Exam Review

CS351: Systems Programming
Day 15: Oct. 11, 2022

Instructor:
Nik Sultana

Today

- **Exam & Grade Structure**
- Demo Test review
- Course Review

Exam Structure

- **Similar structure & interface to the Demo Test**
- **Open book/notes/Internet**
- **Individual exam**
- **Duration: 45 minutes**
  - Exam window opens at 08:30 and closes at 10:00.
  - Don’t spend too long on a question: if stuck, move to the next question and come back to it later.
- **10 questions spanning everything we’ve covered so far.**
- **Max marks: 120**
  i.e., can boost final grade by 5%
- **Exam is online but being on campus gives you best chance of getting technical support.**

Grade Structure

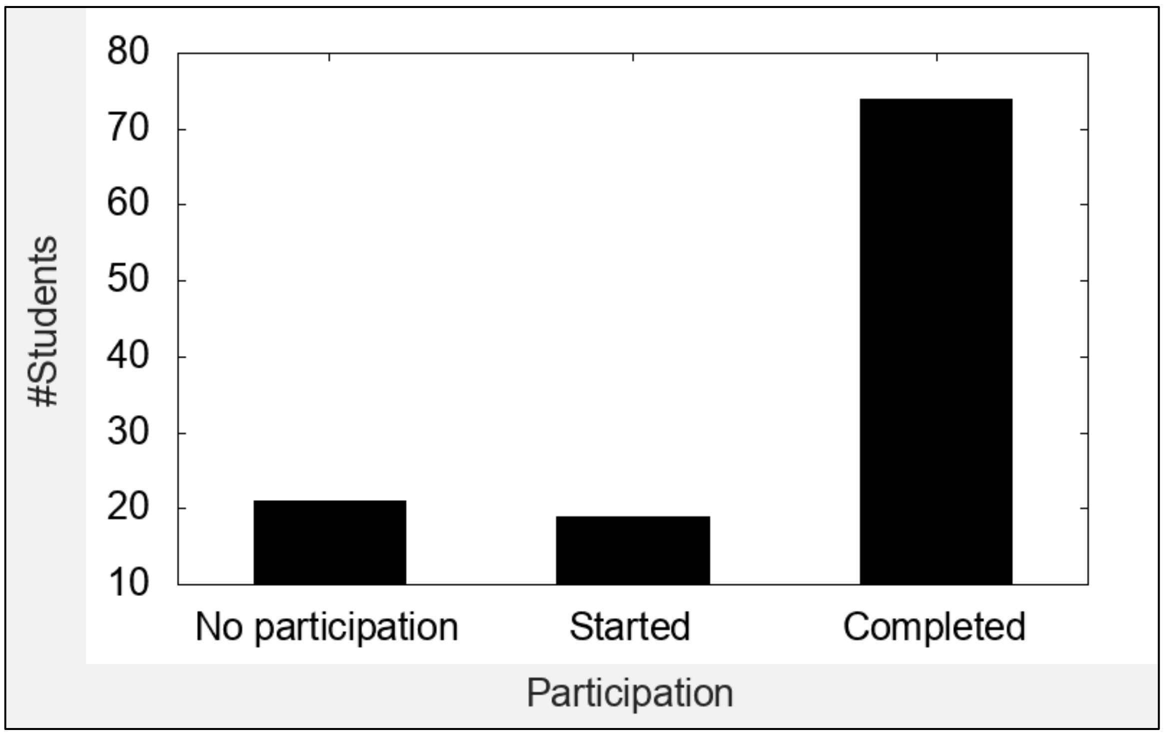
- **Midterm grade != midterm exam**
- **Midterm grade** mirrors the final grade structure:
  - 50% labs (i.e., labs 1 and 2 in this case)
  - 50% midterm exam
- **On Blackboard you'll see the Mid-term grade, the mid-term exam marks, and you can already see lab marks.**

Today

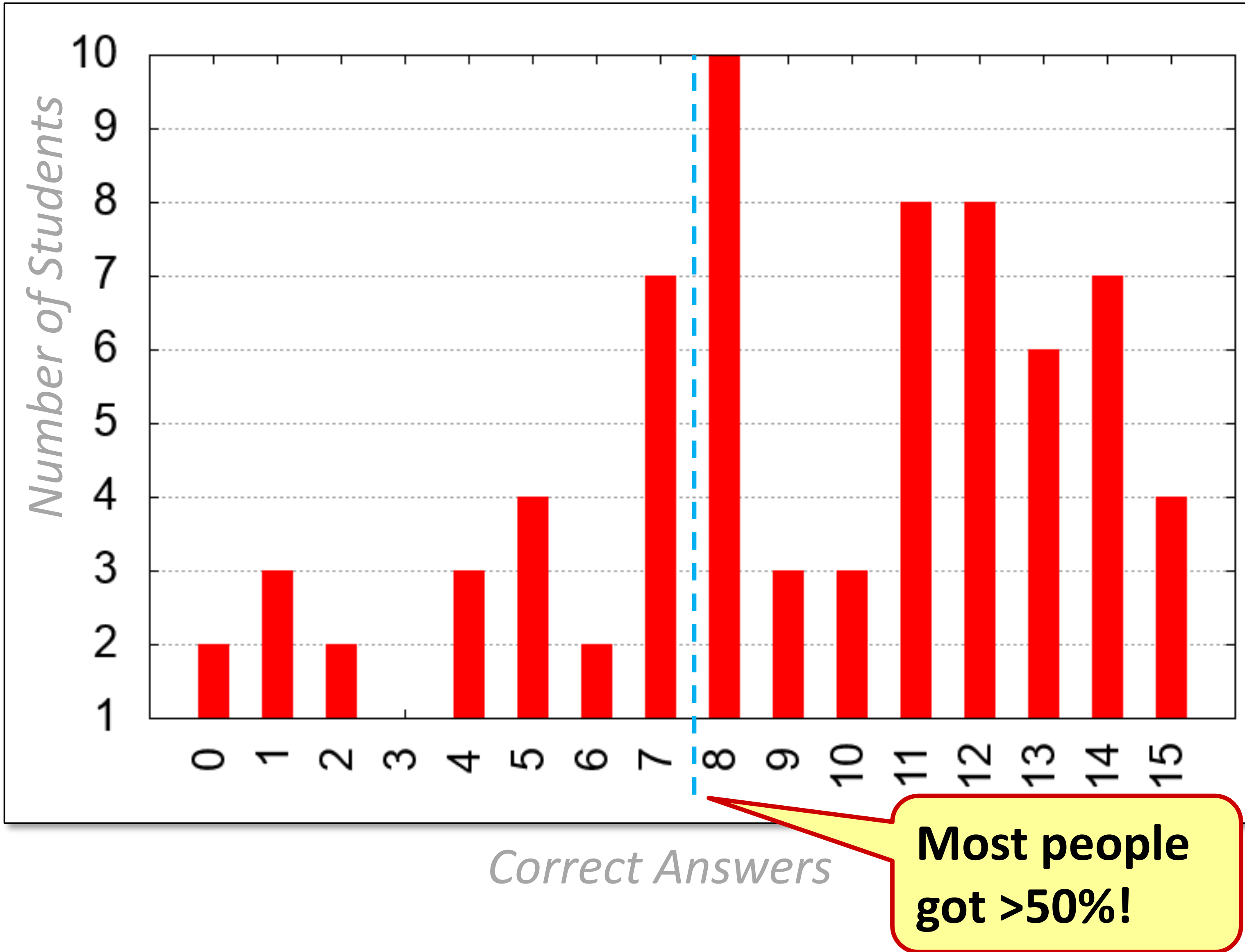
- Exam & Grade Structure
- Demo Test review
- Course Review

