DEVELOPING DEVELOPERS

MATTHIAS FELLEISEN, PLT, NUPRL

THE BEGINNING (1992/95)

CS I

C,
Pascal,
Ratfor,
Fortran

AP, high schools

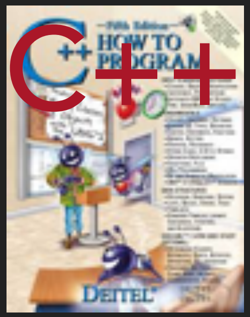
the "better math"

"computational" physics

economics "come alive"

THE BEGINNING (1992/95)

- Robby Findler
- Kathi Fisler
- Matthew Flatt
- Shriram Krishnamurthi
- Emmanuel Schanzer



C++

CS I

C,
Pascal,
Ratfor,
Fortran

Dist Sys Dev

Sw Dev ~ just before students study Sw Eng

CS II: if CS I is about "Scheme", what roles does CS it serve?

TeachScheme!

Program By Design

Bootstrap

- Robert Cartwright (Rice)
- Robby Findler
- Peter Druschel (MPI-SWS)
- Mike Ernst (UW)

THE BEGINNING (1992/95)

CS I

C,
Pascal,
Ratfor,
Fortran

CS II: if CS I is about "Scheme", what roles does CS it serve?

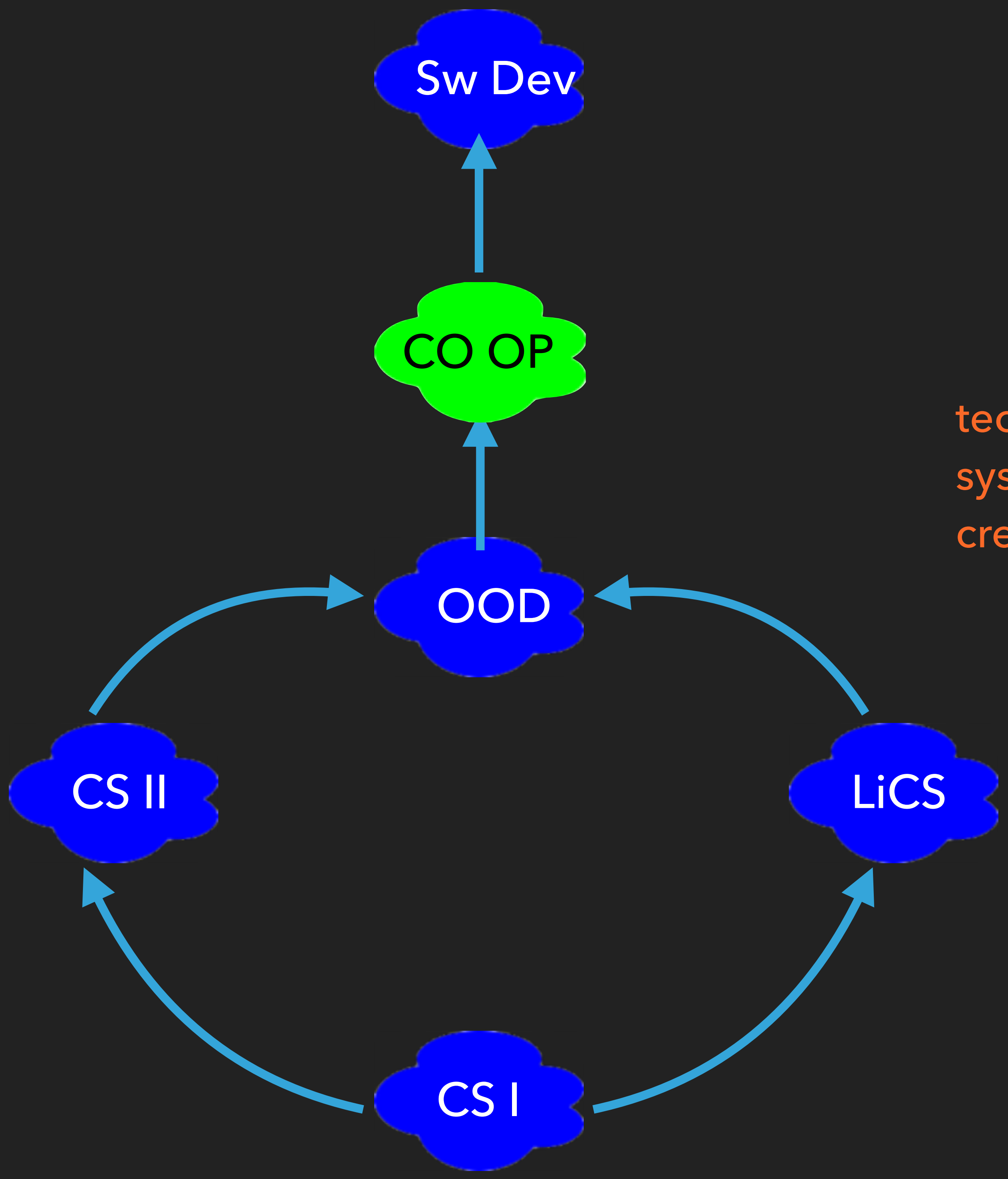
Sw Dev ~ just before students study Sw Eng

Dist Sys Dev

WHERE I AM TODAY

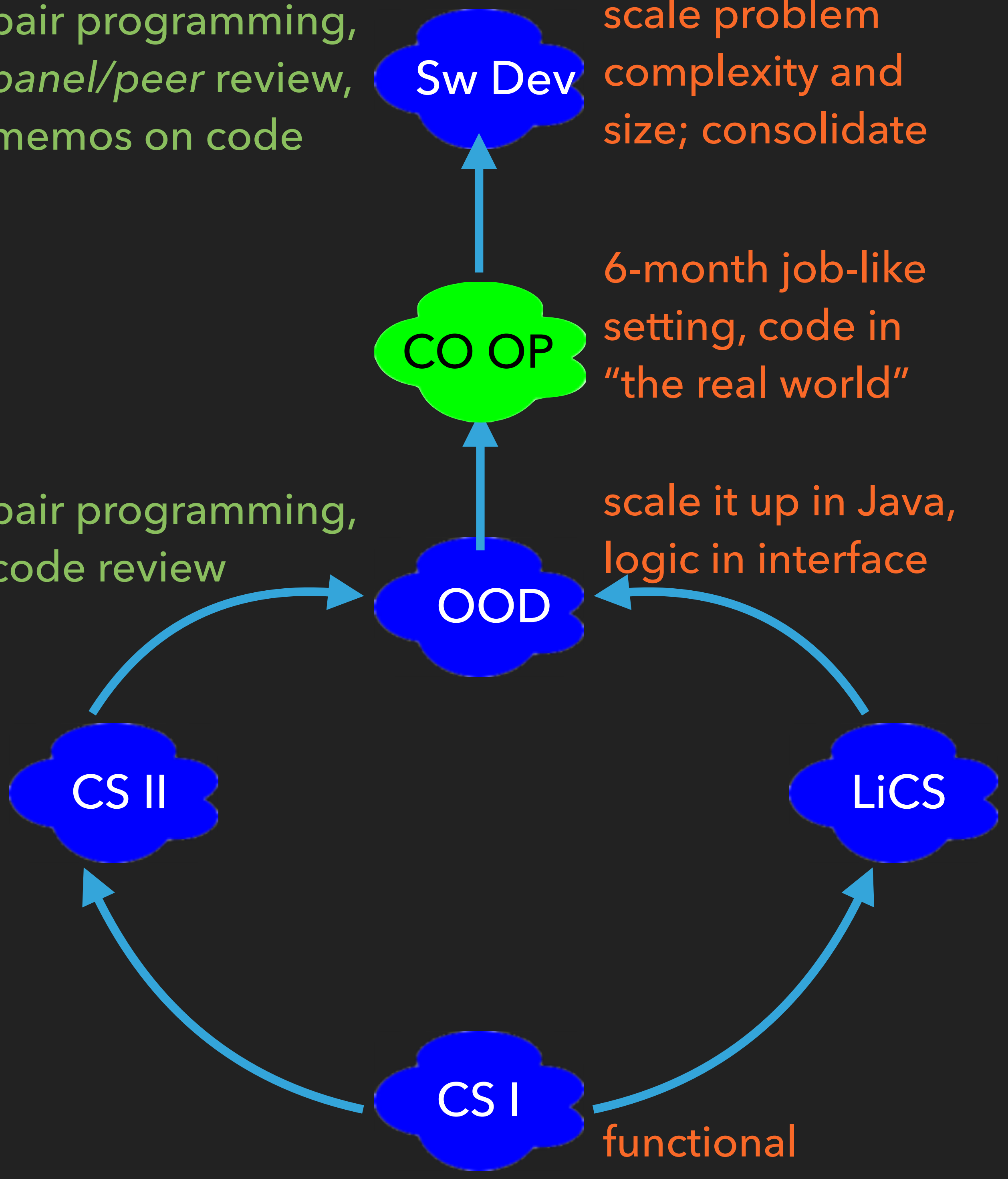
TODAY'S WORLD @ NU CCIS: TECHNICAL SKILLS & COMMUNICATION SKILLS

communication skills: conversing about code



technical skills: systematic creation of code

TODAY'S WORLD @ NU CCIS: TECHNICAL SKILLS & COMMUNICATION SKILLS



pair programming,
panel/peer review,
memos on code

scale problem
complexity and
size; consolidate

6-month job-like
setting, code in
"the real world"

pair programming,
code review

scale it up in Java,
logic in interface

systematic design,
typed & OOPL (Java)

pair programming,
code review

proving theorems about
(functional) code, dual to
systematic design
pair programming

pair programming

functional
programming for
systematic design

TODAY'S WORLD @ NU CCIS: TECHNICAL SKILLS & COMMUNICATION SKILLS

- Why should we care about software development?
- What are doing wrong and what can we do better?
- How can we change our introductory software development curriculum?

WHY CARE ABOUT SOFTWARE DEVELOPMENT?

SOFTWARE DEVELOPMENT IS A CHALLENGE & AN OPPORTUNITY

Do our colleagues really not care?

- research problems for the lone ranger
- software as prototypes, at most
- few maintain software over years

- there is no research here, just teaching
- coding is easy anyways
- kids get jobs if they can spell "C"

THE MORAL IMPERATIVE

Thesis

Our graduates will find jobs as long as they can spell the name of the C programming language. Every minute we spend on them, we won’t spend on research and papers and grants.

THE MORAL IMPERATIVE

AntiThesis

99%

Our graduates will find jobs as long as they can spell the name of the C programming language. Every minute we spend on them, we won't spend on research and papers and grants.

SynThesis

Colleges promise – in our name – that we add value to our undergraduates for the rest of their lives. We have *a moral obligation* to live up to our premise and *a commercial one,* too.

DEVELOPING SOFTWARE IS HARD.

Thesis

Programming is easy, we can teach it one or two courses. The software architects design, and programmers just code. But architecture is software engineering, not software development

DEVELOPING SOFTWARE IS HARD.

AntiThesis

Programming is easy, we can teach it one or two courses. The software architects design, and programmers just code. But architecture is software engineering, not software development

workmanship of certainty vs workmanship of risk

David Pye, *The Nature and Art of Workmanship*, Cambium 2002

SynThesis

Software development is “workmanship of risk” because (most of) it is a thinking activity and articulating thoughts. And that is hard.

DEVELOPING SOFTWARE IS HARD.

AntiThesis

Programming is easy, we can teach it one or two courses. The software architects design, and programmers just code. But architecture is software engineering, not software development

workmanship of certainty vs workmanship of risk

David Pye, *The Nature and Art of Workmanship*, Cambium 2002

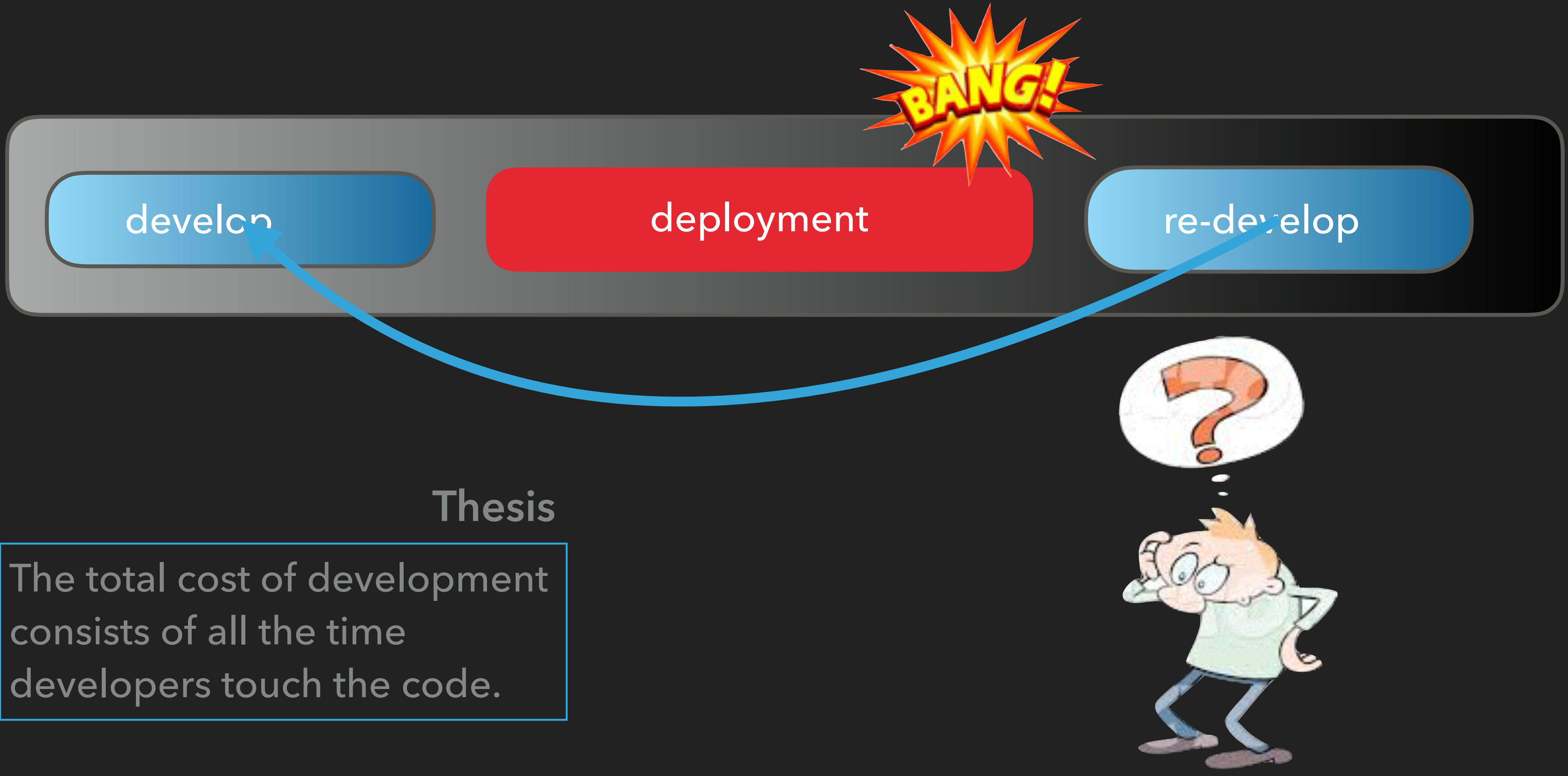
SynThesis

Programs must be written for people to read, and only incidentally for machines to execute.

