SE3910 – REAL TIME SYSTEMS

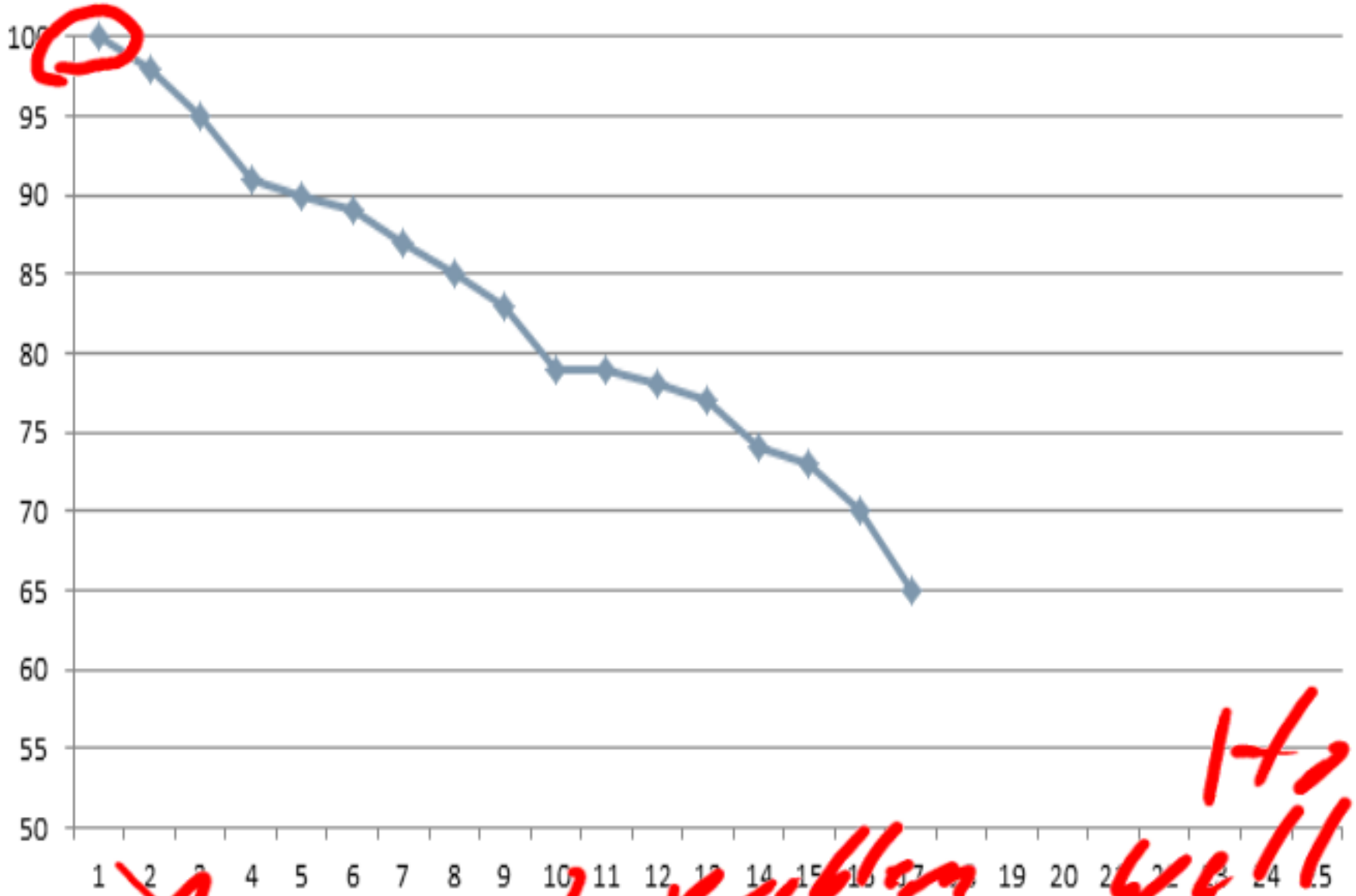
Real Time Systems Programming Languages

Good Morning
Class!
How is everyone today?
⇒ Wishing for more
break.

Good News I
bring.

⇒ Exams are Good
graded!