Demo Test: participation

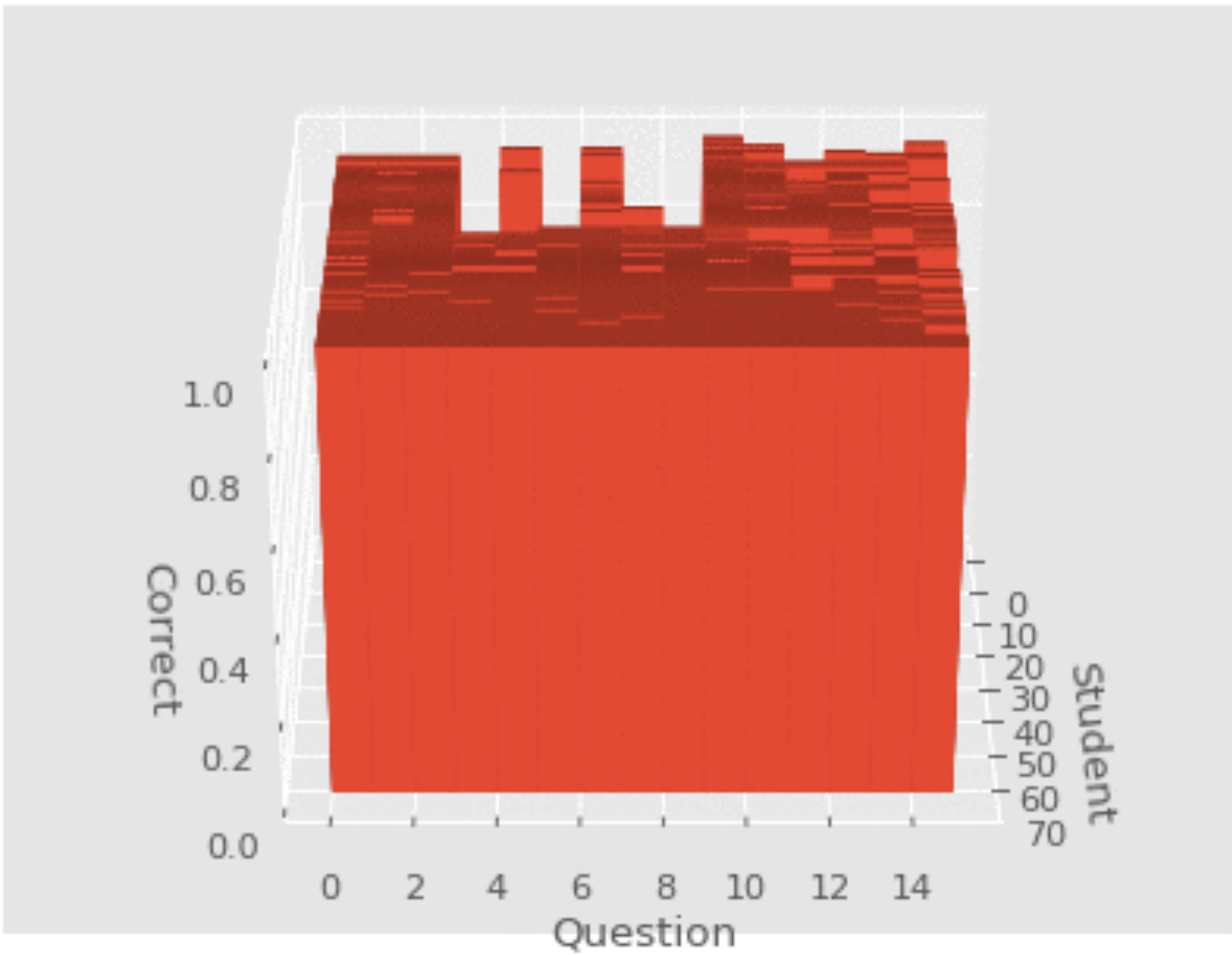
- **Establishes significance of analysis on next slides.**
  We'll analyse how the population performed in each question.



