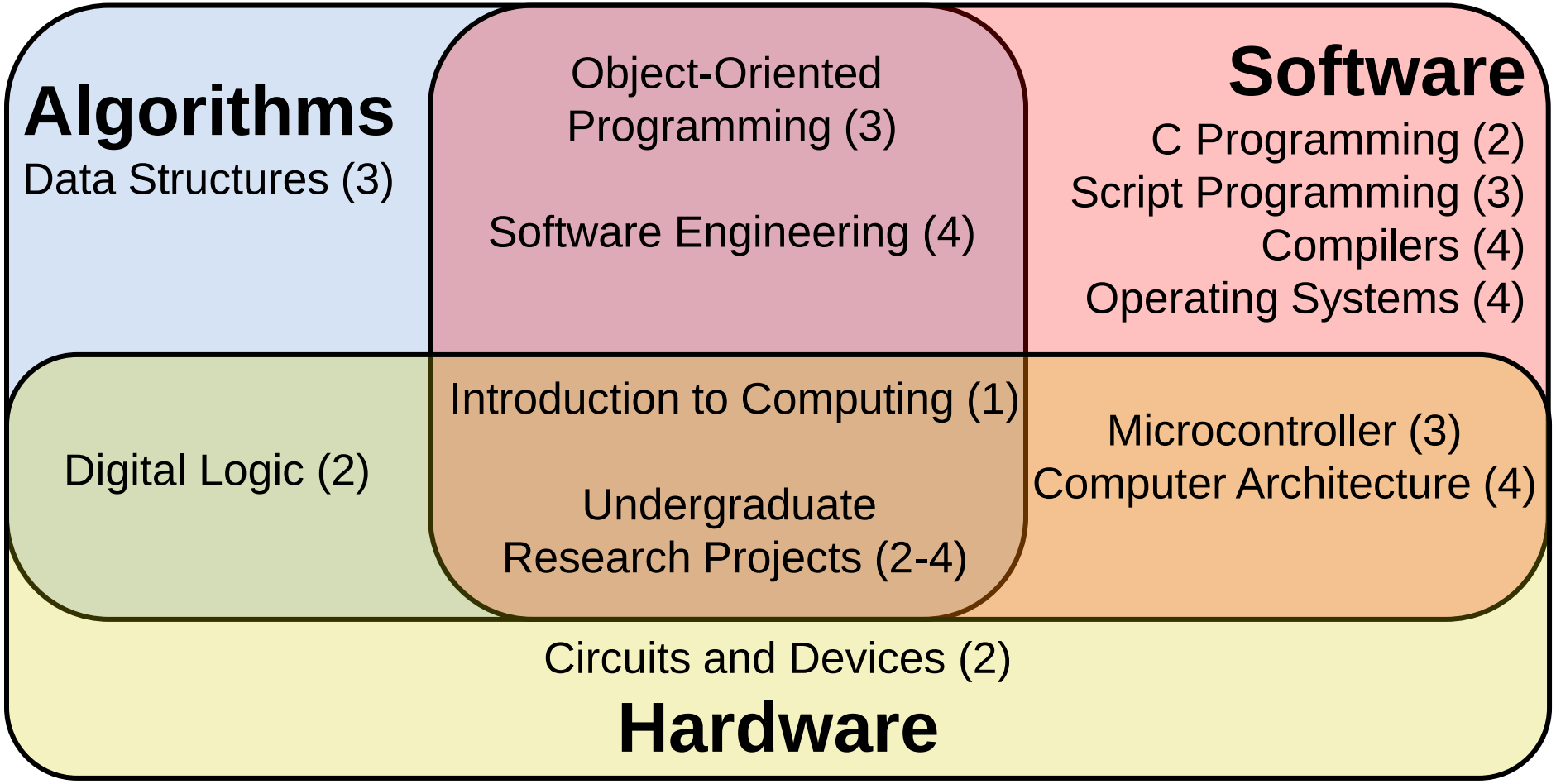
Introducing Parallel Computing in Undergraduate Curriculum

Cordelia M.Brown, Yung-Hsiang Lu, Samuel Midkiff
Electrical and Computer Engineering
Purdue University, West Lafayette