MIDTERM EXAM



How well covered

Average by problem

83%	Average	83%	90%	89%	79%	100%	87%	92%	81%	66%	90%	90%	93%	81%	96%	86%	96%	90%	83%
83%	Median	85%	88%	100%	75%	100%	100%	100%	83%	100%	90%	90%	100%	80%	100%	80%	100%	90%	80%
10%	STD	11%	12%	22%	16%	0%	28%	19%	19%	43%	10%	10%	10%	12%	9%	12%	9%	10%	13%
100.00%	Max	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
65.00%	Min	65.00%	62.50%	33.33%	50.00%	100.00%	0.00%	40.00%	50.00%	0.00%	80.00%	80.00%	80.00%	60.00%	80.00%	60.00%	80.00%	80.00%	60.00%

MSOE UNIVERSITY

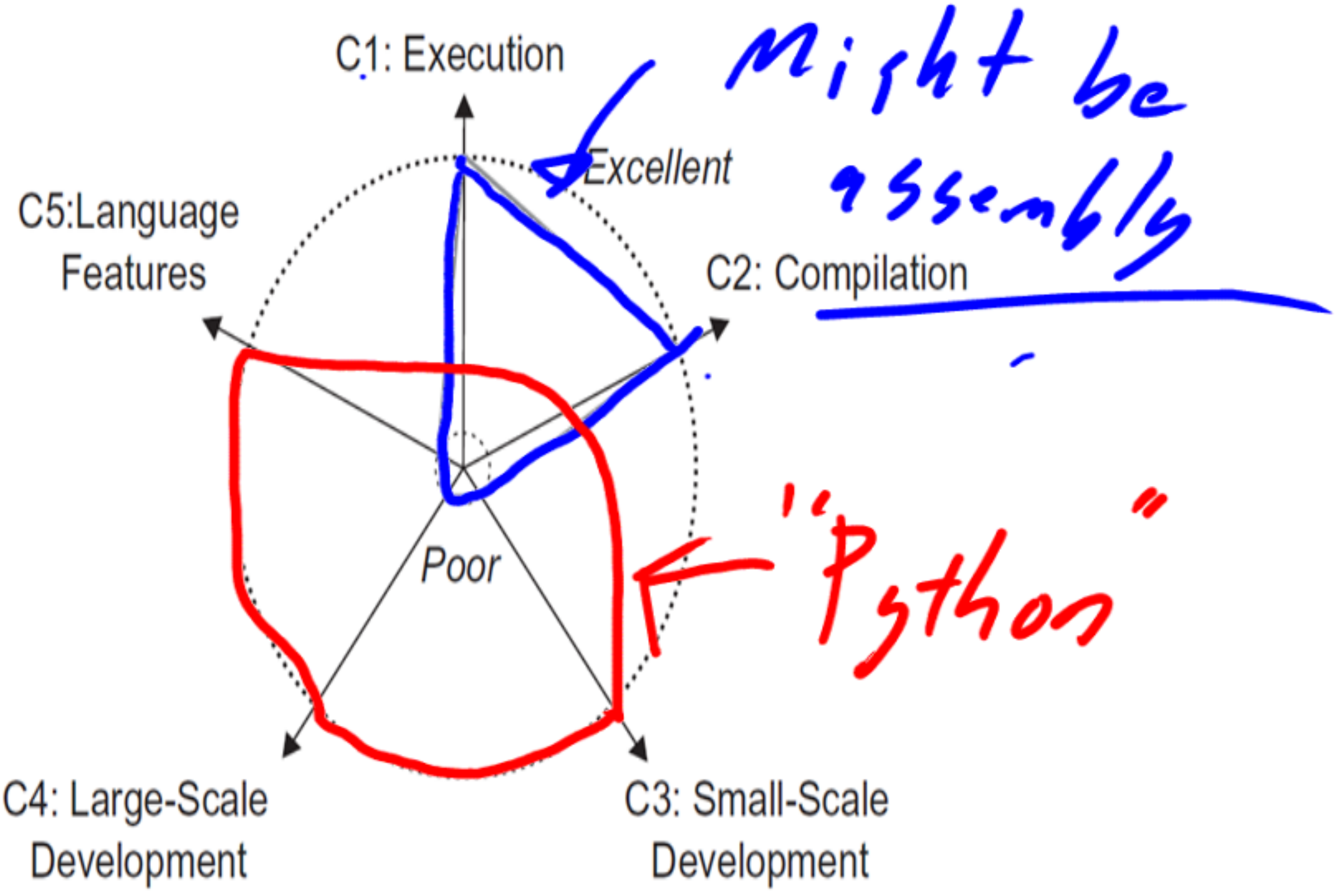
ROADMAP

- Today
  - Midterm Exam
  - Programming Languages
- Wednesday
  - An Introduction to audio
- Friday (Tentative)
  - Real Time Coding Standards - Misra C / C++ :