Abelson and Sussman, *Structure and Interpretation of Computer Programs,* MIT Press, 1984

WE MUST LEARN TO APPRECIATE DEVELOPMENT TIME & QUALITY.

What is the cost of turning thoughts into code?



Thesis

The total cost of development consists of all the time developers touch the code.

WE MUST LEARN TO APPRECIATE DEVELOPMENT TIME & QUALITY.

- Developers are scarce.
- Ergo, developer time is scarce (expensive).
- Companies should worry about how they use their developers time.
- Developers should care where they spend their (collective) time.

YOUR DEVELOPERS HATE VACATIONS.

DO THEY ALL HAVE RELATIONSHIP TROUBLE ALL THE TIME?

ALL DEVELOPERS HAVE TEENAGERS AT HOME. BEEN THERE, DONE THAT.

IN SUMMARY: WHY DO WE CARE

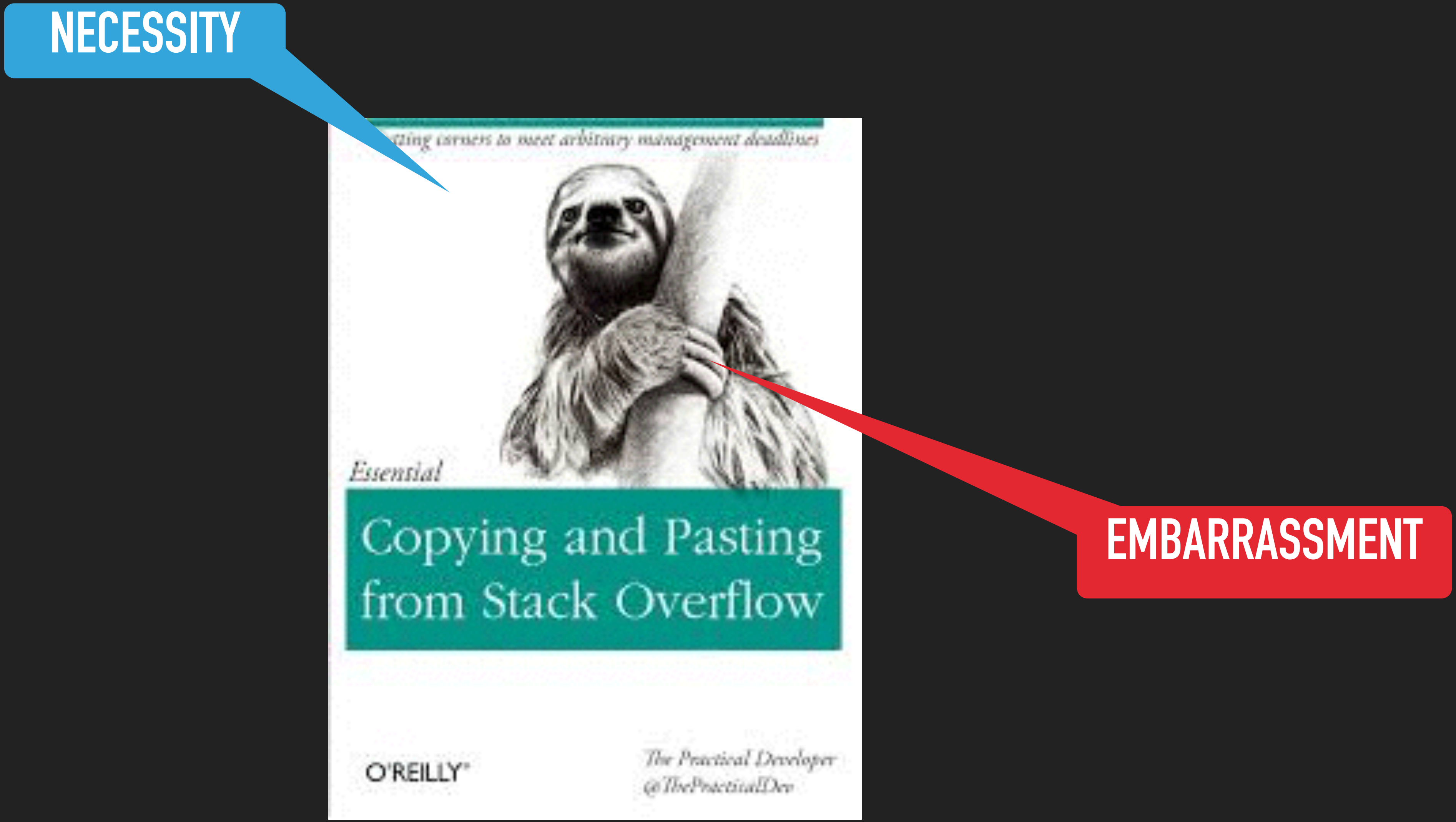
We have a *moral* and *commercial* obligation.

We actually don’t know how to teach software development properly.

There is a *research* and a *teaching* opportunity.

WHAT ARE WE DOING WRONG, WHAT CAN WE DO DIFFERENTLY

WHAT DO OUR STUDENTS KNOW WHEN THEY GRADUATE



NECESSITY

tting corners to meet arbitrary management deadlines

Essential

Copying and Pasting from Stack Overflow

O'REILLY

The Practical Developer
@ThePracticalDev

EMBARRASSMENT

WHAT WE TEACH WHEN WE TEACH ‘CODING’

- Algol 60/Simula 67
- Pascal
- C
- Scheme
- C++
- Eiffel
- Haskell
- Java
- Alice/Scratch
- Python

10 cool languages in 30 years

COSMO
Sayı 53
HEDİYE KİTAP!
50 SEKS ÖNERİSİ
Mehmet Aslan'dan Akıllı Flört Taktikleri
BİKİNİ VE MAYO DOSYASI
Karnınızı Güçlendirin, Yaza Bomba Gibi Girin
En Yeni Genç Yetenekler
Bu İsimlerle Çok Sık Karşılaşacaksınız
HILARY DUFF
COSMO'nun KONUKLARI

Can we do better?

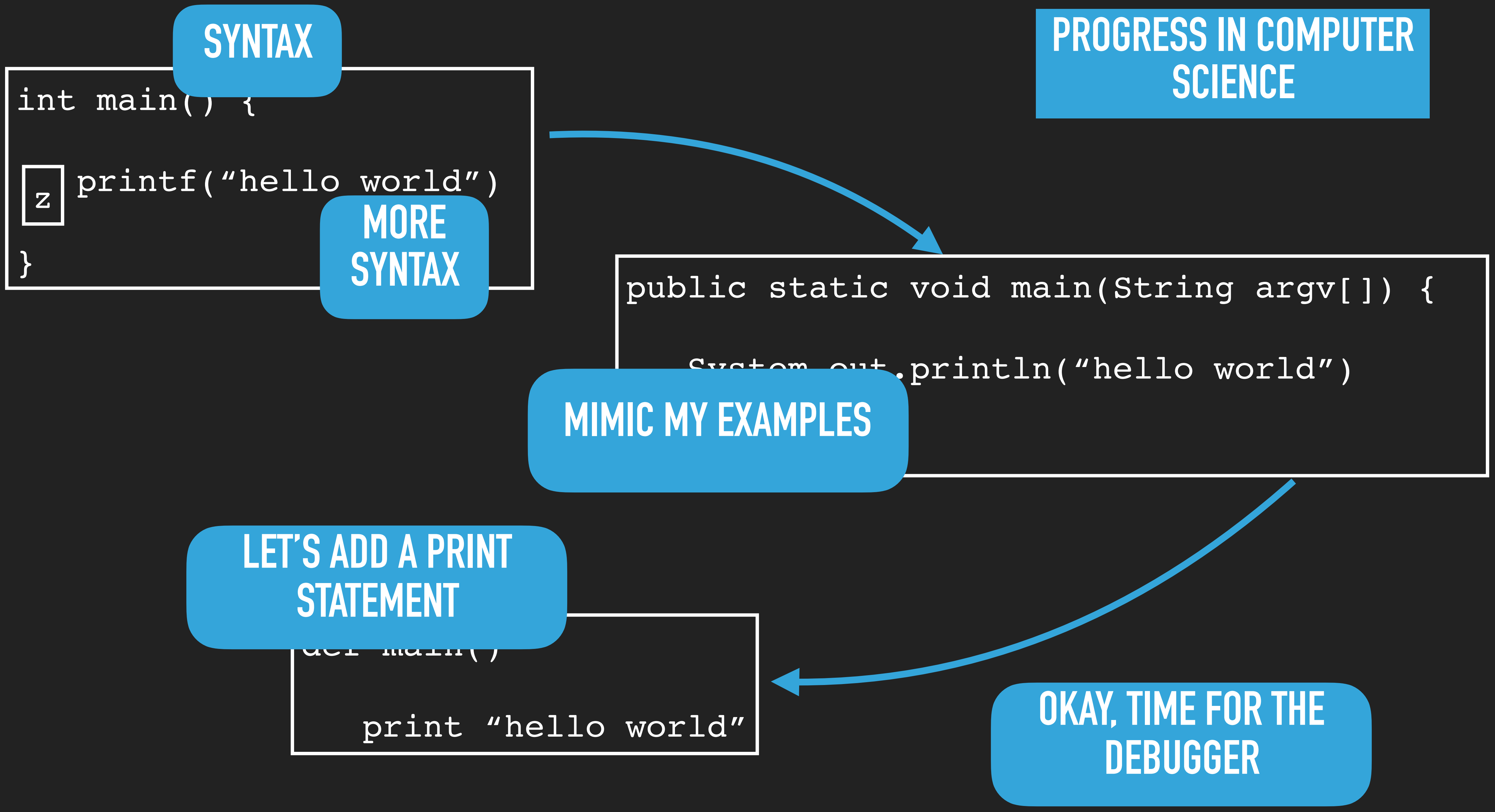
WHAT WE TEACH WHEN WE TEACH ‘CODING’

Is this all we offer?

- "hello world"
- puzzles
- graphics
- GUIs
- web connections
- apps for your phone
- parallel processing tricks
- hack fests
- 3D printing

10 sexy tricks in 30 years

WHAT WE TEACH WHEN WE TEACH ‘CODING’



SYNTAX

```
int main() {

z printf(“hello world”)

}
```

MORE SYNTAX

PROGRESS IN COMPUTER SCIENCE

```
public static void main(String argv[]) {

    System.out.println(“hello world”)
```

MIMIC MY EXAMPLES

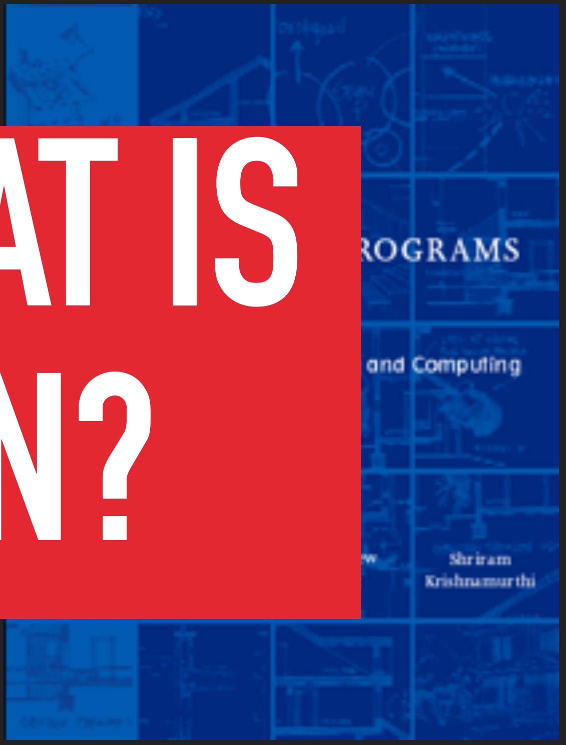
LET’S ADD A PRINT STATEMENT

```
def main()

    print “hello world”
```

OKAY, TIME FOR THE DEBUGGER

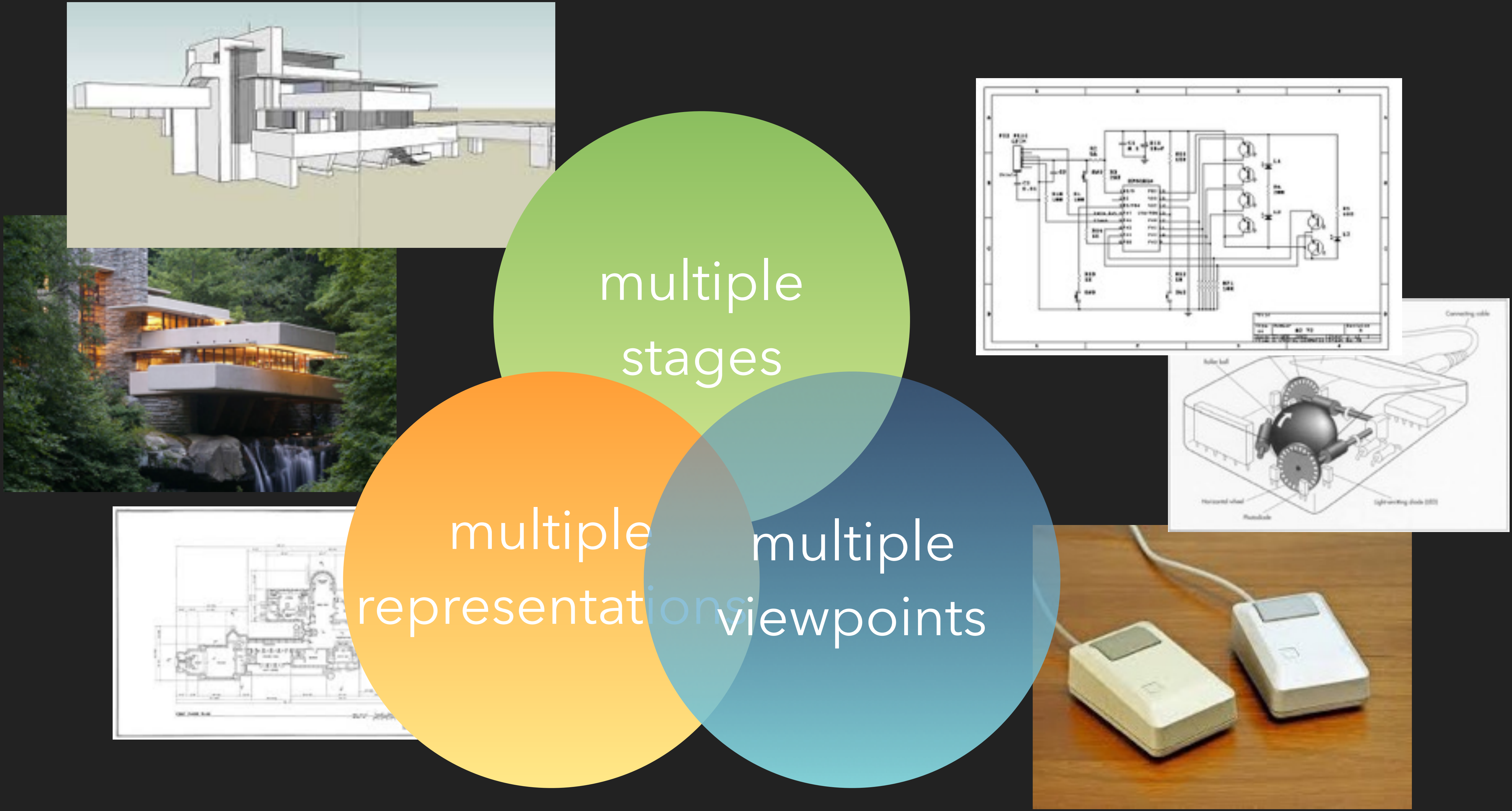
WHAT COULD WE TEACH? DEVELOPMENT ~ SYSTEMATIC DESIGN

- Design all the way down.
- Empower student
  help themselves.
- Inspect and review code.