Curriculum Update

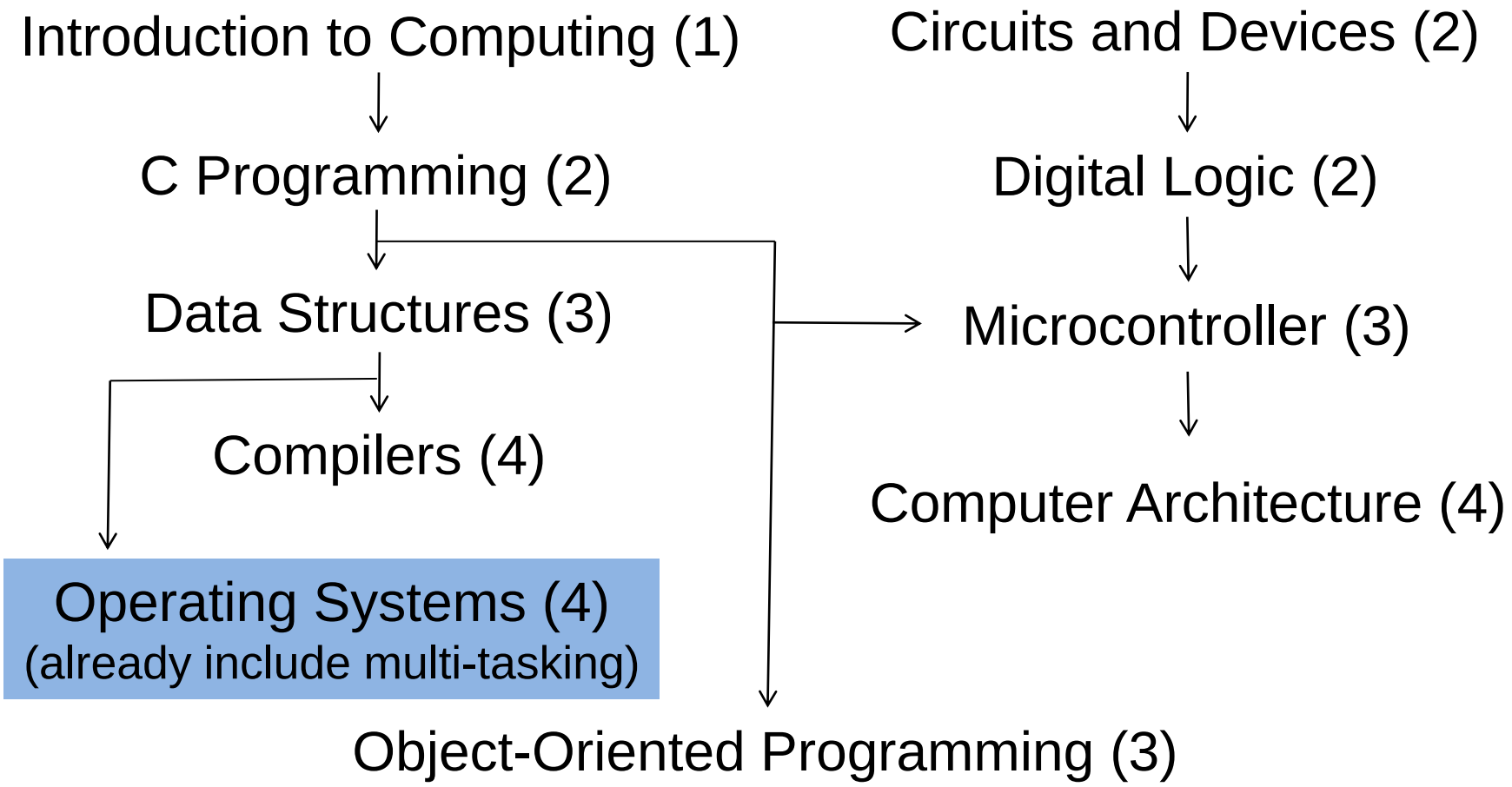
- Goal: Include parallel computing in ***many*** undergraduate courses, not a special new one
- Reason: Students learn different aspects of parallel computing throughout the four years.
- Steps:
  - Identify which courses to change
  - Determine the orders of the changes
  - Eliminate duplicates and unnecessary contents
  - Change the course requirements (ABET)
  - Implement and integrate changes

Identify the Courses to Change



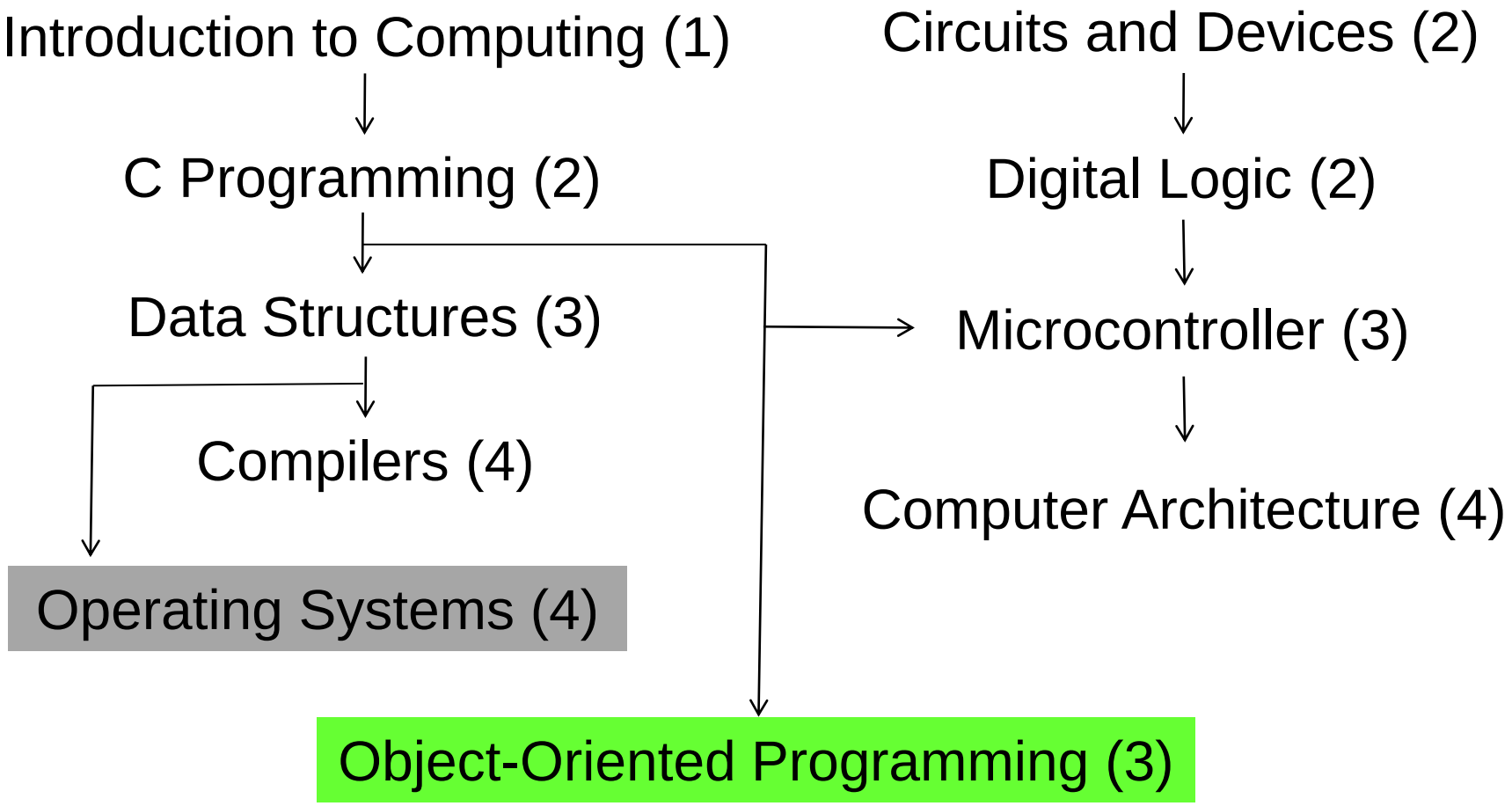
* The numbers mean the years when students take the courses.
Most courses are offered twice a year.