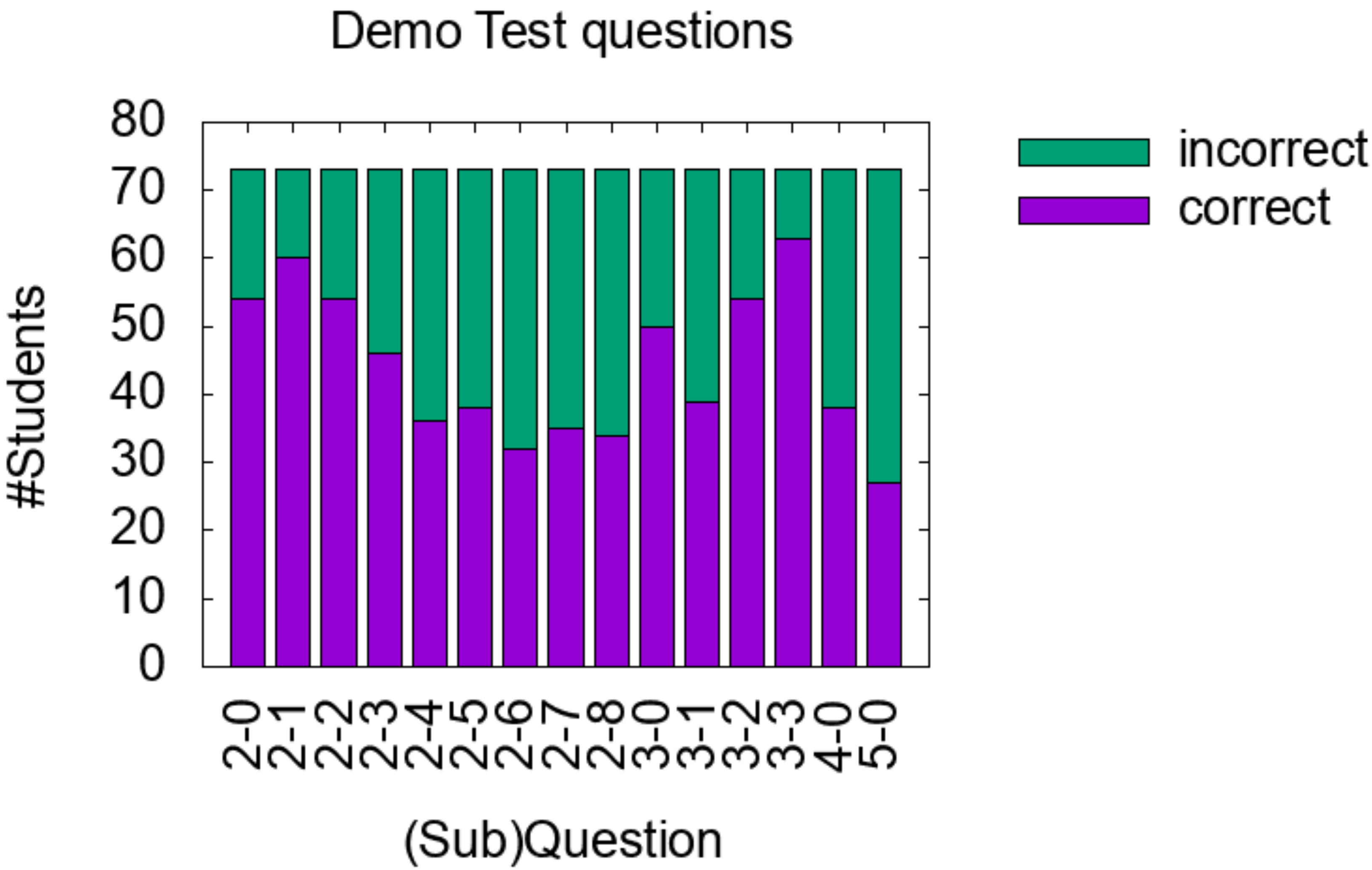
Demo Test: test-level histogram



Demo Test: p/question & p/student results



Demo Test: per-question results



Demo Test: Q2

- **For more: Read Chapter 3 of CS:APP3e and practice problems**

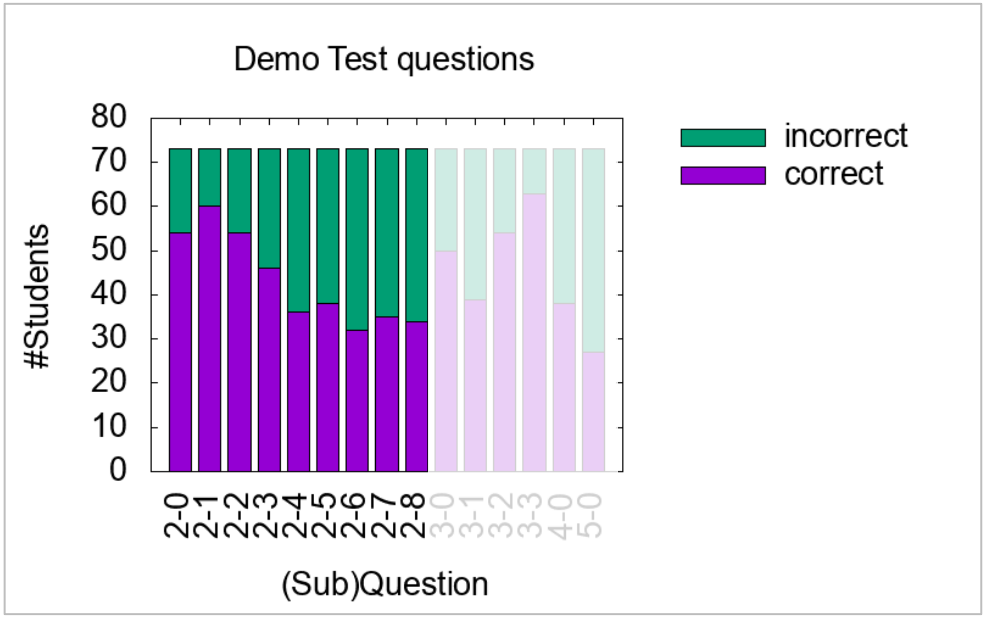
Assume the following values are stored at the indicated memory addresses and registers:

Address	Value
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

Register	Value
%rax	0x100
%rcx	0x1
%rdx	0x3

Provide values for operands indicated in the following table:

Operand	Value
%rax	[1]
0x104	[2]
0x108	[3]
(%rax)	[4]
4(%rax)	[5]
9(%rax, %rdx)	[6]
260(%rcx, %rdx)	[7]
0xFC(, %rcx, 4)	[8]
(%rax, %rdx ,4)	[9]



From Day 6

Address Computation Examples

%rdx	0xf000
%rcx	0x0100

Expression	Address Computation	Address
0x8(%rdx)	0xf000 + 0x8	0xf008
(%rdx,%rcx)	0xf000 + 0x100	0xf100
(%rdx,%rcx,4)	0xf000 + 4*0x100	0xf400
0x80(,%rdx,2)	2*0xf000 + 0x80	0x1e080

From Day 6

Complete Memory Addressing Modes

- **Most General Form**

  D(Rb,Ri,S) Mem[Reg[Rb]+S*Reg[Ri]+ D]

  - D: Constant "displacement" 1, 2, or 4 bytes
  - Rb: Base register: Any of 16 integer registers
  - Ri: Index register: Any, except for `%rsp`
  - S: Scale: 1, 2, 4, or 8 (*why these numbers?*)

- **Special Cases**

  (Rb,Ri) Mem[Reg[Rb]+Reg[Ri]]

  D(Rb,Ri) Mem[Reg[Rb]+Reg[Ri]+D]

  (Rb,Ri,S) Mem[Reg[Rb]+S*Reg[Ri]]

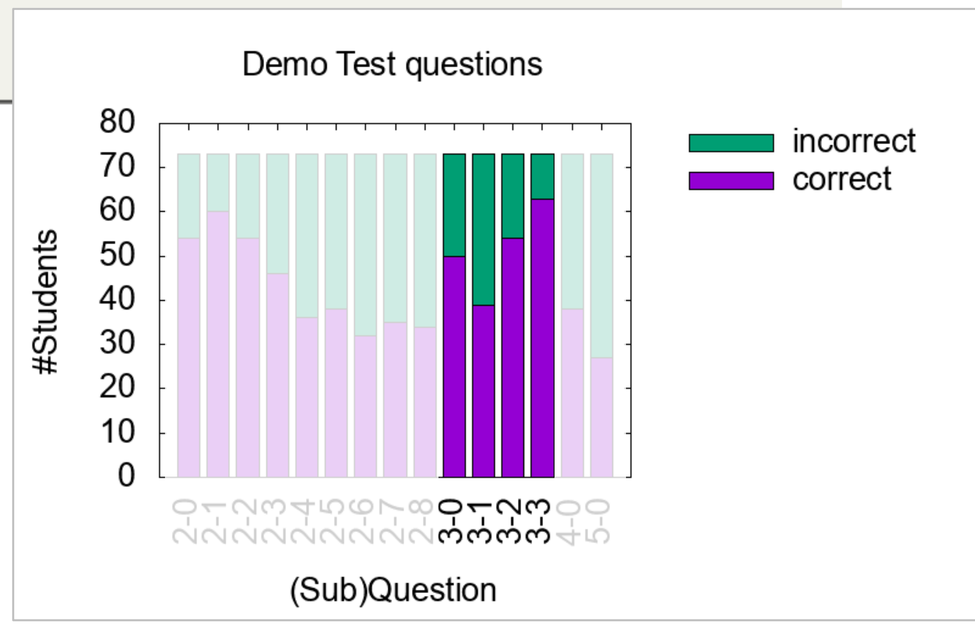
Demo Test: Q3

- **For more: Read Chapter 2 of CS:APP3e and practice problems**

Complete this program to make it well-typed:

```
[A] f(int x, [B] [C]) {
  for (int [D] = 0; i < x; i++) {
    j = (long)(i + x);
  }

  return (char)j;
}
```

