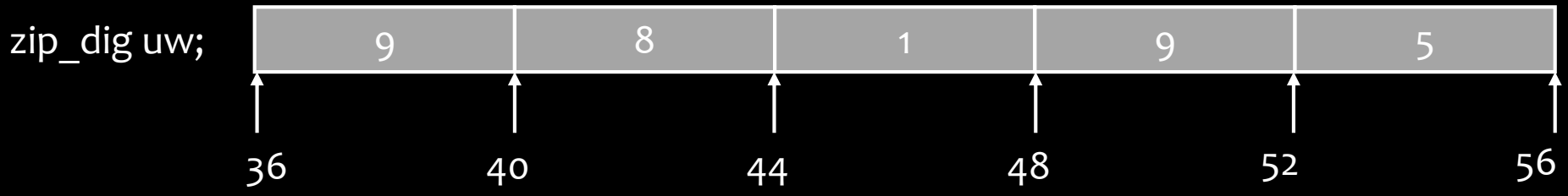


```
int get_digit
  (zip_dig z, int dig)
{
  return z[dig];
}
```

```
# %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;
}
```

```
    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
```

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

Registers

| | |
|---|---|
| %ecx | z |
| %eax | zi |
| %ebx | zend |

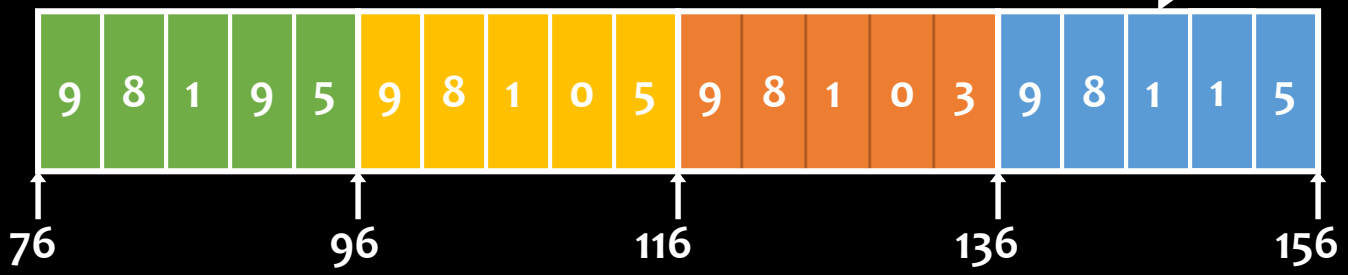
```
#define PCOUNT 4
zip_dig sea[PCOUNT] =
  {{ 9, 8, 1, 9, 5 },
   { 9, 8, 1, 0, 5 },
   { 9, 8, 1, 0, 3 },
   { 9, 8, 1, 1, 5 }};
```

Address

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

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

sea[3][2];




sea[3][3] = 1

sea[2][5] = 5

sea[2][-1] = 5

sea[4][-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

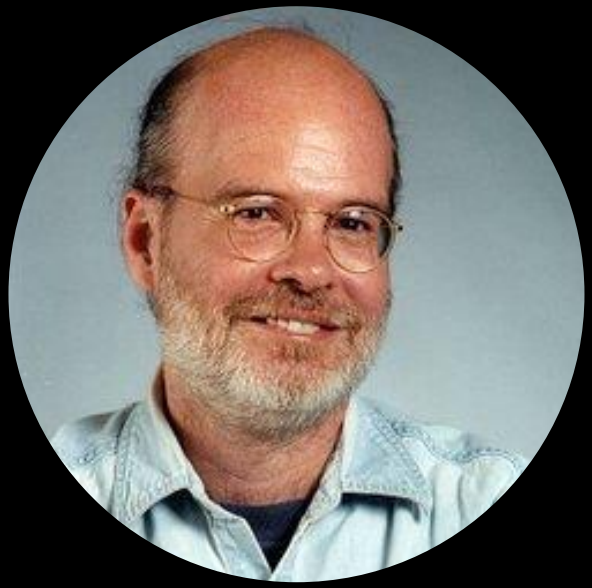
&D[i][j]
$=x_D+L(Ci+j)$

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.