Determine the Order of Changes

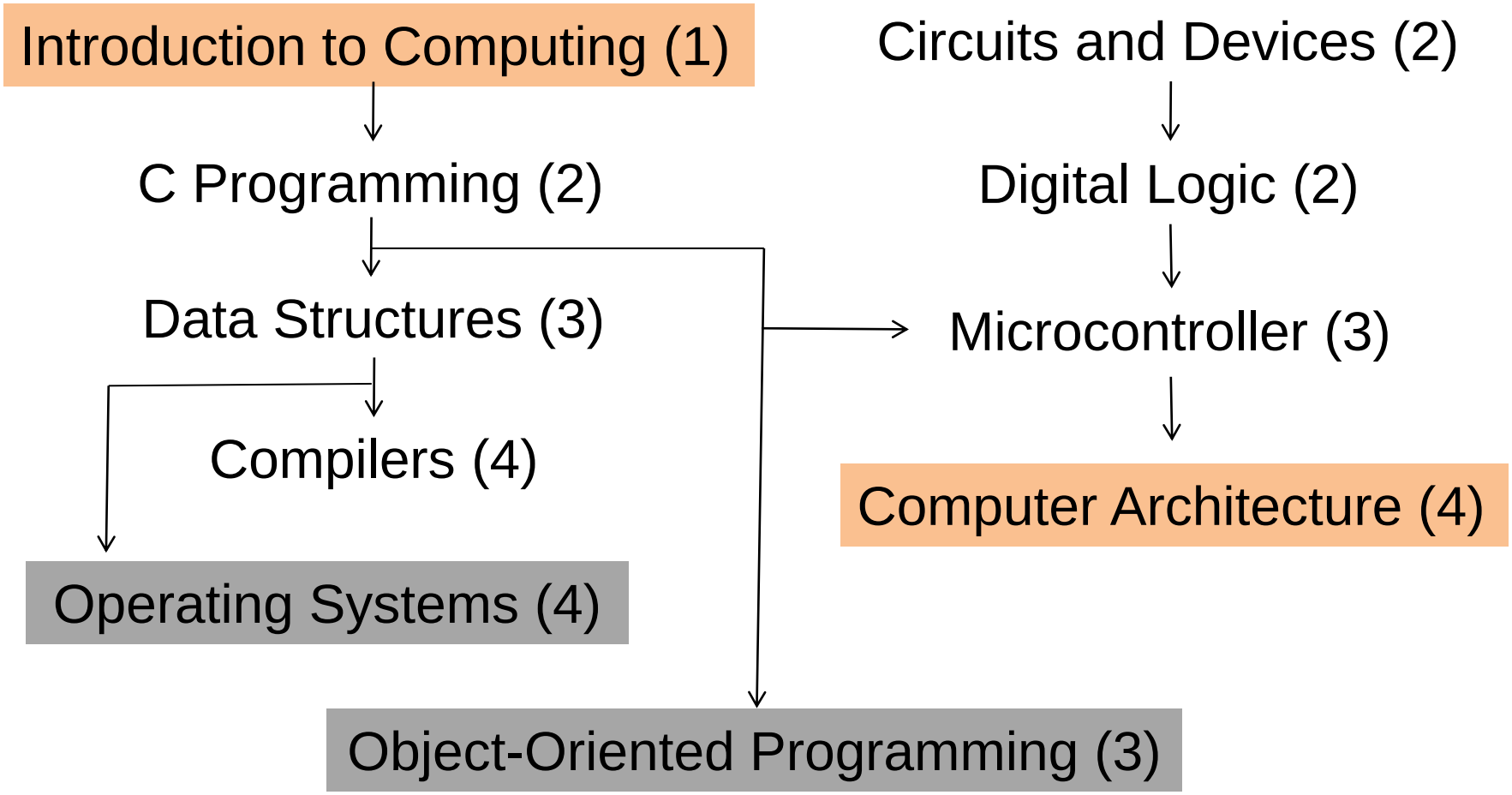


This project was supported in part by NSF CNS 0722212. Any opinions, findings, and conclusions or recommendations expressed in this presentation are those of the authors and do not necessarily reflect the view of the National Science Foundation.