BUT WHAT IS DESIGN?

WHAT COULD WE TEACH? DEVELOPMENT ~ SYSTEMATIC DESIGN



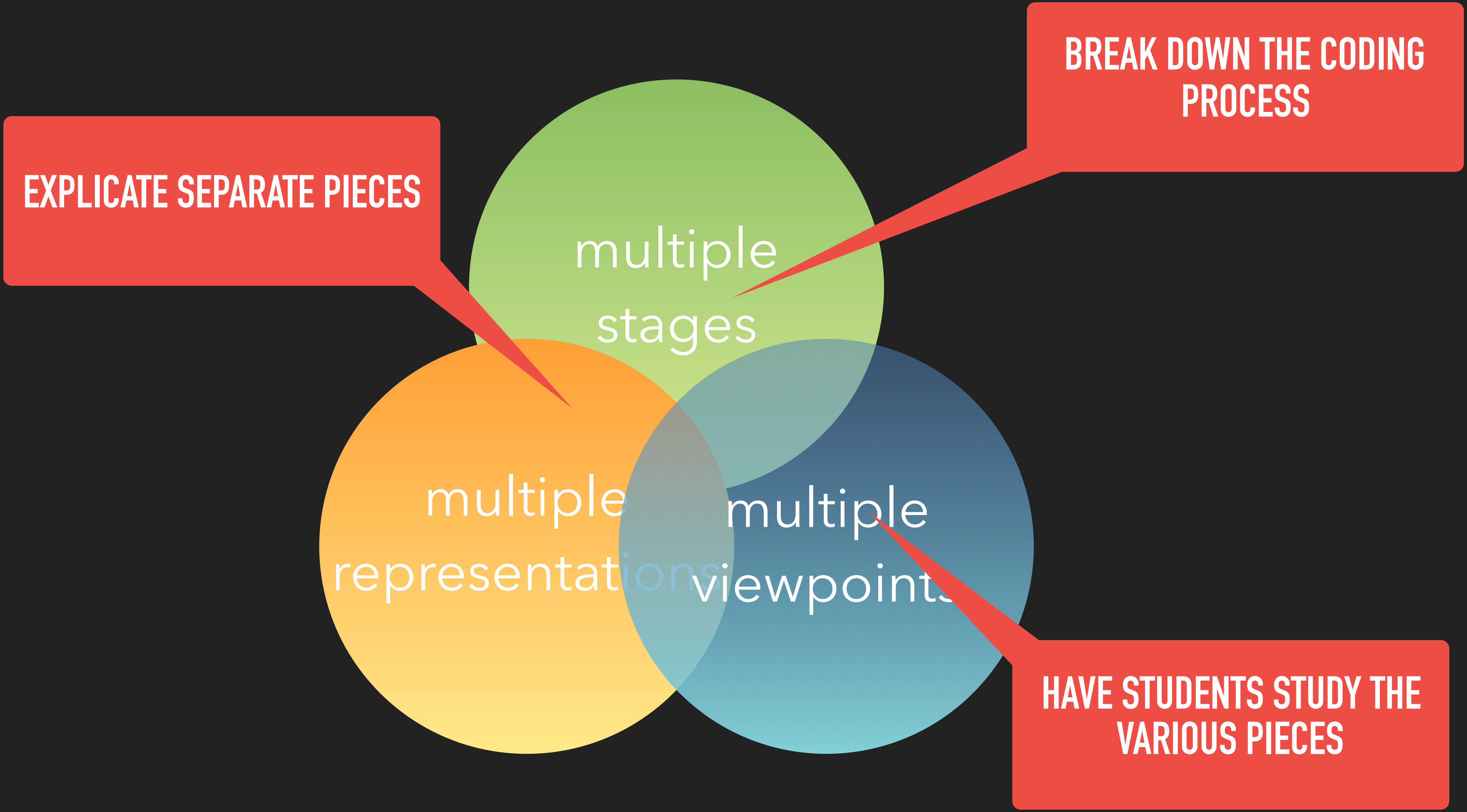
multiple stages

multiple representations

multiple viewpoints

(This slide stolen from Shriram Krishnamurthi)

WHAT COULD WE TEACH? DEVELOPMENT ~ SYSTEMATIC DESIGN



multiple stages

multiple representations

multiple viewpoints

EXPLICATE SEPARATE PIECES

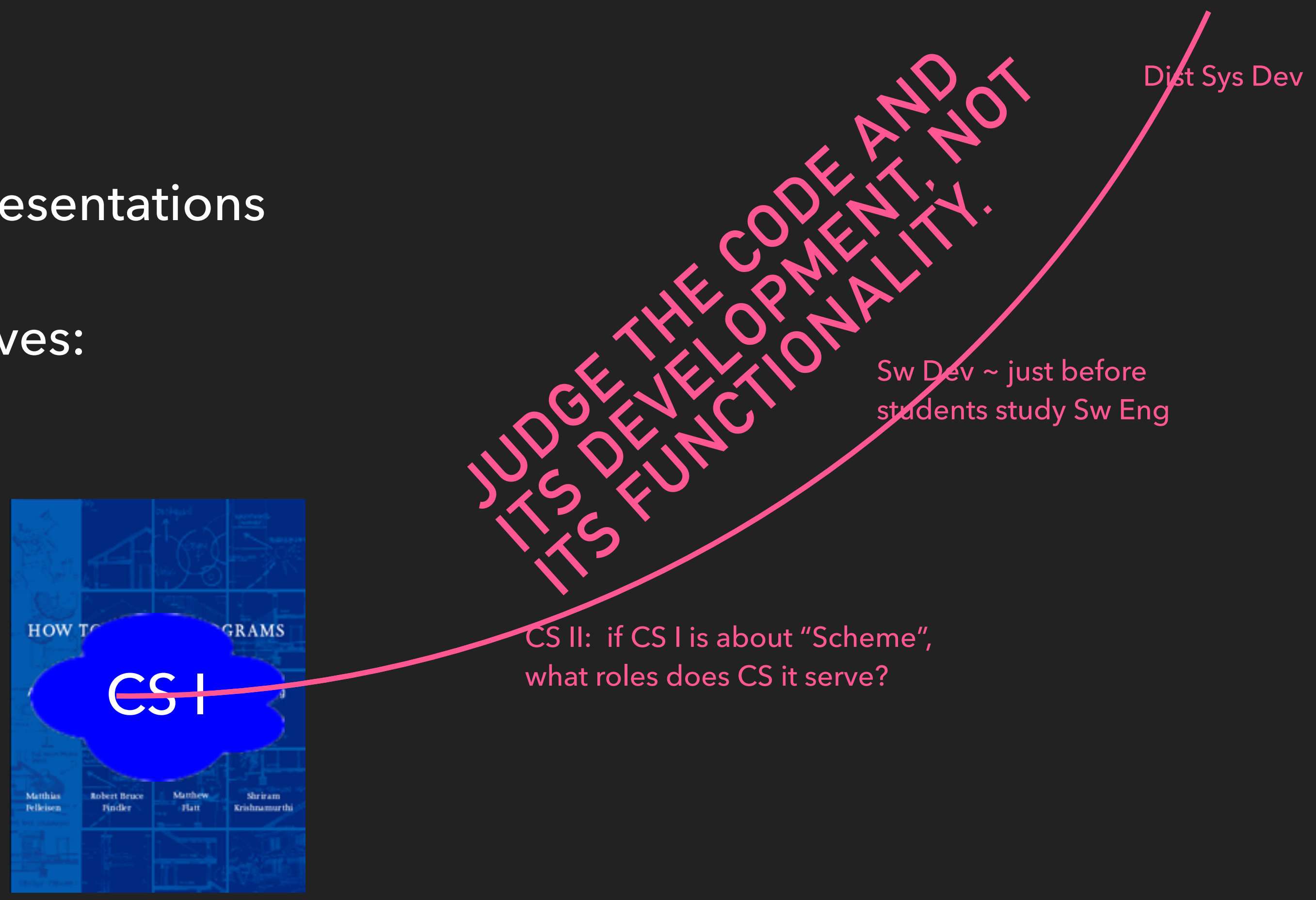
BREAK DOWN THE CODING PROCESS

HAVE STUDENTS STUDY THE VARIOUS PIECES

IN SUMMARY: WHAT CAN WE DO

At every scale of software development, students must learn to

- stage the development process.
- understand software via multiple representations
- view code from at least two perspectives: producer and consumer.



JUDGE THE CODE AND ITS DEVELOPMENT, NOT ITS FUNCTIONALITY.

Dist Sys Dev

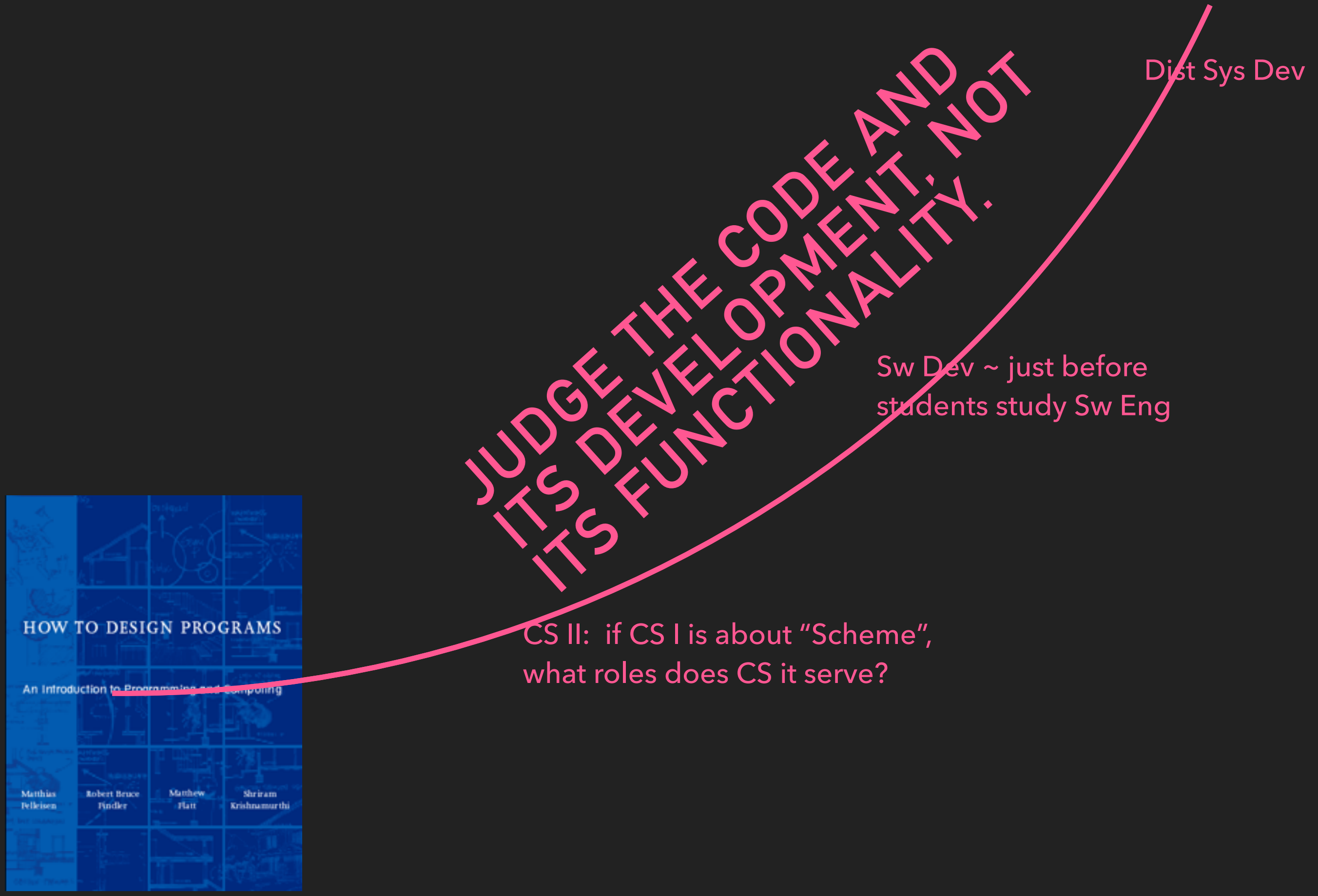
Sw Dev ~ just before students study Sw Eng

CS II: if CS I is about “Scheme”, what roles does CS it serve?

HOW CAN WE CHANGE OUR SOFTWARE DEVELOPMENT CURRICULUM?

HOW CAN WE TEACH SYSTEMATIC DESIGN ACROSS THE SCALE

- We need *several* courses that inspect students' code for its communicative qualities.
- Every course must enhance both
  - design skills
  - communication skills
- The courses must be coordinated.



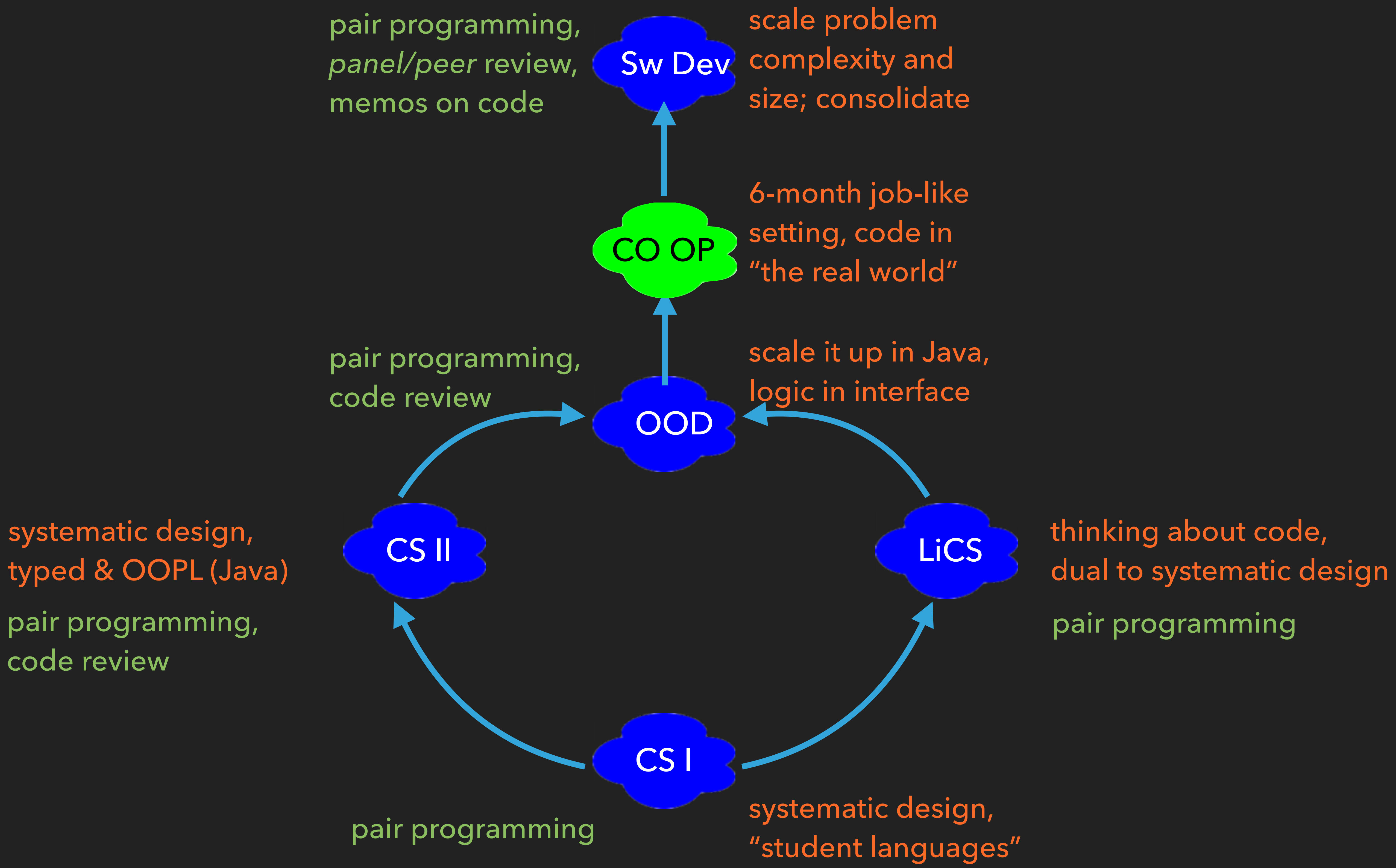
JUDGE THE CODE AND ITS DEVELOPMENT, NOT ITS FUNCTIONALITY.