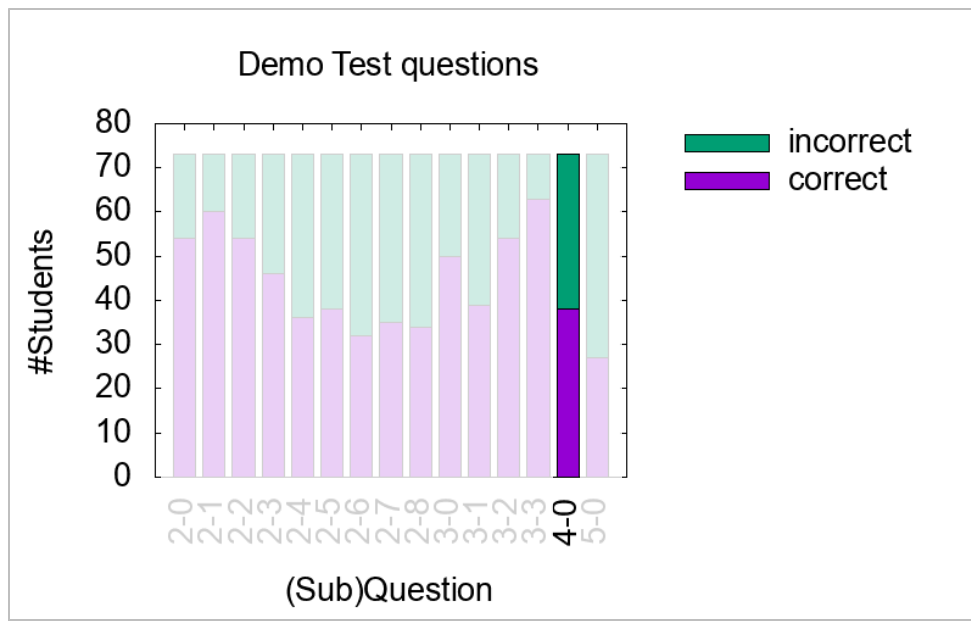


Demo Test: Q4

- **For more: Read Chapter 3 of CS:APP3e and Chap. 5 of K&R.**

Which best describes the type of p, declared below?

```
char (*p[10])(int *);
```



From Day 10

Pointers in C

- We encountered pointers several times so far.
  As with any language: **practice makes perfect!**
- K&R Chapter 5 (can get from library – see announcement on Blackboard and at last lecture).

2. Consider the following C declaration:

```
int iarr[100];
void *p = iarr;
```

Which of the following expressions is semantically equivalent to "`iarr[50]`"?

(a) `*(int *)((char *)p + 50 * sizeof(int))`

(b) `*(int *)(p + 50 * sizeof(int *))`

(c) `((int *)((char *)p + 50))[0]`

(d) `*(char *)((int *)p + 50)`

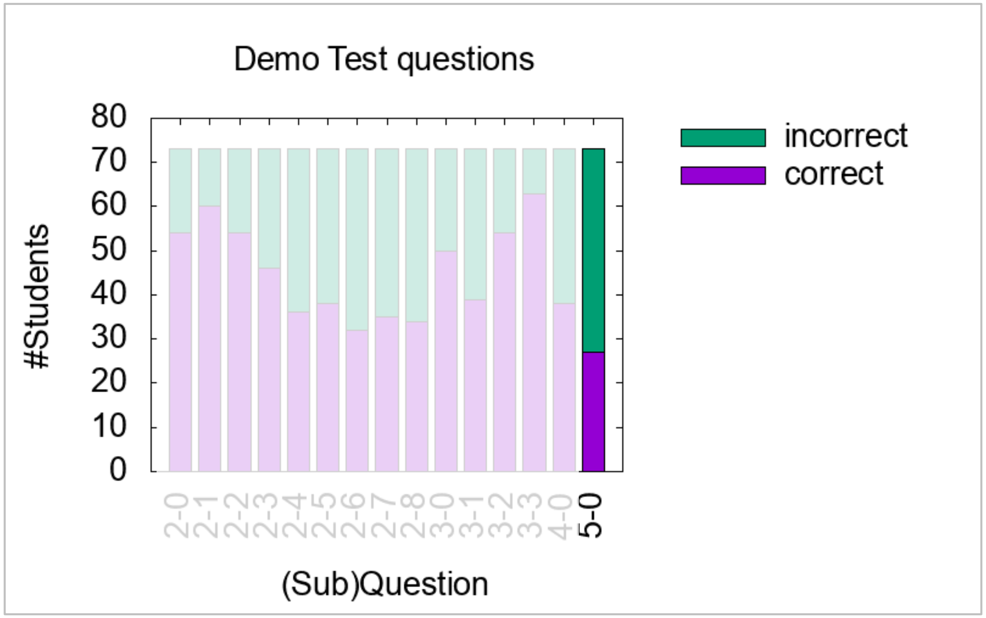
- See past exam questions: http://www.cs.iit.edu/~nsultana1/teaching/F22CS351/otherresources.html

Demo Test: Q5

- **For more: Read Chapter 3 of CS:APP3e and practice problems**

What is wrong with the following structure declaration?

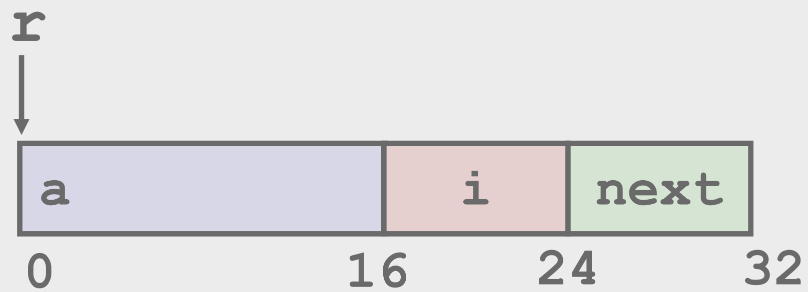
```
struct foo {
  void *val;
  struct foo *p, *q;
  struct foo x, y;
};
```



From Day 9

Structure Representation

```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



- Structure represented as block of memory
  - Big enough to hold all of the fields
- Fields ordered according to declaration
  - Even if another ordering could yield a more compact representation
- Compiler determines overall size + positions of fields
  - Machine-level program has no understanding of the structures in the source code

Today

- Exam & Grade Structure
- Demo Test review
- **Course review**

What did we cover so far?

- Representing data
- Representing programs
- Linking
- Memory
- (+ C and x86_64 toolchains + C review)

Representing data

- **Numeral encoding** (Theory and Practice)
  - Scope: no theorems or proofs since this isn't a maths course (but helps to understand them)
  - Scope does include two's complement arithmetic
  - Encoding of integers (signed & unsigned) in C, and max and min values.
  - Conversions/casts between both
- **Encoding other types** (wrt Machine Programming)
  - Arrays, Structs, Unions
  - Alignment

From Day 3

Two-complement Encoding Example (Cont.)