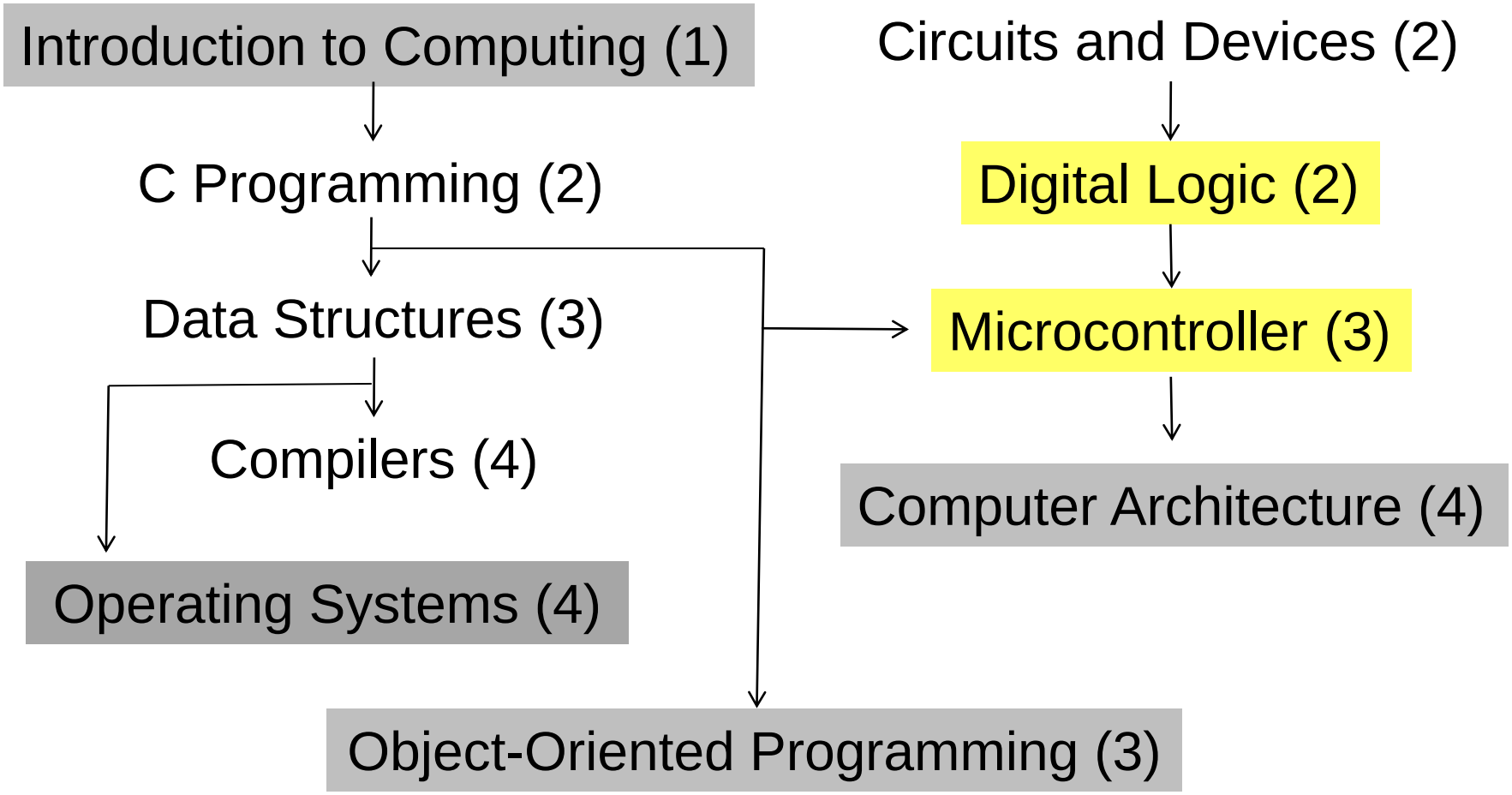
First Change: Elective Course



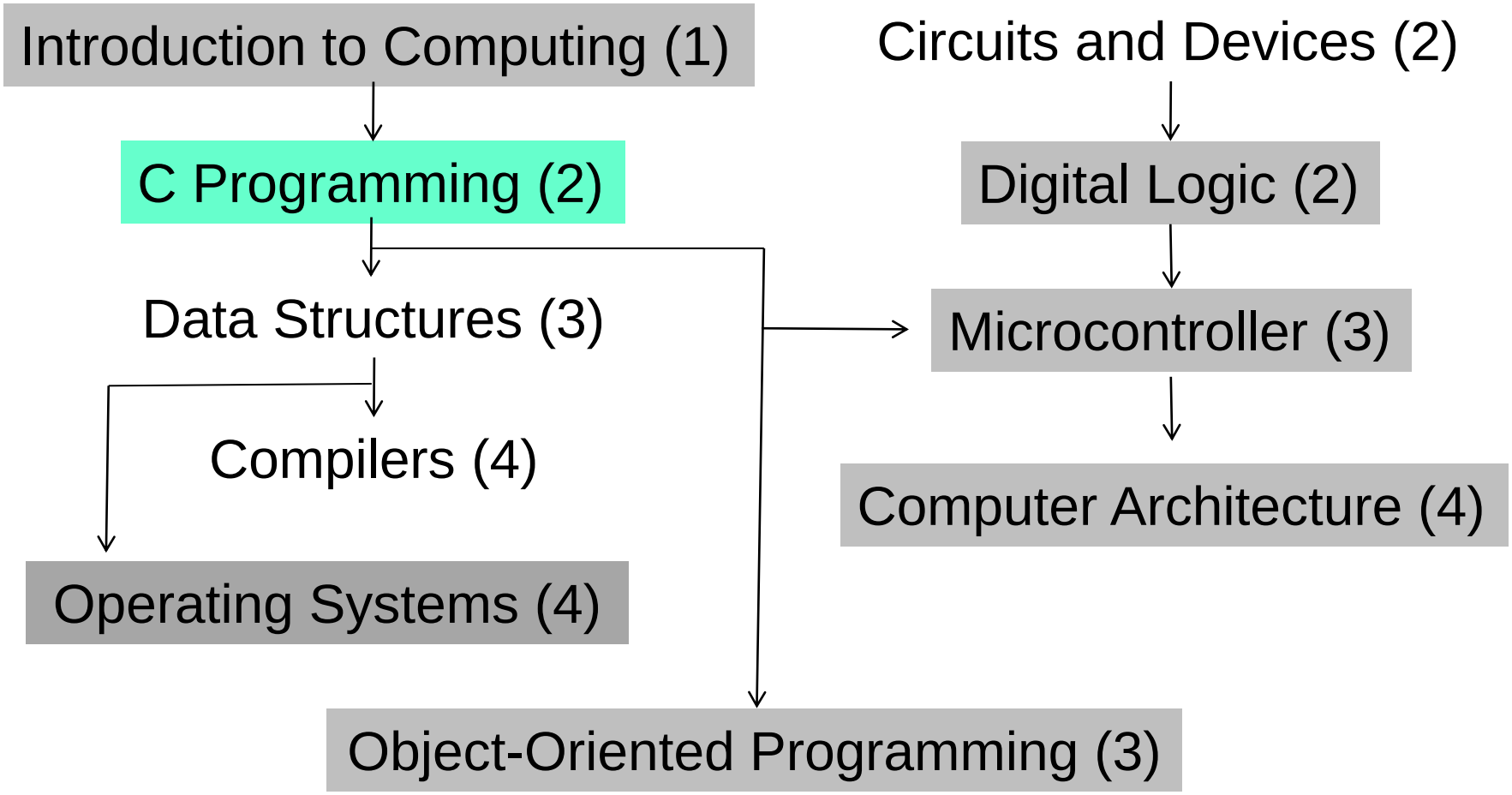
Second Changes from the Ends



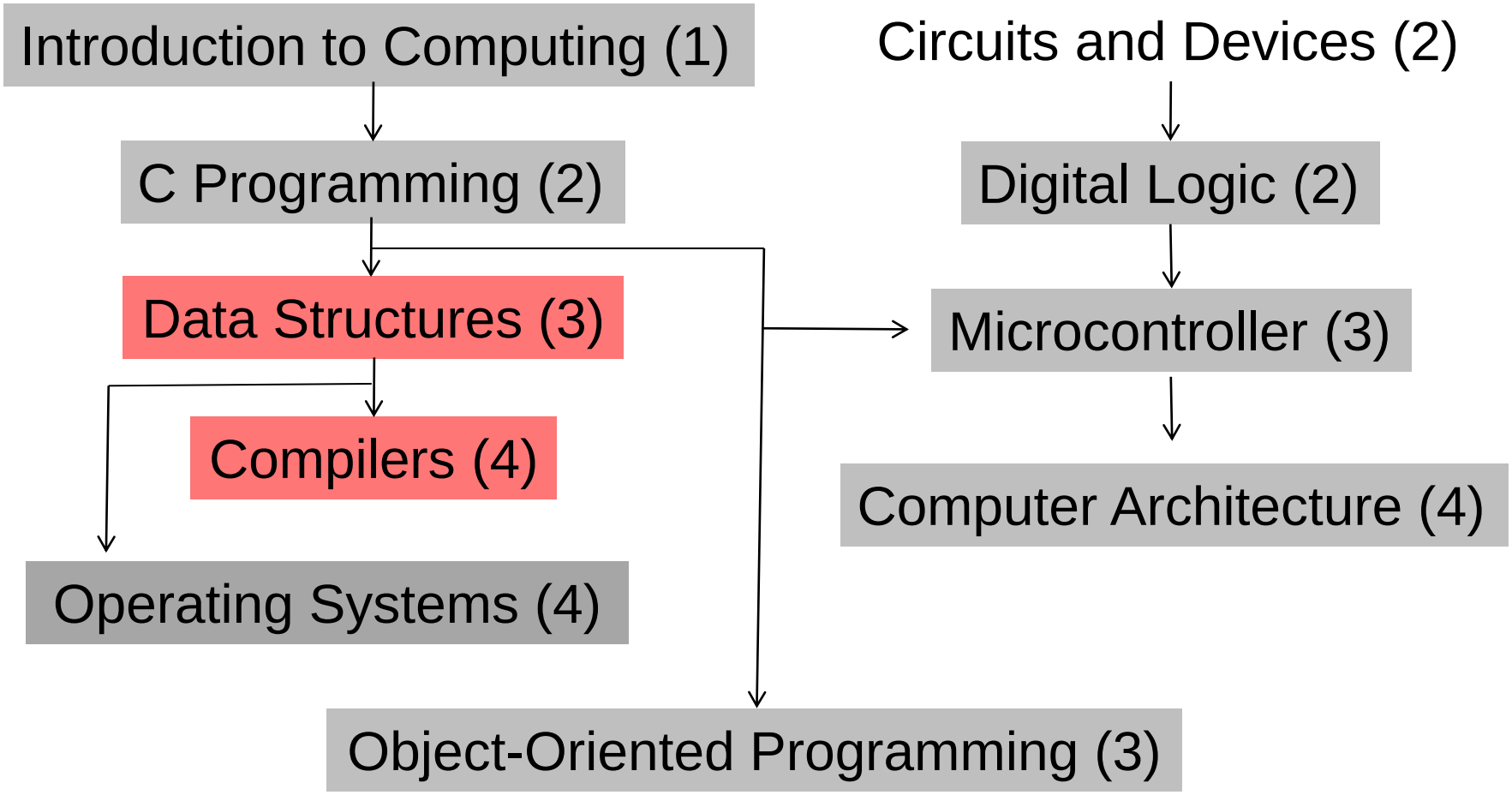
Changes in Intermediate Levels



Latest Change



Not Changed (Yet)



First Change (OOP)

- It is elective and not a prerequisite of any required course.
- Java has built-in support for threads with synchronized methods. C++ can use library (Qt) for threads. GUI uses threads.
- The original course content include duplicate materials that can be eliminated: how to use and ~~how to implement~~ container classes

  $\Rightarrow$ already taught in data structures.