Dist Sys Dev

Sw Dev ~ just before students study Sw Eng

CS II: if CS I is about “Scheme”, what roles does CS it serve?

HOW CAN WE TEACH SYSTEMATIC DESIGN ACROSS THE SCALE



pair programming,
panel/peer review,
memos on code

scale problem
complexity and
size; consolidate

6-month job-like
setting, code in
"the real world"

pair programming,
code review

scale it up in Java,
logic in interface

systematic design,
typed & OOPL (Java)

pair programming,
code review

thinking about code,
dual to systematic design

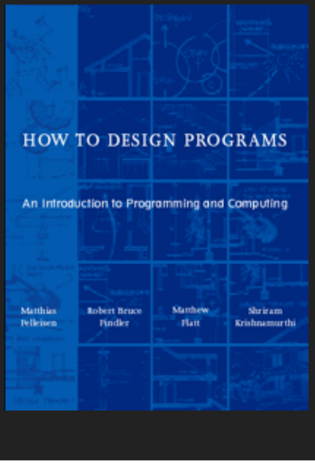
pair programming

pair programming

systematic design,
"student languages"

HOW CAN WE TEACH SYSTEMATIC DESIGN AT THE PROGRAM LEVEL

- data analysis, data definition, data examples
- signature and purpose statement
- functional examples
- function template
- function definition
- tests and testing

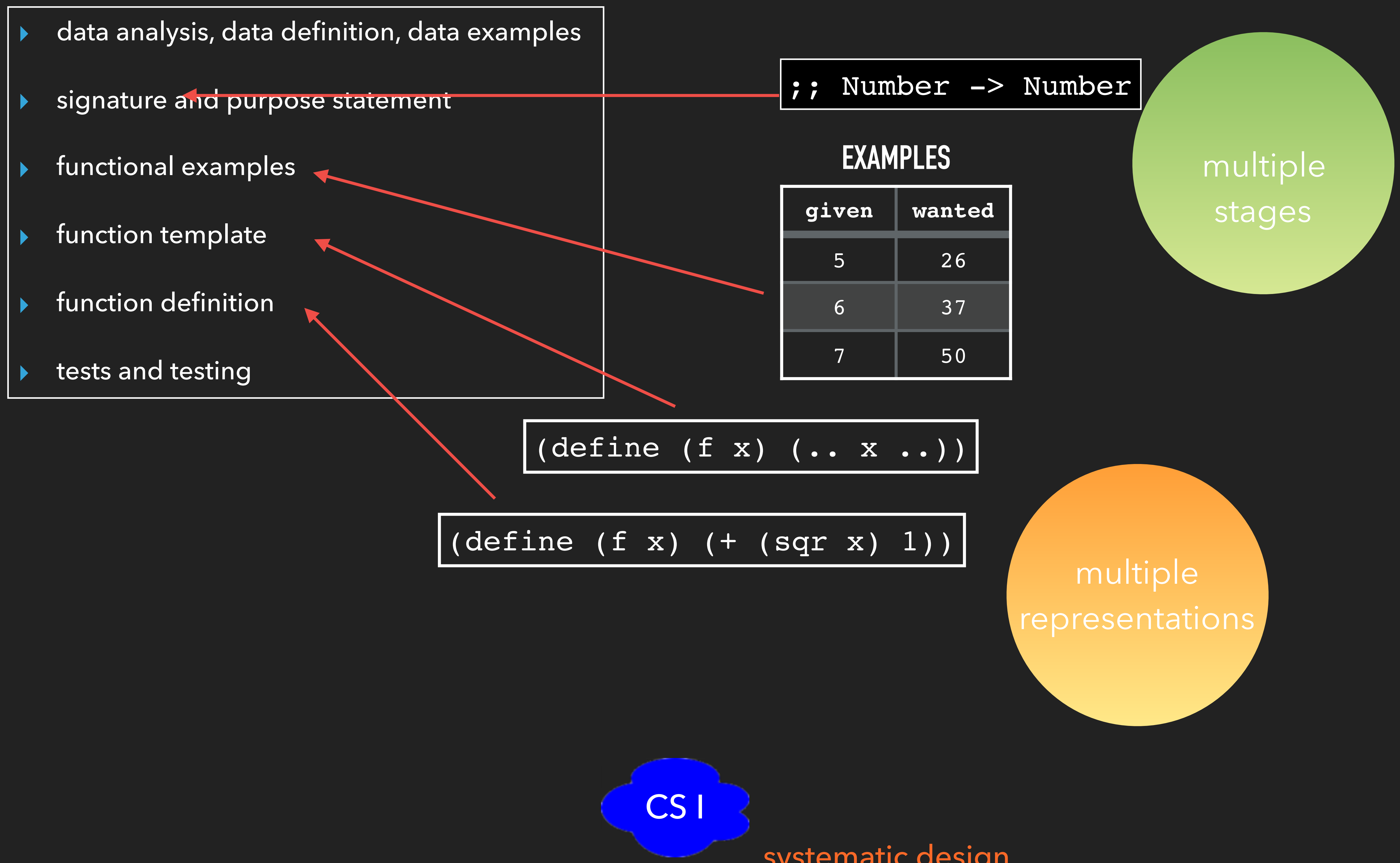


multiple stages

CS I

systematic design,
“student languages”

HOW CAN WE TEACH SYSTEMATIC DESIGN ACROSS THE SCALE



- data analysis, data definition, data examples
- signature and purpose statement
- functional examples
- function template
- function definition
- tests and testing

```
;; Number -> Number
```

EXAMPLES

given	wanted
5	26
6	37
7	50

```
(define (f x) (.. x ..))
```

```
(define (f x) (+ (sqr x) 1))
```

multiple stages

multiple representations

CS I

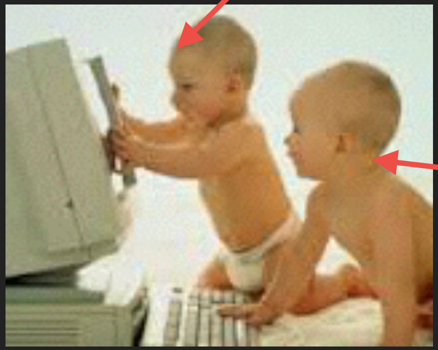
systematic design,
“student languages”

HOW CAN WE TEACH SYSTEMATIC DESIGN AT THE COMPONENT LEVEL

What are all these ()s and ;s doing here?

multiple viewpoints

"Co-pilot" ← reader



"Pilot" ← writer

CS II

LiCS

pair programming, code review

pair programming

CS I

pair programming

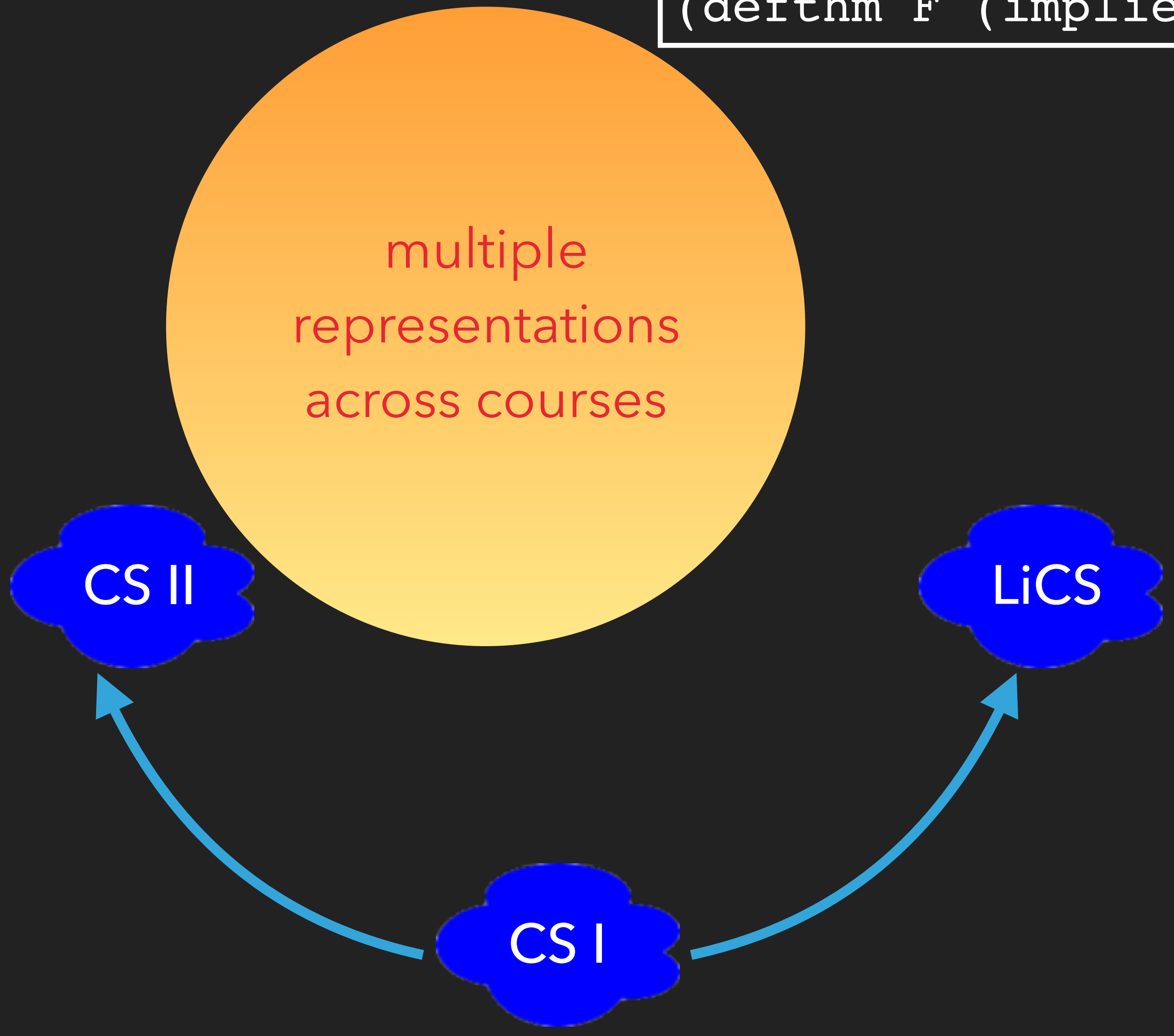
HOW CAN WE TEACH SYSTEMATIC DESIGN AT THE COMPONENT LEVEL

```
class Mathy {

 int f(int x) {

  return x*x+1;

 }

}
```

systematic design,
typed & OOPL (Java)

```
(defun f (x) (+ (sqr x) 1))

(defthm F (implies (natp x) (> (f x) x))
```

thinking about code,
dual to systematic design



multiple representations across courses

CS II

LiCS

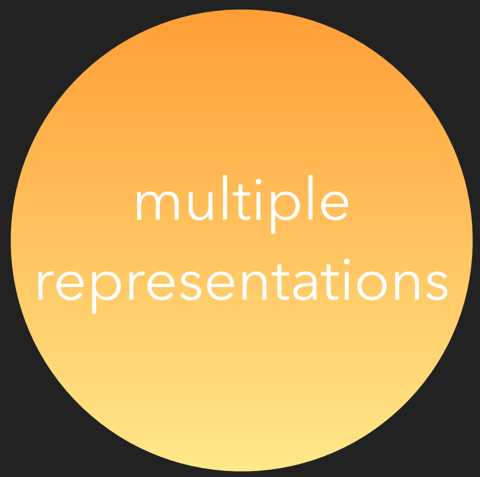
CS I

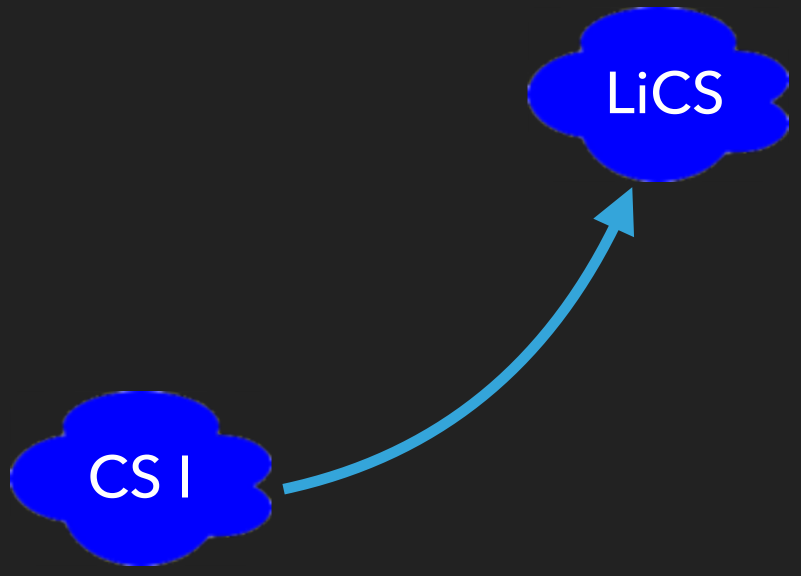
HOW CAN WE TEACH SYSTEMATIC DESIGN AT THE COMPONENT LEVEL

```
(defthm F (implies (listp l) (natp (f l)))
```

```
IH(l) <=>

   (implies (listp l) (natp (f l)))

by cases on the structure of l:

 — l is ‘(): 0

 — l is (cons A k) .. IH(k) ..
```

```
(defun f (l)

  (cond

    ((endp l) 0)

    (t (+ (f (cdr l)) 1)))
```





thinking about code,
dual to systematic design

HOW CAN WE TEACH SYSTEMATIC DESIGN AT THE COMPONENT LEVEL

Type checking enforces signatures before damage is done.