```
x =      15213: 00111011 01101101
y =     -15213: 11000100 10010011
```

Weight	15213		-15213	
1	1	1	1	1
2	0	0	1	2
4	1	4	0	0
8	1	8	0	0
16	0	0	1	16
32	1	32	0	0
64	1	64	0	0
128	0	0	1	128
256	1	256	0	0
512	1	512	0	0
1024	0	0	1	1024
2048	1	2048	0	0
4096	1	4096	0	0
8192	1	8192	0	0
16384	0	0	1	16384
-32768	0	0	1	-32768
Sum		**15213**		**-15213**

From Day 3

Signed vs. Unsigned in C

- **Constants**
  - By default are considered to be signed integers
  - Unsigned if have “U” as suffix

```
0U, 4294967259U
```

- **Casting**
  - Explicit casting between signed & unsigned same as U2T and T2U

```
int tx, ty;
unsigned ux, uy;
tx = (int) ux;
uy = (unsigned) ty;
```

  - Implicit casting also occurs via assignments and procedure calls

```
tx = ux;
uy = ty;
```

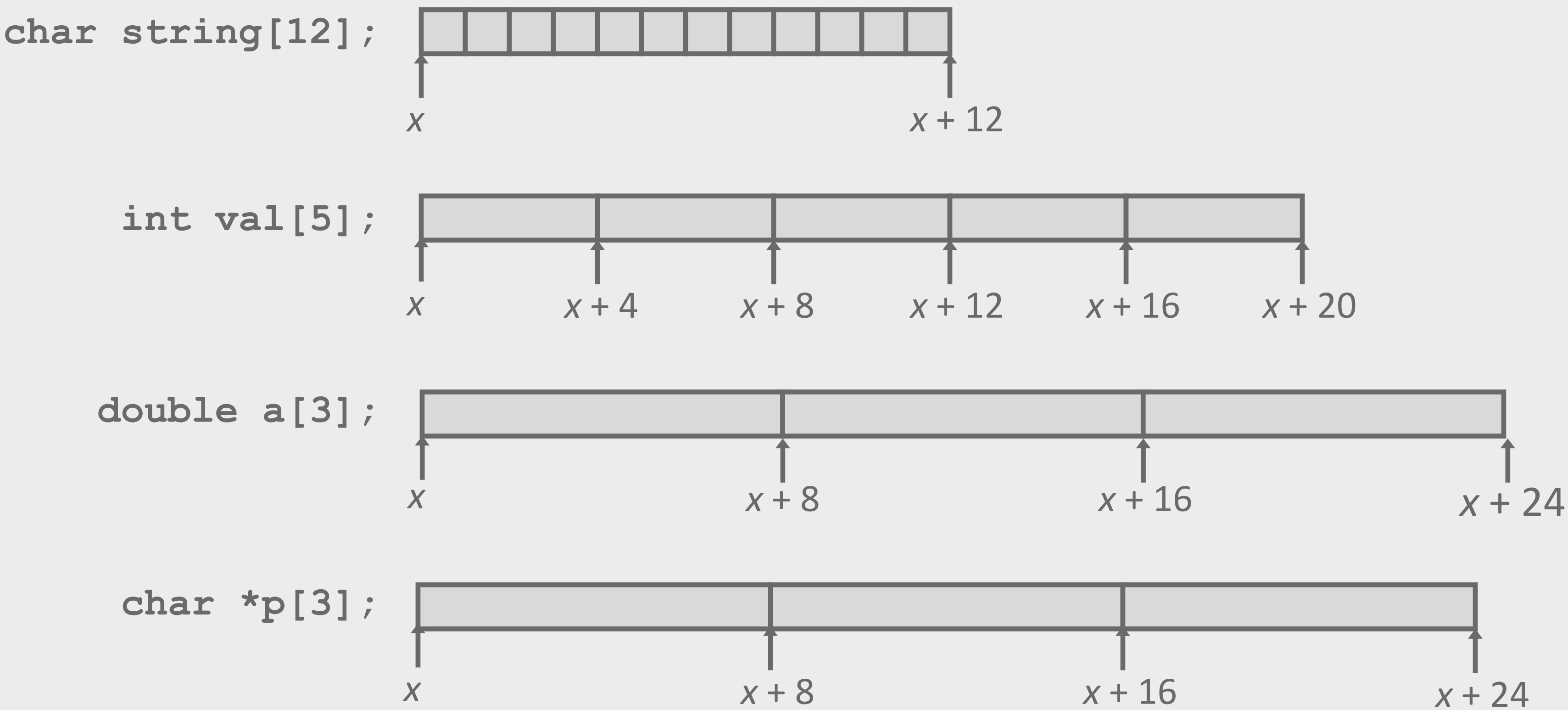
From Day 9

Array Allocation

- **Basic Principle**

  T **A[*L*]**;

  - Array of data type *T* and length *L*
  - Contiguously allocated region of *L* * **sizeof**(*T*) bytes in memory



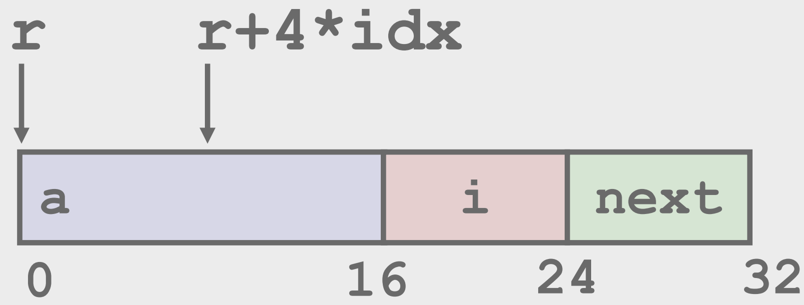
Representing programs

- **Interacting with data**
  (Overlap with previous topic)
  - Arrays, Structs, Unions
  - Alignment
- **Control flow**
  - Branching
  - Procedure calls
  - Loops

From Day 9

Generating Pointer to Structure Member

```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



- **Generating Pointer to Array Element**
  - Offset of each structure member determined at compile time
  - Compute as `r + 4*idx`

```
int *get_ap
 (struct rec *r, size_t idx)
{
  return &r->a[idx];
}
```

```
  # r in %rdi, idx in %rsi
  leaq  (%rdi,%rsi,4), %rax
  ret
```