Connect Parallelism with Life

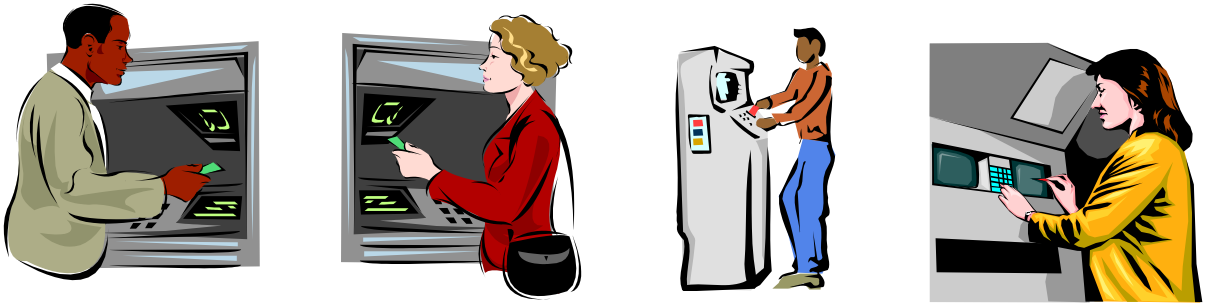
- Use laundry room as examples.
- Many washers + dryers ⇒ hardware resources.
- Many loads of clothes ⇒ data-level parallelism.
- Washing before drying ⇒ dependence and pipeline.

Pipeline in Everyday Life

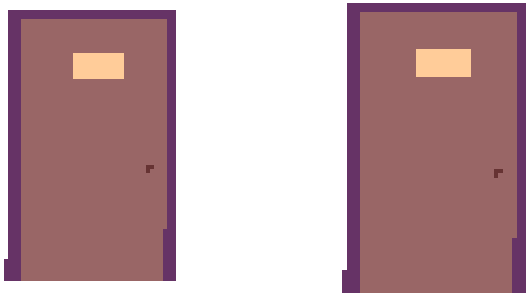
- factory assembly line
- buffet line

Synchronization

- ATM withdrawal to motivate the need of synchronization.



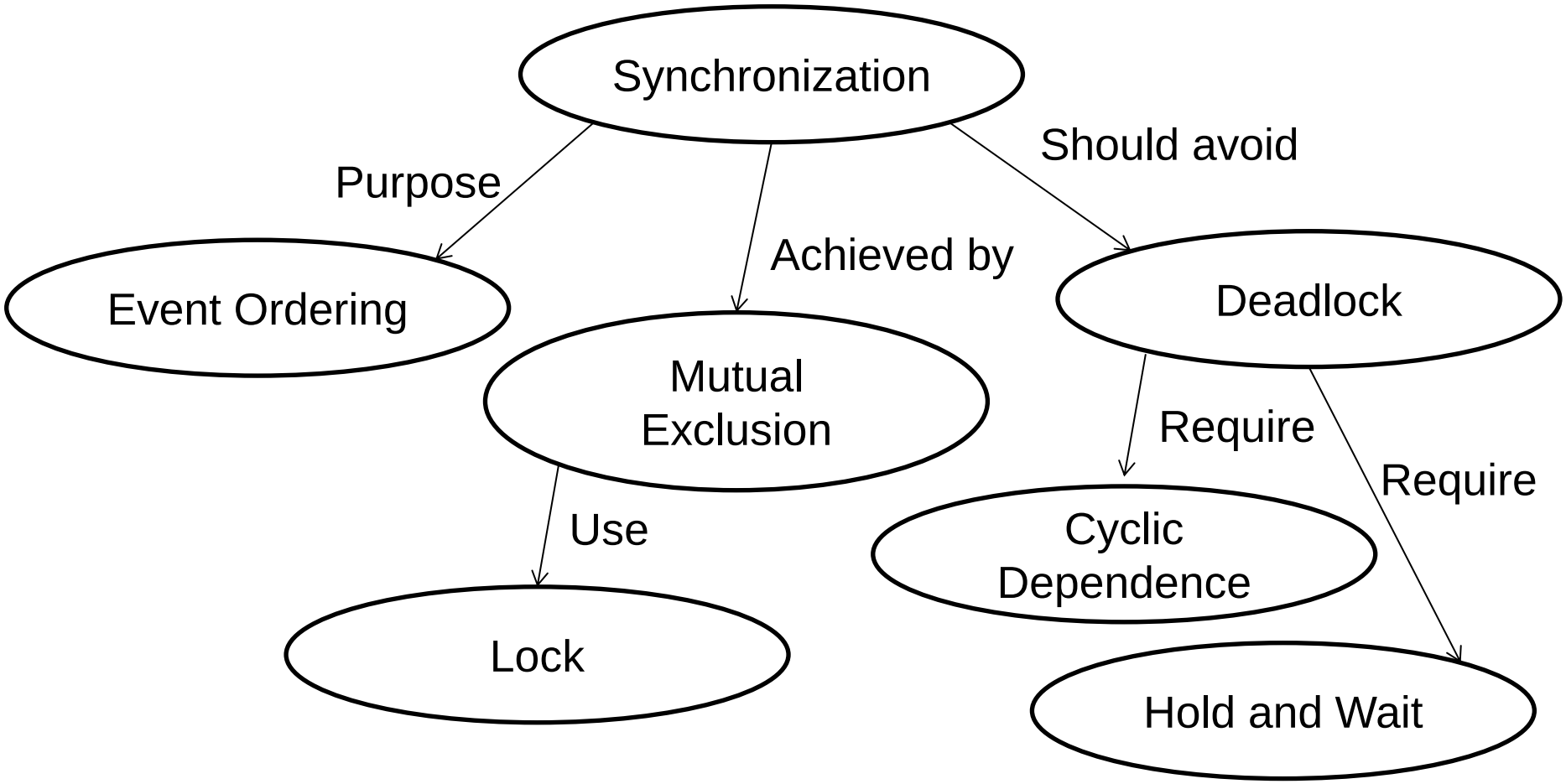
- Library study room with a lock and only one key to explain mutual exclusion.