OBJECTIVES

- Explain the 5 items we must consider when choosing a programming language for real time systems
- Explain the difference between procedural languages and object oriented languages
- Critique the Java language for usage in Real Time Systems
- Optimize source code using well known optimization techniques, such as
  - Repeated calculations
  - Constant folding
  - Loop invariance removal
  - Induction variance
  - Loop unrolling
  - Loop jamming

SELECTING A PROGRAMMING LANGUAGE



- C1: Economy of execution
  - How fast does a program run
- C2 Economy of Compilation
  - How long does it take to go from multiple source files to an executable file
- C3 Economy of Small-Scale development
  - How hard must an individual programmer work?
- C4 Economy of Large Scale Development
  - How hard must a team of programmers work?
- C5 Economy of Language features
  - How hard is it to learn or use a programming language?

- Why is it still used?

— Device Drivers
— Specialized Hw + ...
access

PROCEDURAL LANGUAGES

- Ada, C, Fortran
  - Languages in which the action of the program is defined by a set of operations executed in sequence
- Strongly types versus weakly typed languages
  - Strongly typed – Each variable must have a specific type and must be declared as that type before usage
  - Strongly typed languages – Prohibit the mixing of data types
  - Compiler checking – Data types are confirmed at compile time instead of runtime
  - Weakly types language – The programmer does not need to define all types upfront

EXCEPTION HANDLING IN PROCEDURAL LANGUAGES

- ANSI C: Raise and signal
  - Creating exception handlers
- Void (*signal (int S, void (*func)(int)))(int)
  - Function prototype for an exception handler within a c program
- Int raise(int S)
  - Invokes the handler related to a given exception
- Example code

CARDELLI'S METRICS

- Variable Typing
  - -> Improves code generation and quality
- Compilation
  - Efficient, because each module is compiled independently
- Small Scale Development
  - Efficient, as type checking catches many errors and errors are nearby
  - Coding style can greatly help development
- Large Scale development
  - Code can be rearranged without global effects

OBJECT ORIENTED LANGUAGES

- Increased programmer productivity versus procedural languages
- Increased software reliability
- Higher potential for code reuse

CARDELLI'S LANGUAGES

SE3910 REAL TIME SYSTEMS

WHAT'S WRONG WITH JAVA FOR REAL TIME SYSTEMS

- Unpredictibility
- Garbage Collection

- Run time binding
  - Adds delay to the execution of the program

- Java Virtual Machine
  - We're sitting on top of another OS, divorced from the actual system.

OPTIMIZATION OF CODE

- How can we optimize code?
  - Repeated calculations
  - Constant folding
  - Loop invariance removal
  - Induction variance
  - Loop unrolling
  - Loop jamming

REPEATED CALCULATIONS

- X = 6 + a * b;
- Y = a * b + z;

- How could we simplify this?

- X = 2.0 * x * 3.125;

CONSTANT FOLDING

LOOP INVARIANCE REMOVAL

- X = 100;
- While (x > 0)
- {
  - x = x – y + z;
- }

INDUCTION VARIANCE

- For (I = 1; I <= 10; i++)
- {
  - A[i+1] = 0;
- }

LOOP UNROLLING

- Duplicate instructions executed in a loop to reduce the number of operations and hence the loop overhead incurred

```
For (I = 1; i<=6;i++)
{
        A[i] = a[i]*8;
}
```

LOOP JAMMING

- Combine similar loops together to improve performance

```
For (I = 1; i<=100;i++)
{
        X[i] = y[i]*8;
}
For (I = 1; I <= 100; i++)
{
        Z[i] = x[i] * y[i];
}
```