From Day 9

Alignment Principles

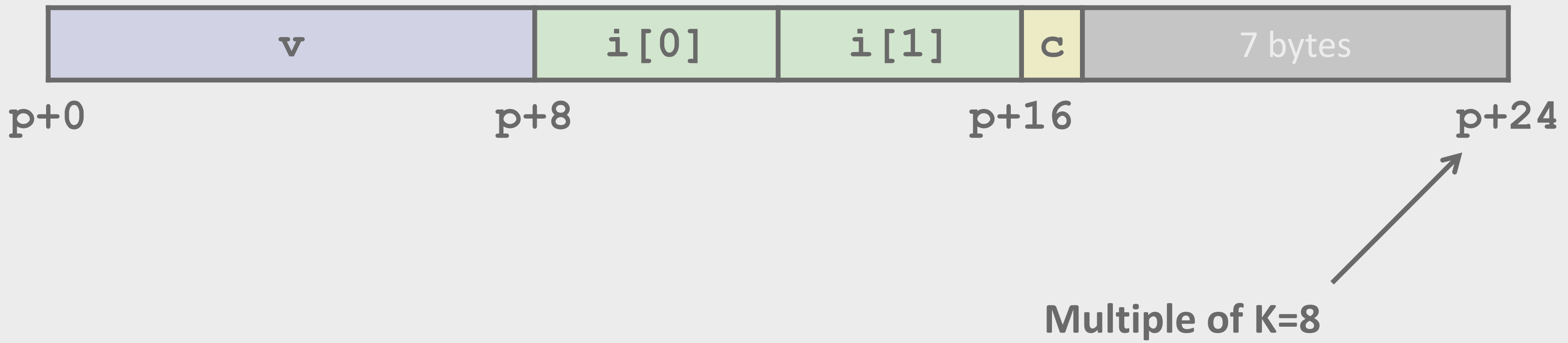
- **Aligned Data**
  - Primitive data type requires K bytes
  - Address must be multiple of K
  - Required on some machines; advised on x86-64
- **Motivation for Aligning Data**
  - Memory accessed by (aligned) chunks of 4 or 8 bytes (system dependent)
    - Inefficient to load or store datum that spans quad word boundaries
    - Virtual memory trickier when datum spans 2 pages
- **Compiler**
  - Inserts gaps in structure to ensure correct alignment of fields

From Day 9

Meeting Overall Alignment Requirement

- For largest alignment requirement K
- Overall structure must be multiple of K

```
struct S2 {
  double v;
  int i[2];
  char c;
} *p;
```

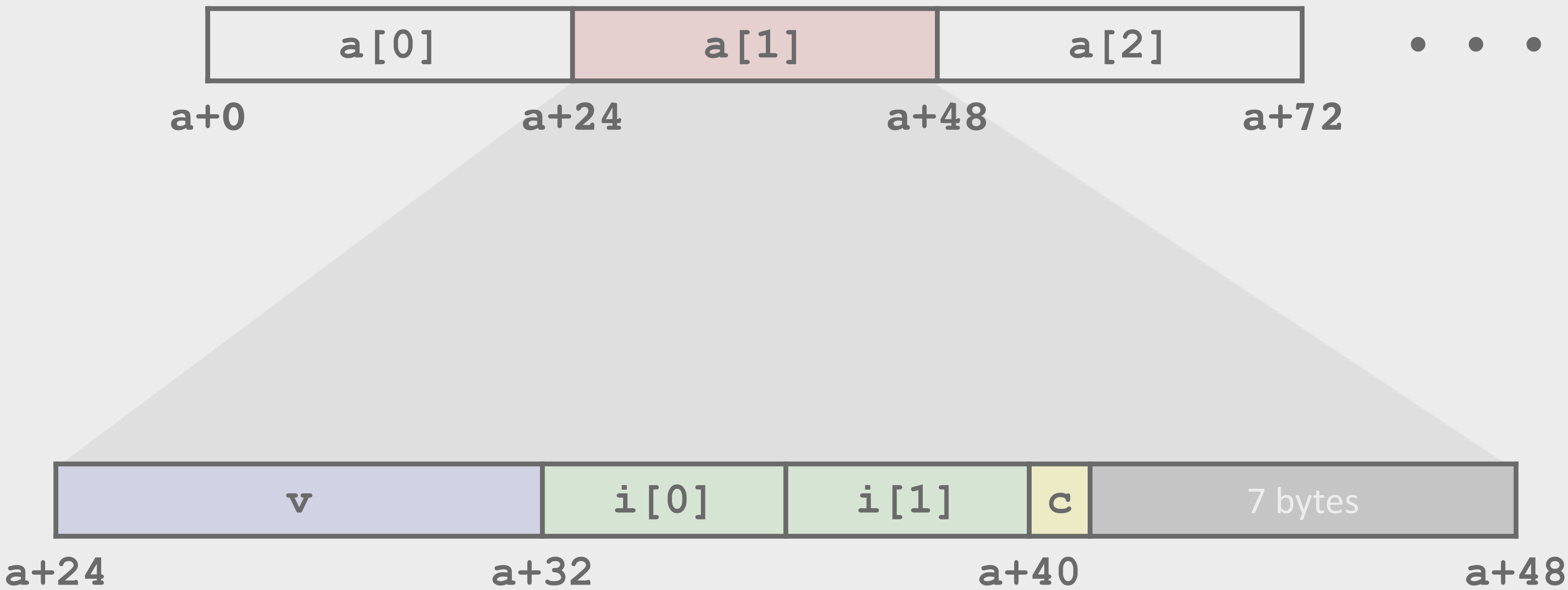


From Day 9

Arrays of Structures

- Overall structure length multiple of K
- Satisfy alignment requirement for every element

```
struct S2 {
  double v;
  int i[2];
  char c;
} a[10];
```



From Day 10

Conditional Branch Example

- Generation

```
unix> gcc –Og -S –fno-if-conversion control.c
```

```
long absdiff
  (long x, long y)
{
  long result;
  if (x > y)
    result = x-y;
  else
    result = y-x;
  return result;
}
```

```
absdiff:
   cmpq    %rsi, %rdi  # x:y
   jle     .L4
   movq    %rdi, %rax
   subq    %rsi, %rax
   ret
.L4:       # x <= y
   movq    %rsi, %rax
   subq    %rdi, %rax
   ret
```

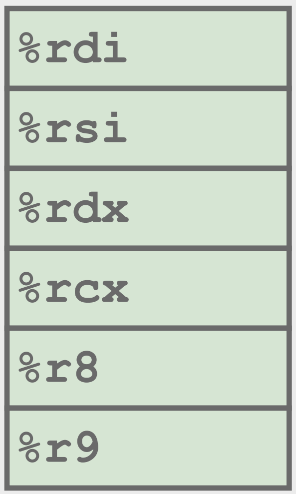
Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rax	Return value

From Day 8

Procedure Data Flow

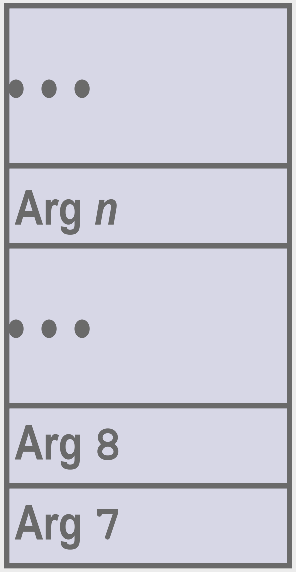
Registers

- First 6 arguments



- Return value

Stack



- Only allocate stack space when needed

Linking

- **Toolchain flow**
- **Resolution**
  - Symbol not found?
  - >1 symbols found?
- **Relocation**
- **Static and Dynamic**

From Day 12

Step 1: Symbol Resolution

Referencing a global…

…that's defined here

```c
int sum(int *a, int n);

int array[2] = {1, 2};

int main()
{
    int val = sum(array, 2);
    return val;
}
```

main.c

```c
int sum(int *a, int n)
{
    int i, s = 0;

    for (i = 0; i < n; i++) {
        s += a[i];
    }
    return s;
}
```

sum.c

Defining a global

Linker knows nothing of `val`

Referencing a global…

…that's defined here

Linker knows nothing of `i` or `s`

From Day 12

Linker Symbols

- **Global symbols**
  - Symbols defined by module *m* that can be referenced by other modules.
  - E.g.: non-`static` C functions and non-`static` global variables.