Concept Inventory

- Purpose: develop a set of questions to evaluate students' understanding of parallel computing ***across their four years of studies.***
- It is a guideline for updating courses and designing assessments for multiple courses.
- Requirements: The questions must be understandable ***without*** using terminology introduced later (e.g. synchronization, mutual exclusion, lock, locality, cache miss ...)
- Approach: use everyday examples to motivate and to describe the problems

Concept Inventory (Excerpt)



The complete concept inventory is in the paper.

Sample Assignments

- Programming assignments
  - Matrix multiplication
  - Image pixel-wise color inversion
  - Network echo server
- Non-programming assignments
  - Amdahl's Law
  - Distinguish SISD/SIMD/MISD/MIMD
  - Conditions and sample code for deadlocks

Most Recent Changes

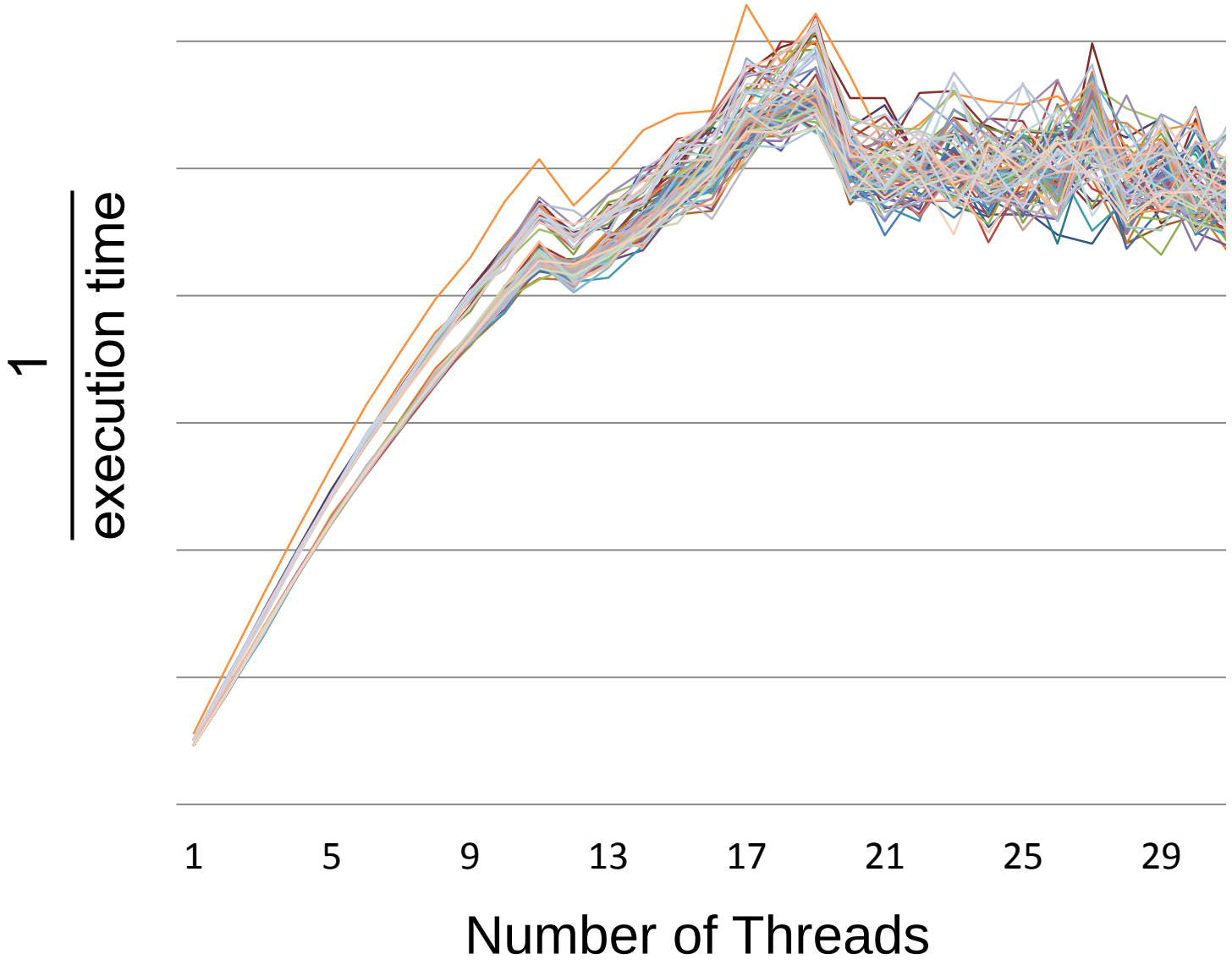
- Second programming class (C)
- 2012 IEEE/TCPP Early Adopter Grant
- two $\Rightarrow$ three credit units since Fall 2012
- For most students, this is the first experience of writing programs with threads
- Programming assignments:
  - Image pixel-wise color inversion
  - Subset sums (count the number of solutions)
- Non-programming assignments: Amdahl's Law and distinguish SISD/SIMD/MISD/MIMD

Evaluation (SIMD, pthreads)