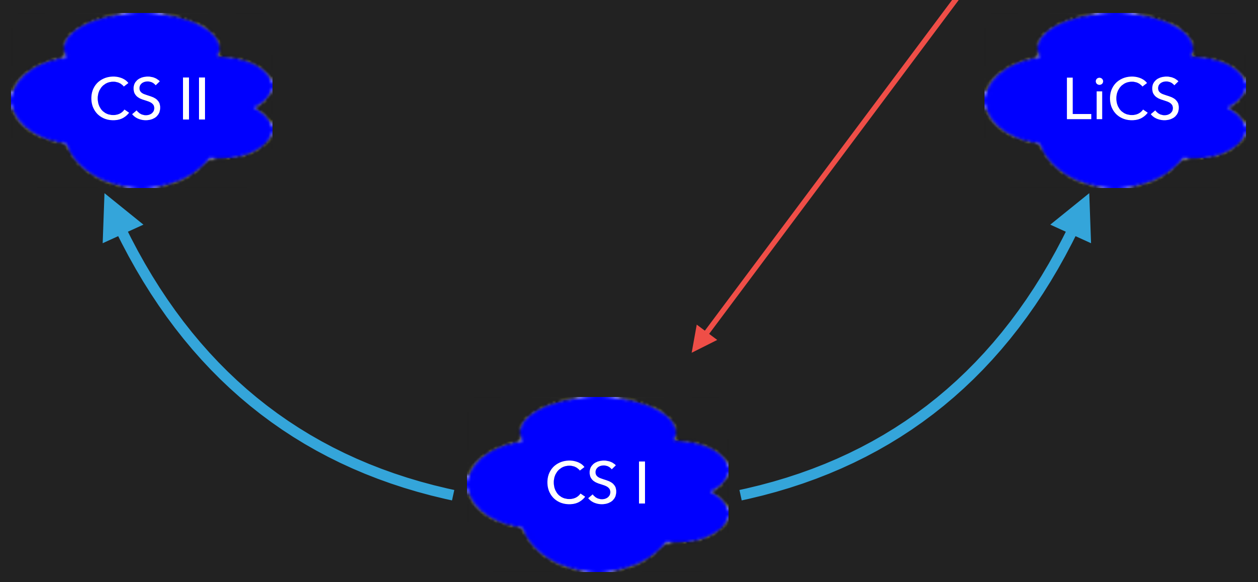
Object-oriented design turns functional design on its side (but that’s it).

multiple different kind of stages in downstream courses

First test, then formulate theorems.

Induction is the dual of structural recursion.

systematic design, typed & OOPL (Java)



thinking about code, dual to systematic design

HOW CAN WE TEACH SYSTEMATIC DESIGN AT THE COMPONENT LEVEL

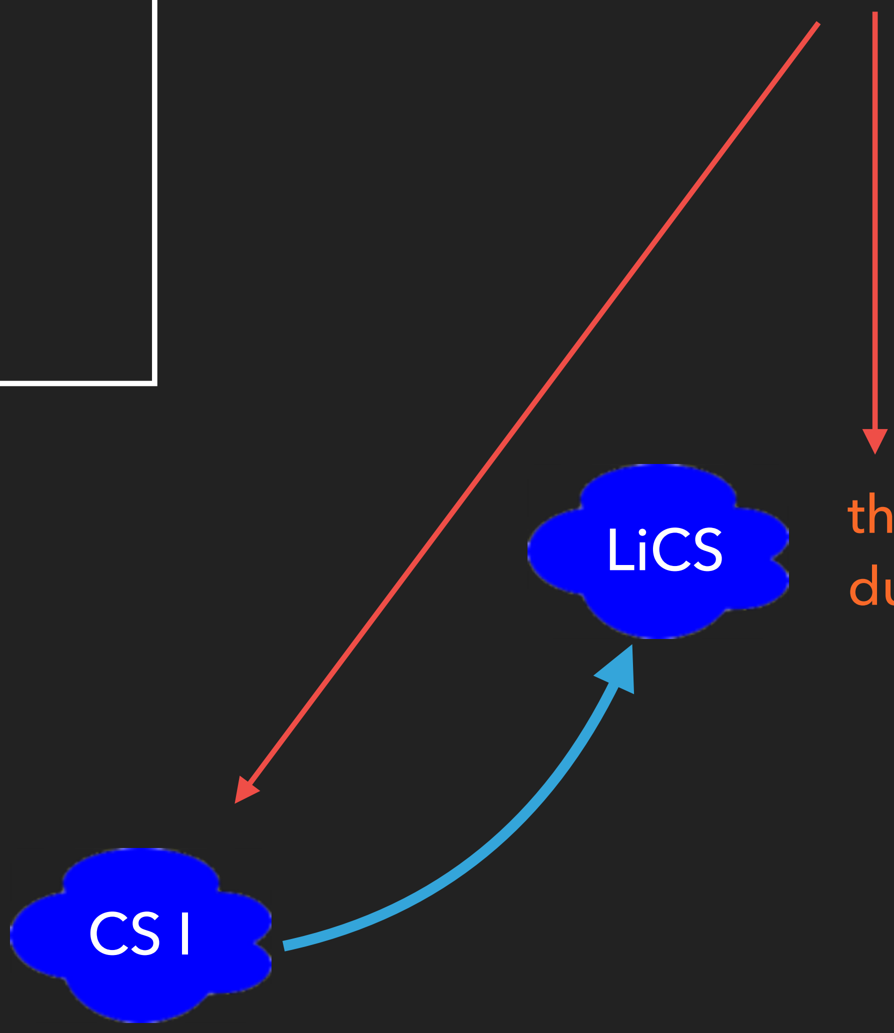
```
(defthm F (implies (listp l) (natp (f l)))
```

```
IH(l) <=>

   (implies (listp l) (natp (f l))

by cases on the structure of l:

 — l is ‘(): 0

 — l is (cons A k) .. IH(k) ..
```

```
(defun f (l)

  (cond

    ((endp l) 0)

    (t (+ (f (cdr l)) 1)))
```

First test, then formulate theorems.

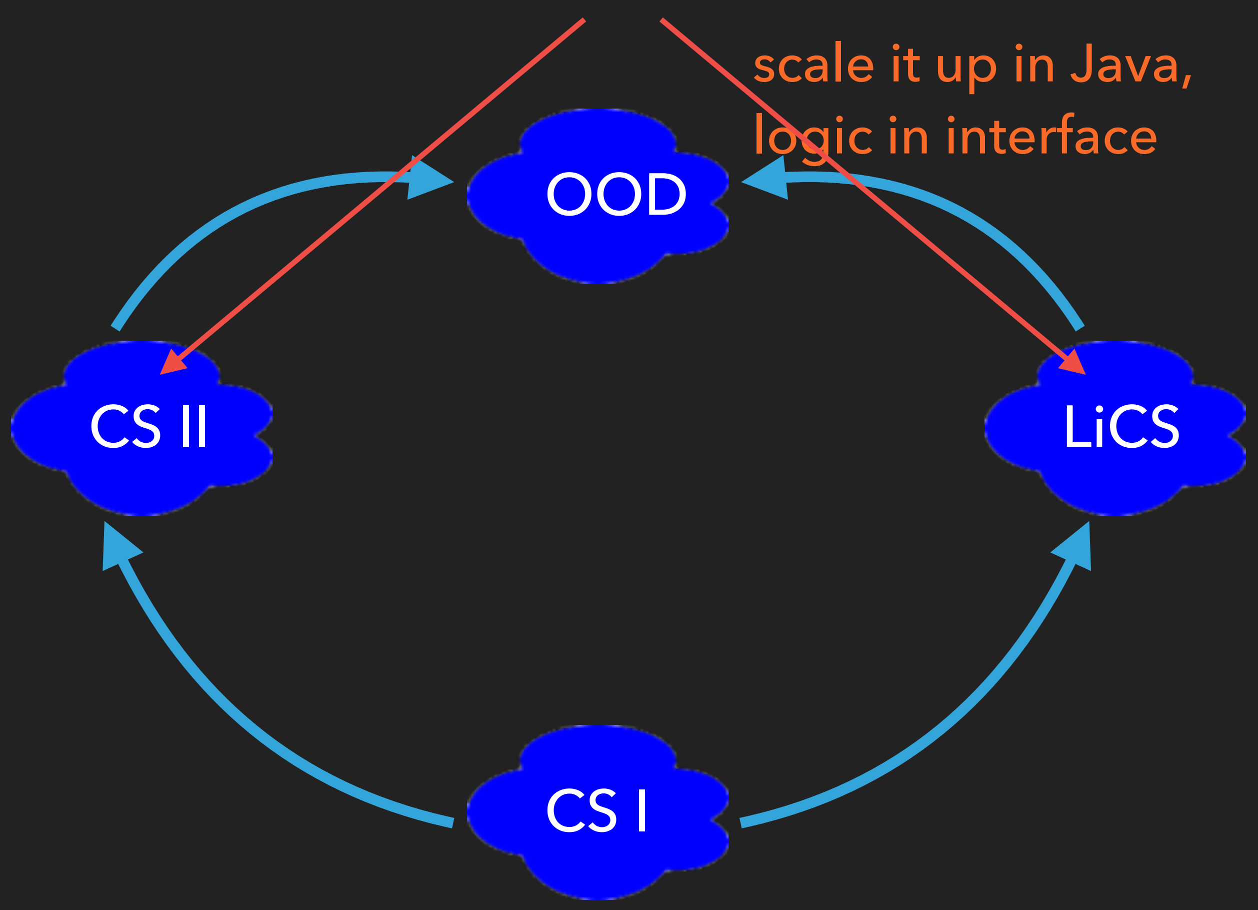
Induction is the dual of structural recursion.



thinking about code,
dual to systematic design

HOW CAN WE TEACH SYSTEMATIC DESIGN FOR SMALL SYSTEMS

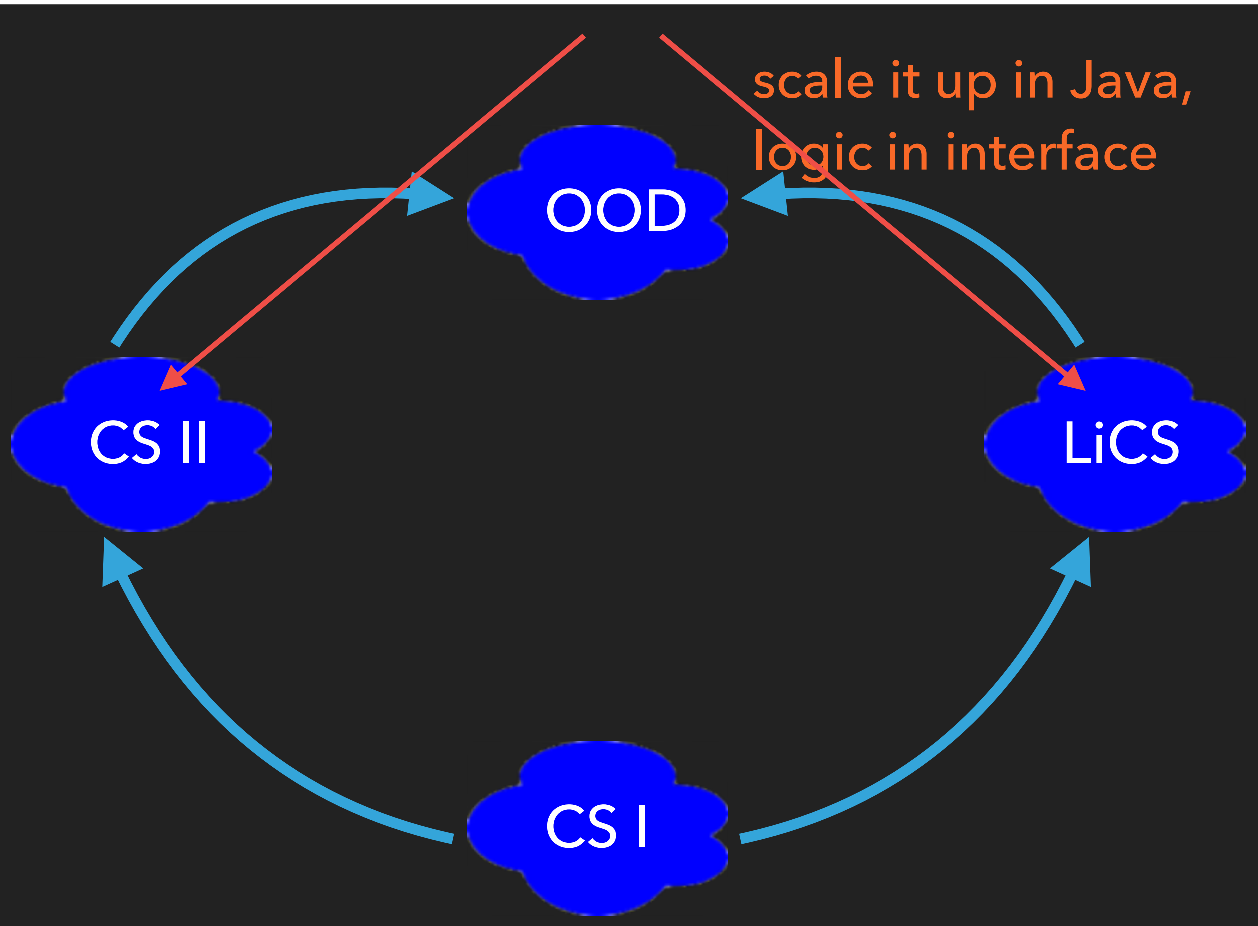
Bring distinct representations together in one unit of code



join multiple representations

HOW CAN WE TEACH SYSTEMATIC DESIGN FOR SMALL SYSTEMS

```
interface ISpecies {

   @pre this.oneIsHungry()

   @post !@result.isPresent() || @result.get() = s +1

   Optional<Integer> feed1(int s)

}
```