- **External symbols**
  - Global symbols that are referenced by module *m* but defined by some other module.

- **Local symbols**
  - Symbols that are defined and referenced exclusively by module *m*.
  - E.g.: C functions and global variables defined with the `static` attribute.
  - **Local linker symbols are *not* local program variables**

From Day 12

Global Variables

- **Avoid if you can**

- **Otherwise**
  - Use `static` if you can
  - Initialize if you define a global variable
  - Use `extern` if you reference an external global variable

Memory

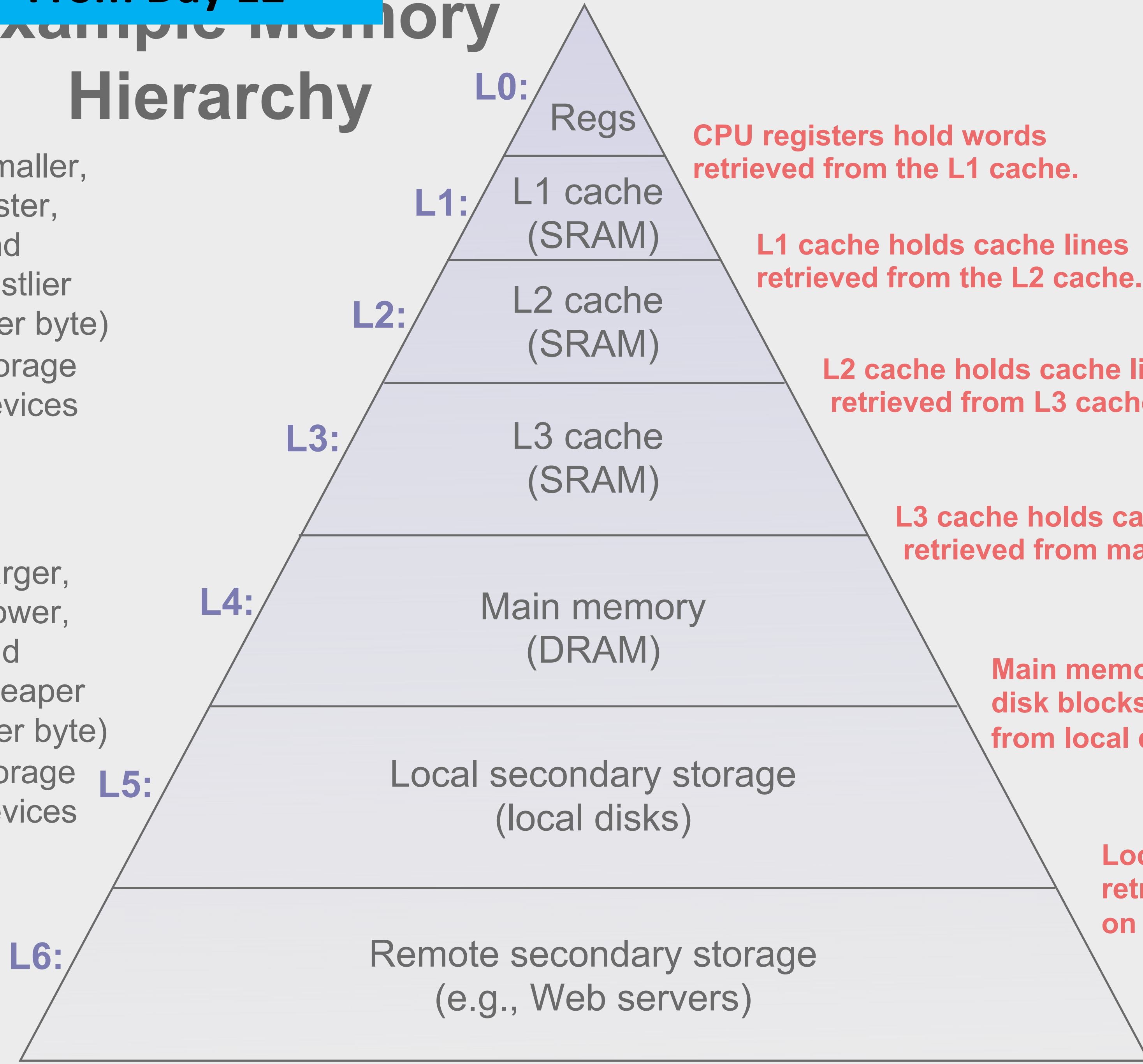
- Memory hierarchy
- Memory mountain: throughput vs stride vs size
- Cache structure and look-up

From Day 12

Example Memory Hierarchy

Smaller, faster, and costlier (per byte) storage devices

Larger, slower, and cheaper (per byte) storage devices



CPU registers hold words retrieved from the L1 cache.

L1 cache holds cache lines retrieved from the L2 cache.

L2 cache holds cache lines retrieved from L3 cache

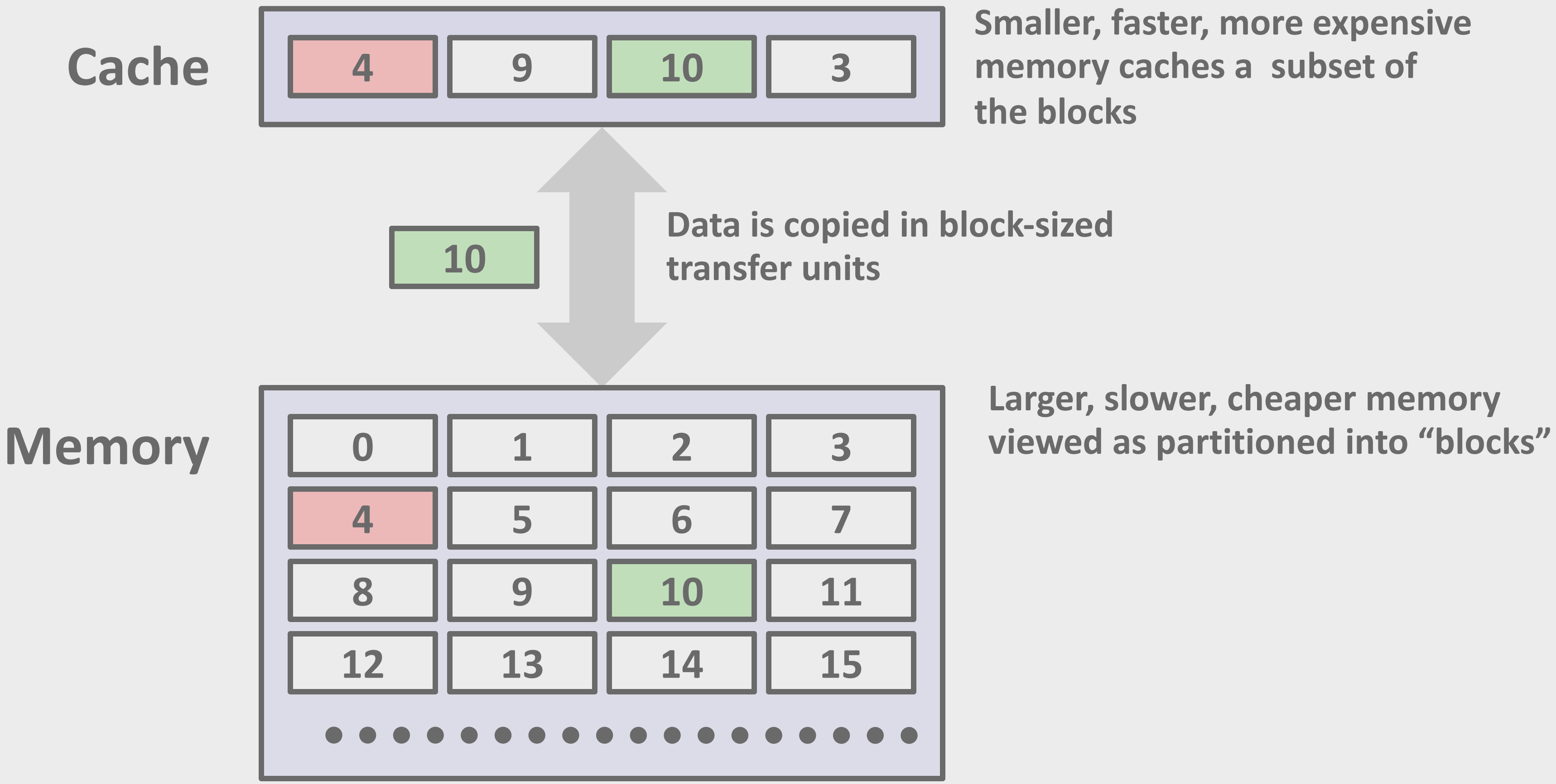
L3 cache holds cache lines retrieved from main memory.