Image color inversion

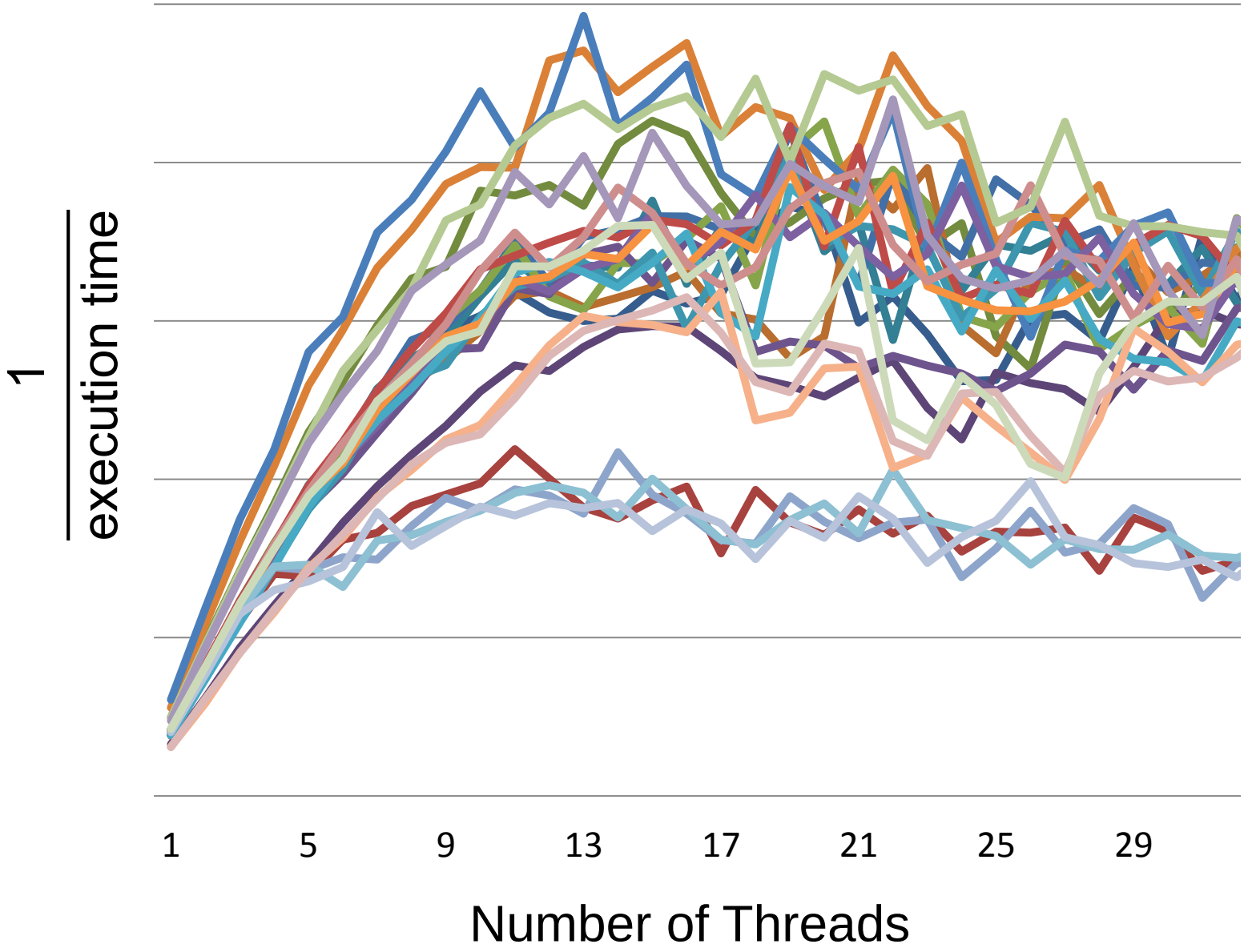
```
for (p = 0; p < numPixels; p ++)
{
    for (c = 0; c < 3; c ++) // RGB 3 colors
        {
            pixels[p].color[c] = 255 -
                    pixels[p].color[c];
        }
}
// parallelization: divide numPixels into
// non-overlapping regions for the threads
```

The time for color inversion
The time for reading and writing files is excluded



Evaluation (SIMD, pthreads)

- Subset sum
- Given a positive integer n and a set of positive integers $S = \{s_1, s_2, ..., s_k\}$
- Find all subsets $A = \{a_1, a_2, ..., a_m\}$ ($A \subseteq S$, $m \leq k$) such that $a_1 + a_2 + ... + a_m = n$
- Count the number of subsets
- Parallelization:
  - Divide the 2^n-1 subsets into regions
  - Each thread checks all subsets in that region
  - If a solution is found, a shared variable numberSolution increments


1
execution time
1
5
9
13
17
21
25
29
Number of Threads

Observations

- Most students understand the concepts and can write correct parallel programs using pthreads.
- Some are not aware of the performance impacts of redundant statements in inner loops.
- Some students know the need of mutual exclusion but each thread has a unique lock.
- Some students put private data (not shared) inside the critical sections.
- Some use expensive operations (for example multiplication or division instead of shifts).

Lessons Learned

- Students are excited learning new concepts related to parallel computing.
- Curriculum update can take several years.
- The changes should be introduced gradually, with the consideration of dependence among courses. The changes should start from a course which has topics that can be eliminated.
- Students should know efficient algorithms are more important than parallelization only.

Lessons Learned

- Assignments should be designed to reduce dependence. For example, many students do not know locality yet $\Rightarrow$ The speedup of matrix multiplication is limited by cache performance
- Some assignments should have high computation and low communication or IO (e.g. subset sum).
- Performance competition can encourage students to pay attention to details.

Conclusion

- We present our experience updating the curriculum including parallel computing in multiple courses throughout the four years.
- We explain the sequence of changes and the rationales of the sequences.
- We describe the concept inventory for cross-cohort evaluations.
- The early-adopter changes provide promising results; most students understand the concepts and can write simple parallel programs.