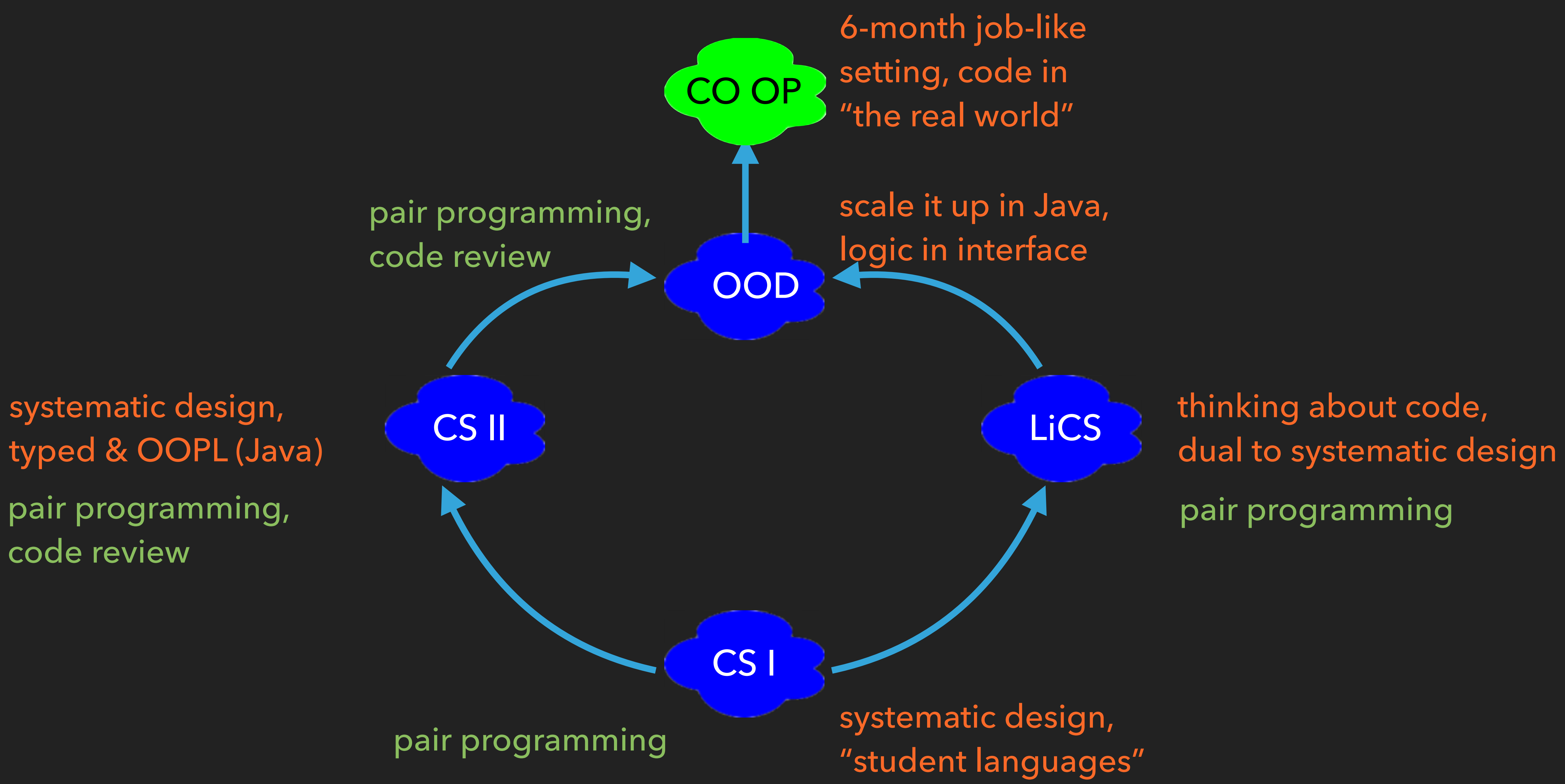
scale it up in Java,
logic in interface

join multiple representations

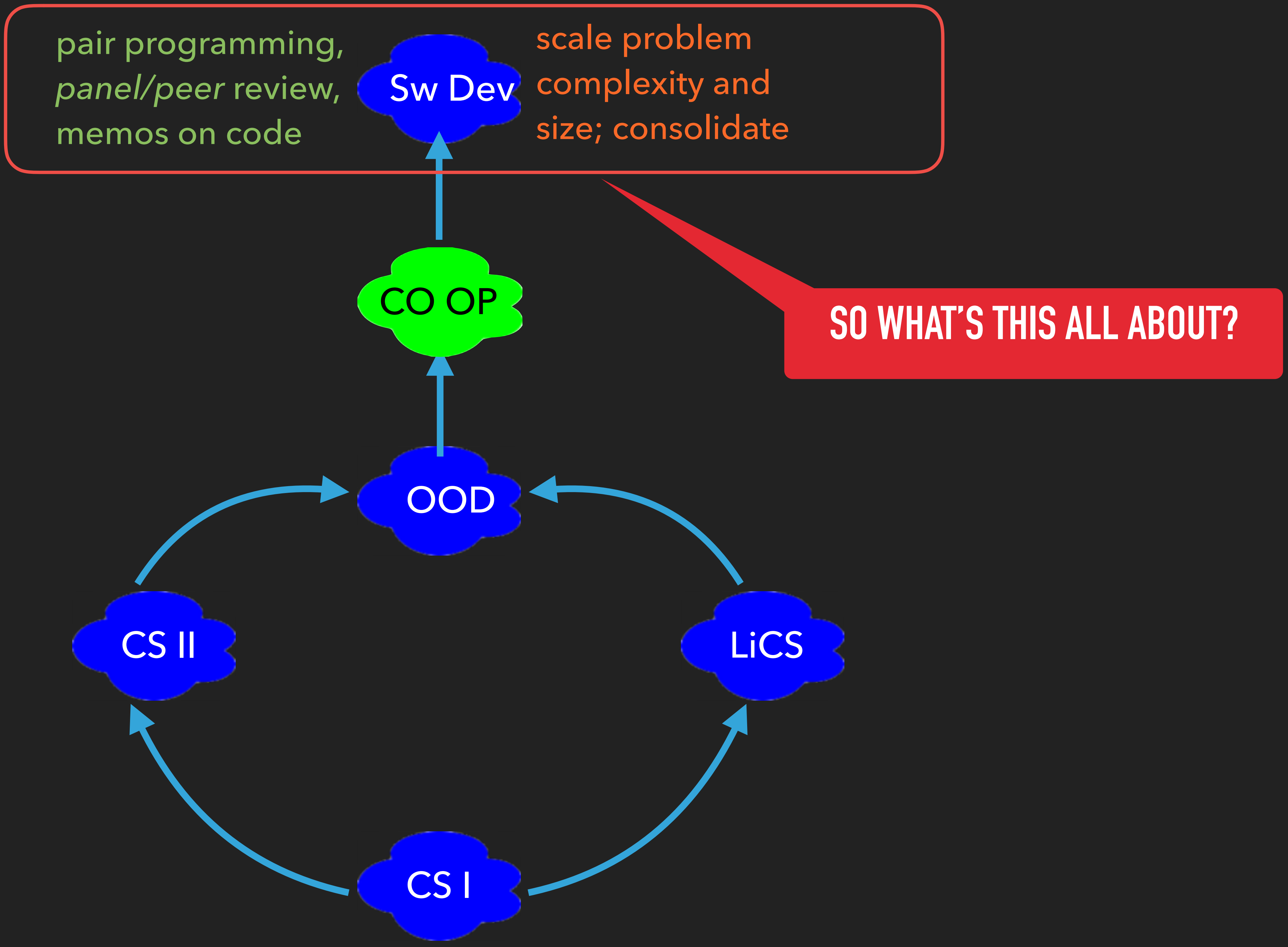
HOW CAN WE TEACH SYSTEMATIC DESIGN FOR SMALL SYSTEMS

My first co-op: “Day 4 and I am already demoing code. I LOVE MY LIFE.”

A co-op employer often expects students to pick up yet another language.



TODAY'S WORLD @ NU CCIS: TECHNICAL SKILLS & COMMUNICATION SKILLS



pair programming, *panel/peer* review, memos on code

Sw Dev

scale problem complexity and size; consolidate

SO WHAT'S THIS ALL ABOUT?

CO OP

OOD

CS II

LiCS

CS I

A FINAL COURSE ON SOFTWARE DEVELOPMENT

(NOT SOFTWARE ENGINEERING)

TODAY'S WORLD @ NU CCIS: TECHNICAL SKILLS & COMMUNICATION SKILLS

SPECIALIZATIONS & CAP STONES

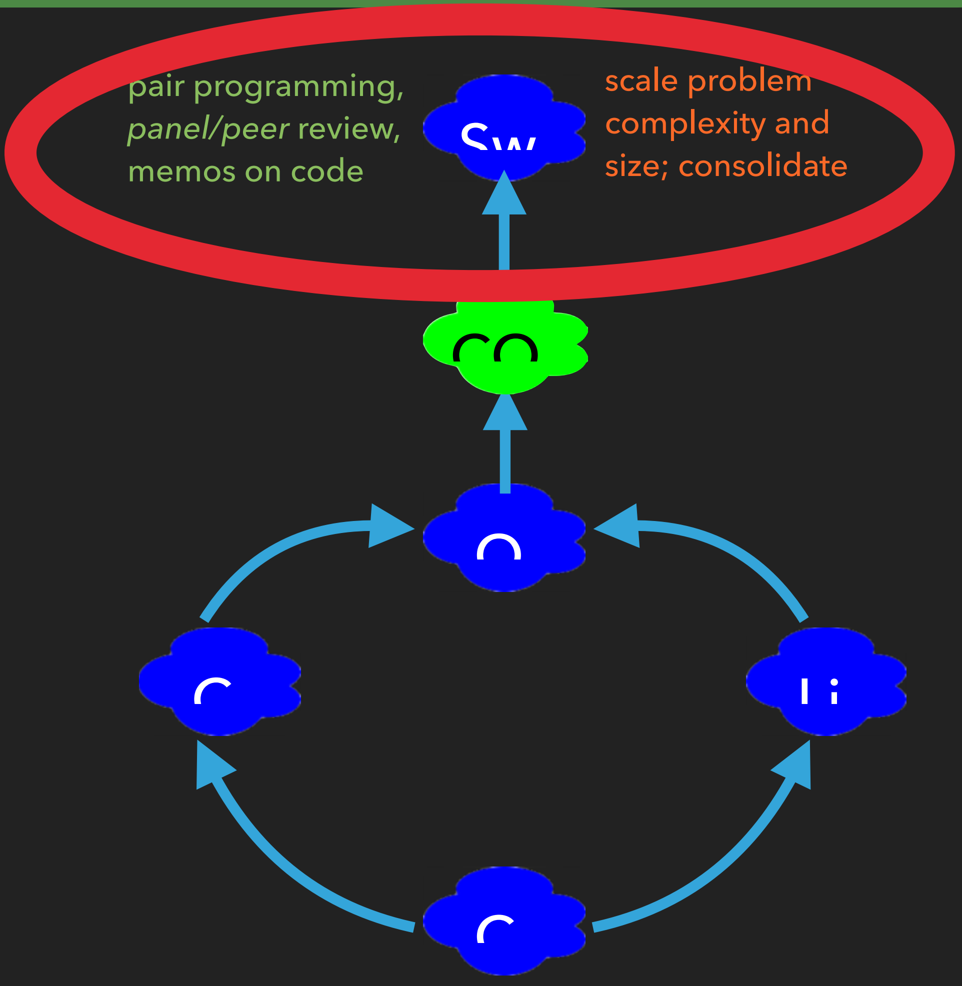
COOP 3

SPECIALIZATIONS: AI, BIG DATA, SYSTEMS, PL, ...

junior & senior years

COOP 2

``middler" year



sophomore year

freshman year

The Situation

The Goal

Learn to produce software for, judge it by,

- its design organization,
- its clarity in ideas, and
- its testability.

Do *not* judge it by its functionality.

The Outline

13 weekly assignments on sw dev ideas

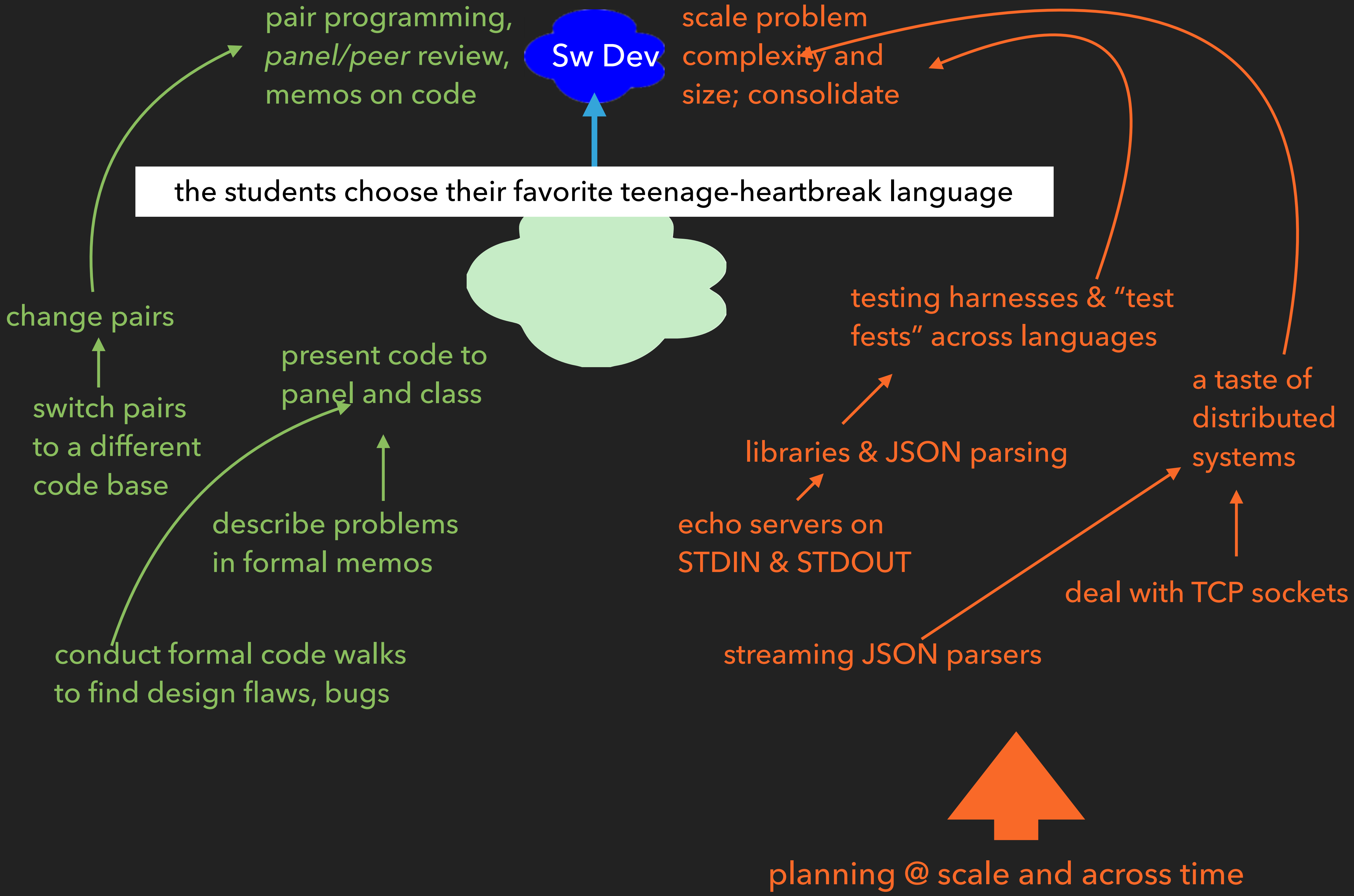
10 we walks

USE BOARD GAME BUT MAKE SURE TO DISCOUNT THE RESULTS OF ANY COMPETITION.
IT'S ABOUT SW DEV NOT AI DEV.