Main memory holds disk blocks retrieved from local disks.

Local disks hold files retrieved from disks on remote servers

From Day 12

General Cache Concepts

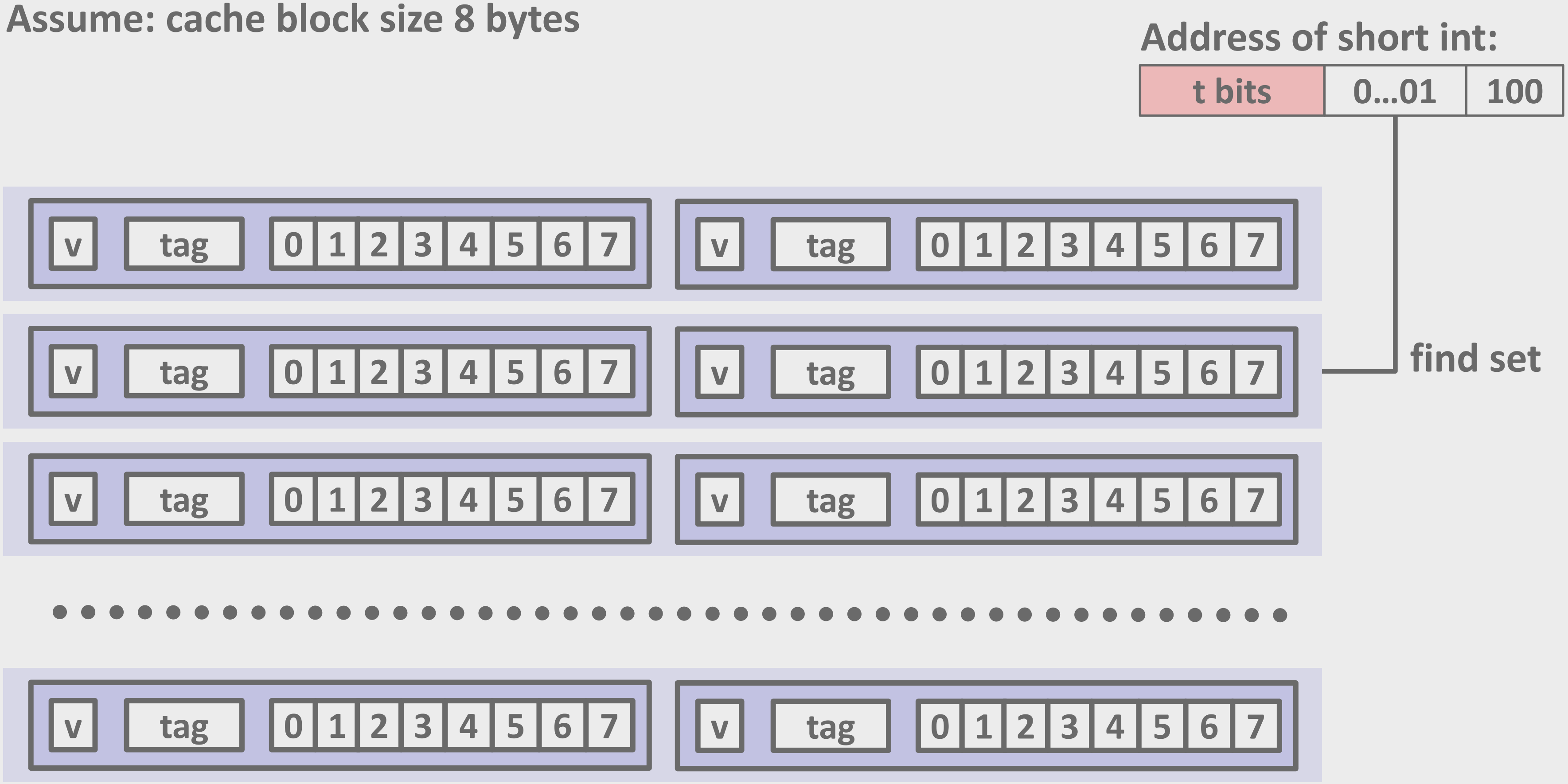


From Day 13

E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set

Assume: cache block size 8 bytes



Next week: recorded lectures

Oct 11 **LEC 15:** Exam review **Preparation:** Re[illegible] CS:APP 1-3,6,7 Come prep[illegible] questions.	Oct 12	Oct 13 **Mid-term Exam** **Scope:** Lectures 1-15.
Oct 18 **LEC 16:** ECF: Exceptions & Processes **Preparation:** Read CS:APP 8.1-8.4	Oct 19	Oct 20 **LEC 17:** ECF: Signals **Preparation:** Read CS:APP 8.5-8.8

- LEC 16 **and** LEC 17 **will be pre-recorded and circulated on Blackboard.**
  - Do not come to SB104 those days **– there will not be an in-person lecture.**
  - **My away-at-a-conference days are marked on the course calendar.**

Questions?

Per-lecture feedback

- Better sooner rather than later!
- I can help with issues sooner.
- There is a per-lecture feedback form.
- **The form is anonymous.**
  (It checks that you're at Illinois Tech to filter abuse, but I don't see who submitted any of the forms.)
- https://forms.gle/qoeEbBuTYXo5FiU1A
- I'll remind about this at each lecture.