- Your favorite programming language
- Living up to interfaces
- Development includes maintenance
- From interfaces to protocols
- Incremental refinement, step 2
- Incremental refinement, step 3
- Changing an API

- bility
- GUIs
- Refactoring
- Designing your own protocol
- Integration time
- Remote proxying
- Strategy [optional]

SOFTWARE DEVELOPMENT, THE COURSE



Sw Dev

the students choose their favorite teenage-heartbreak language

pair programming, *panel/peer* review, memos on code

change pairs

switch pairs to a different code base

present code to panel and class

describe problems in formal memos

conduct formal code walks to find design flaws, bugs

scale problem complexity and size; consolidate

testing harnesses & "test fests" across languages

libraries & JSON parsing

echo servers on STDIN & STDOUT

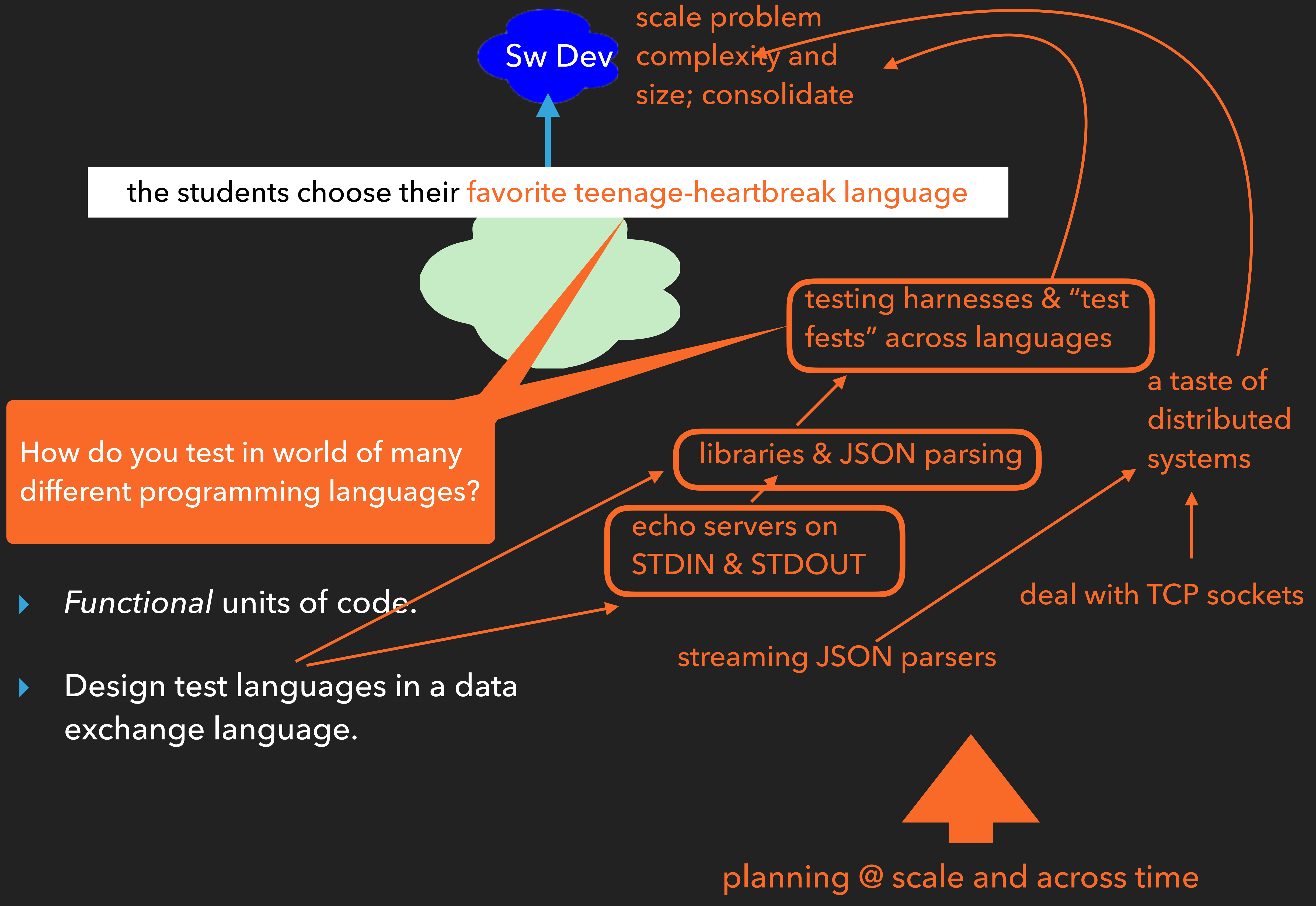
a taste of distributed systems

deal with TCP sockets

streaming JSON parsers

planning @ scale and across time

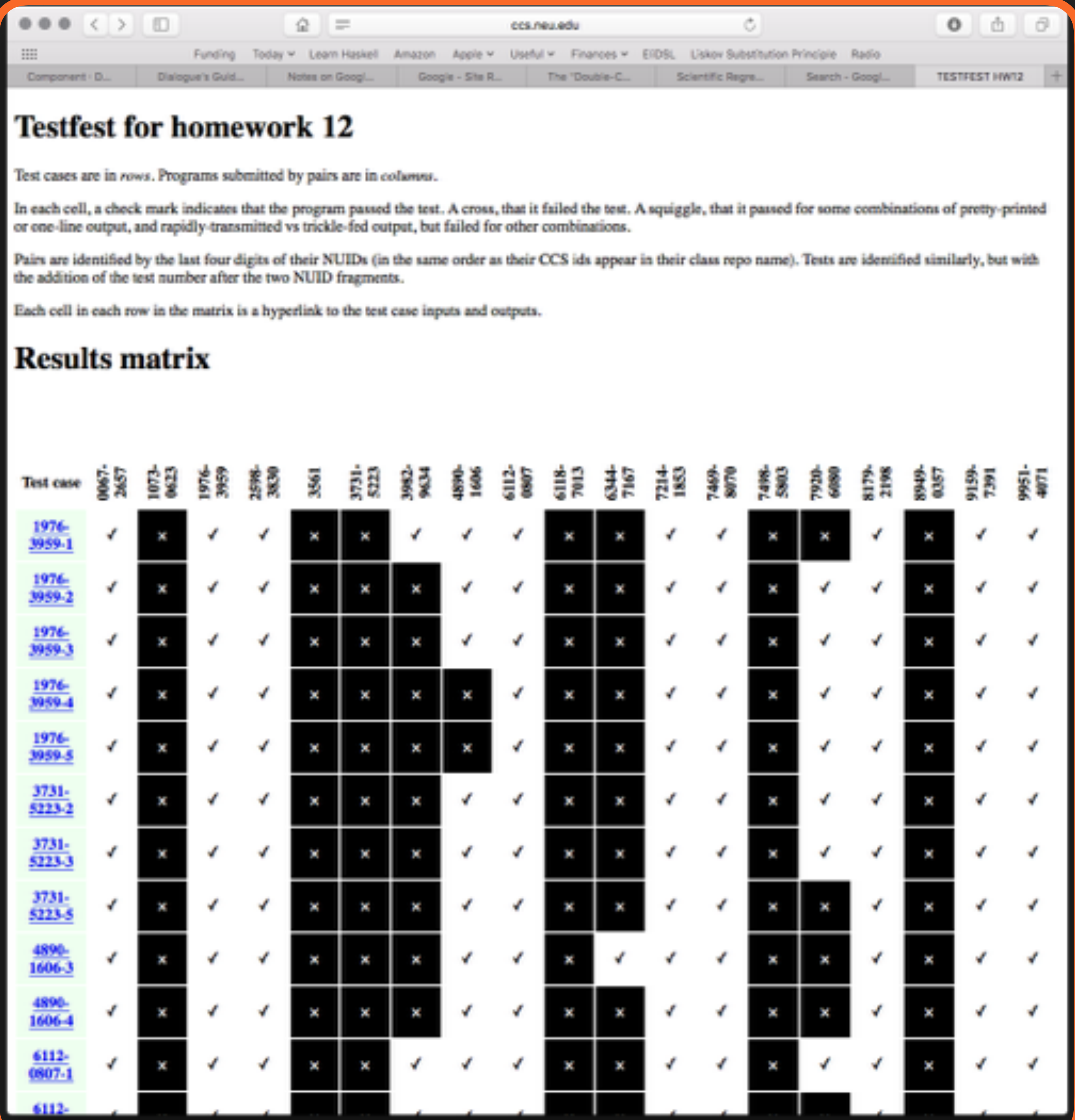
SOFTWARE DEVELOPMENT, THE COURSE


Sw Dev
scale problem complexity and size; consolidate
the students choose their favorite teenage-heartbreak language
testing harnesses & "test fests" across languages
a taste of distributed systems
libraries & JSON parsing
echo servers on STDIN & STDOUT
deal with TCP sockets
streaming JSON parsers
planning @ scale and across time

How do you test in world of many different programming languages?

- *Functional* units of code.
- Design test languages in a data exchange language.

- test fests, running everyone’s tests against everyone’s code.



Testfest for homework 12

Test cases are in *rows*. Programs submitted by pairs are in *columns*.

In each cell, a check mark indicates that the program passed the test. A cross, that it failed the test. A squiggle, that it passed for some combinations of pretty-printed or one-line output, and rapidly-transmitted vs trickle-fed output, but failed for other combinations.

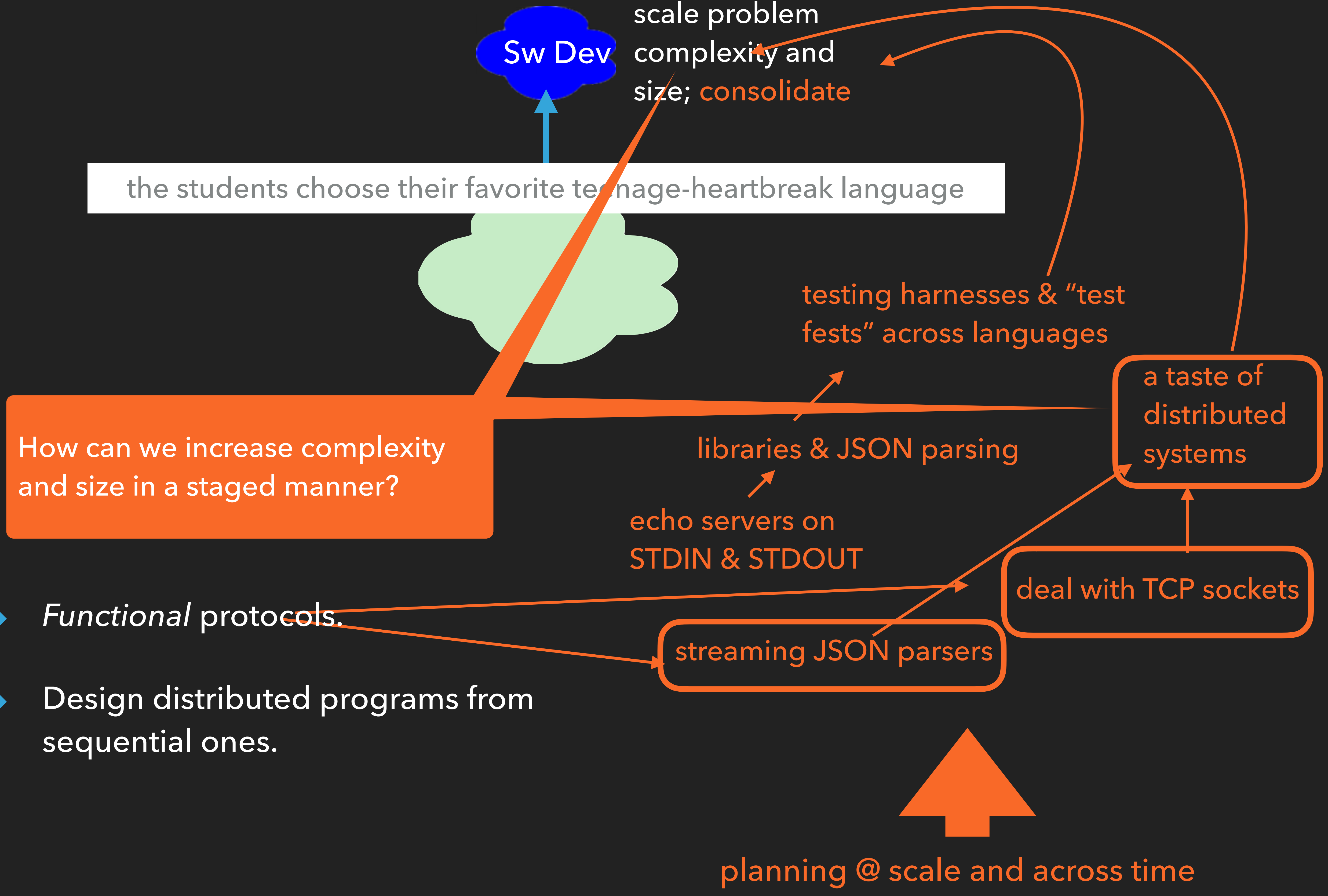
Pairs are identified by the last four digits of their NUIDs (in the same order as their CCS ids appear in their class repo name). Tests are identified similarly, but with the addition of the test number after the two NUID fragments.

Each cell in each row in the matrix is a hyperlink to the test case inputs and outputs.

Results matrix

Test case	0067-2657	1073-0623	1976-3959	2598-3830	3561	3731-5223	3982-9634	4890-1606	6112-0807	6118-7013	6344-7167	7214-1853	7469-8070	7498-5803	7920-6080	8179-2198	8949-0357	9159-7391	9951-4071
1976-3959-1	✓	×	✓	✓	×	×	✓	✓	✓	×	×	✓	✓	×	×	✓	×	✓	✓
1976-3959-2	✓	×	✓	✓	×	×	×	✓	✓	×	×	✓	✓	×	✓	✓	×	✓	✓
1976-3959-3	✓	×	✓	✓	×	×	×	✓	✓	×	×	✓	✓	×	✓	✓	×	✓	✓
1976-3959-4	✓	×	✓	✓	×	×	×	×	✓	×	×	✓	✓	×	✓	✓	×	✓	✓
1976-3959-5	✓	×	✓	✓	×	×	×	×	✓	×	×	✓	✓	×	✓	✓	×	✓	✓
3731-5223-2	✓	×	✓	✓	×	×	×	✓	✓	×	×	✓	✓	×	✓	✓	×	✓	✓
3731-5223-3	✓	×	✓	✓	×	×	×	✓	✓	×	×	✓	✓	×	✓	✓	×	✓	✓
3731-5223-5	✓	×	✓	✓	×	×	×	✓	✓	×	×	✓	✓	×	×	✓	×	✓	✓
4890-1606-3	✓	×	✓	✓	×	×	×	✓	✓	×	✓	✓	✓	×	×	✓	×	✓	✓
4890-1606-4	✓	×	✓	✓	×	×	×	✓	✓	×	×	✓	✓	×	×	✓	×	✓	✓
6112-0807-1	✓	×	✓	✓	×	×	✓	✓	✓	×	×	✓	✓	×	✓	✓	×	✓	✓
6112-																			

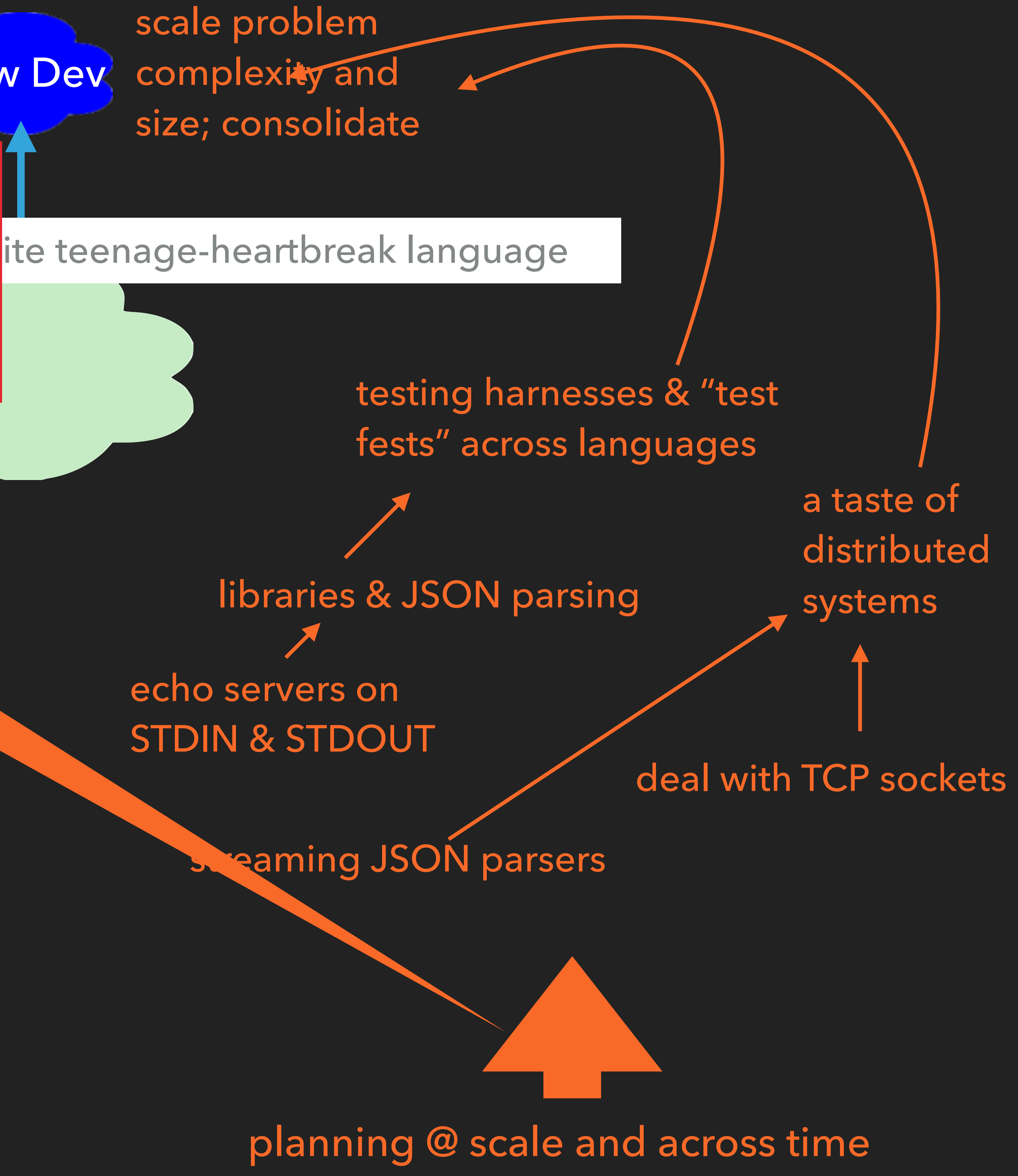
SOFTWARE DEVELOPMENT, THE COURSE



Sw Dev
scale problem complexity and size; consolidate
the students choose their favorite teenage-heartbreak language
How can we increase complexity and size in a staged manner?
testing harnesses & "test fests" across languages
libraries & JSON parsing
echo servers on STDIN & STDOUT
a taste of distributed systems
deal with TCP sockets
streaming JSON parsers
planning @ scale and across time

- *Functional* protocols.
- Design distributed programs from sequential ones.

NO, NOT WITH SUFFICIENT DETAIL.



Sw Dev

scale problem complexity and size; consolidate

ite teenage-heartbreak language

testing harnesses & "test fests" across languages

a taste of distributed systems

libraries & JSON parsing

echo servers on STDIN & STDOUT

deal with TCP sockets

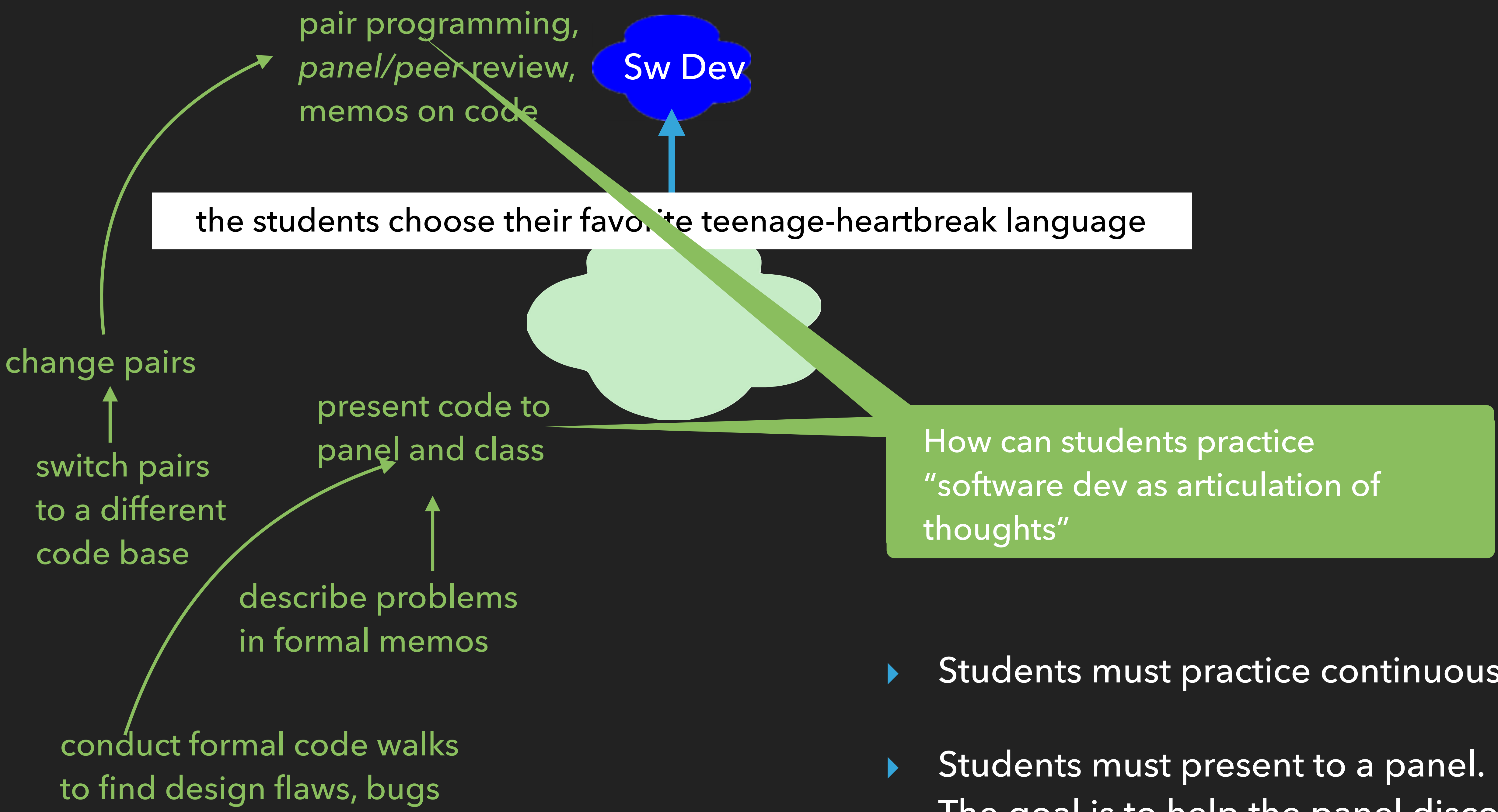
streaming JSON parsers

planning @ scale and across time

Can students figure out the architecture of such systems?

- Interfaces for Foobarmistan.
- Week-by-week training:
  - they design an interface.
  - then we use mine.

SOFTWARE DEVELOPMENT, THE COURSE



pair programming, *panel/peer* review, memos on code

Sw Dev

the students choose their favorite teenage-heartbreak language

change pairs

switch pairs to a different code base

present code to panel and class

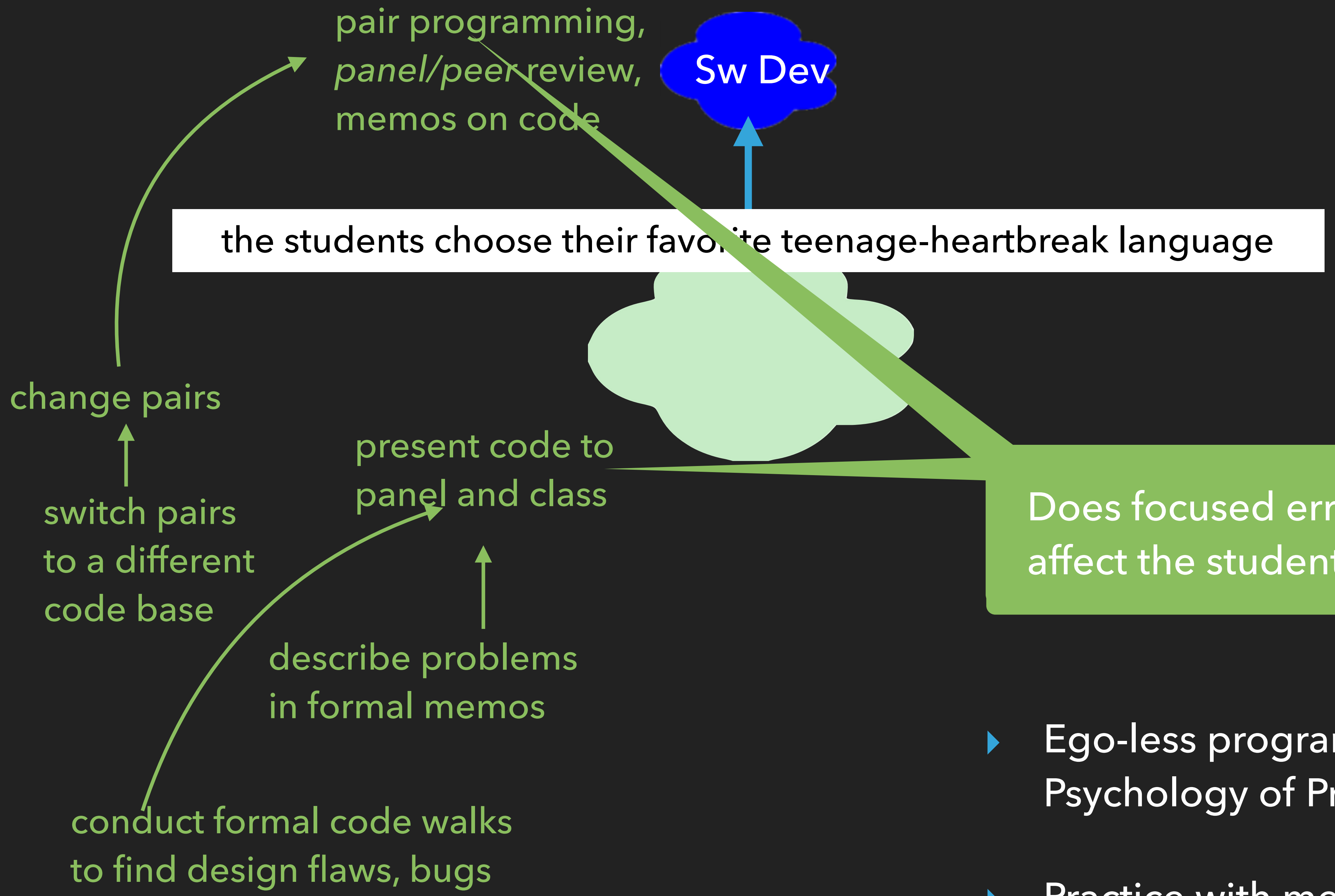
describe problems in formal memos

conduct formal code walks to find design flaws, bugs

How can students practice “software dev as articulation of thoughts”

- Students must practice continuously.
- Students must present to a panel. The goal is to help the panel discover errors in the devs’ thinking.

SOFTWARE DEVELOPMENT, THE COURSE



Sw Dev

the students choose their favorite teenage-heartbreak language

pair programming, *panel/peer* review, memos on code

change pairs

switch pairs to a different code base

present code to panel and class

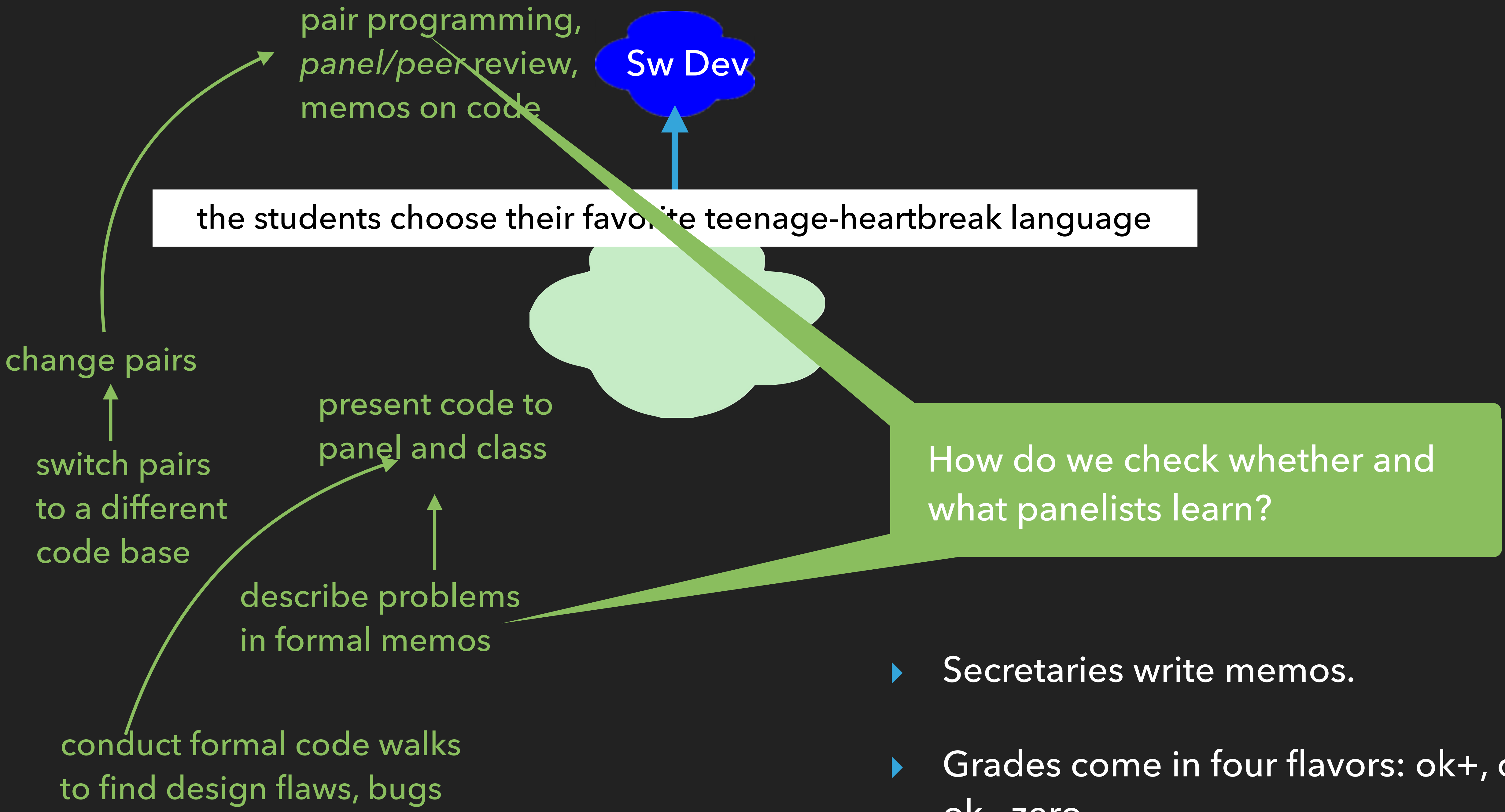
describe problems in formal memos

conduct formal code walks to find design flaws, bugs

Does focused error discovery affect the students' psyche?

- Ego-less programming. Weinberger, Psychology of Programming, 1974.
- Practice with melt-downs.

SOFTWARE DEVELOPMENT, THE COURSE



pair programming, *panel/peer* review, memos on code

Sw Dev

the students choose their favorite teenage-heartbreak language

change pairs

switch pairs to a different code base

present code to panel and class

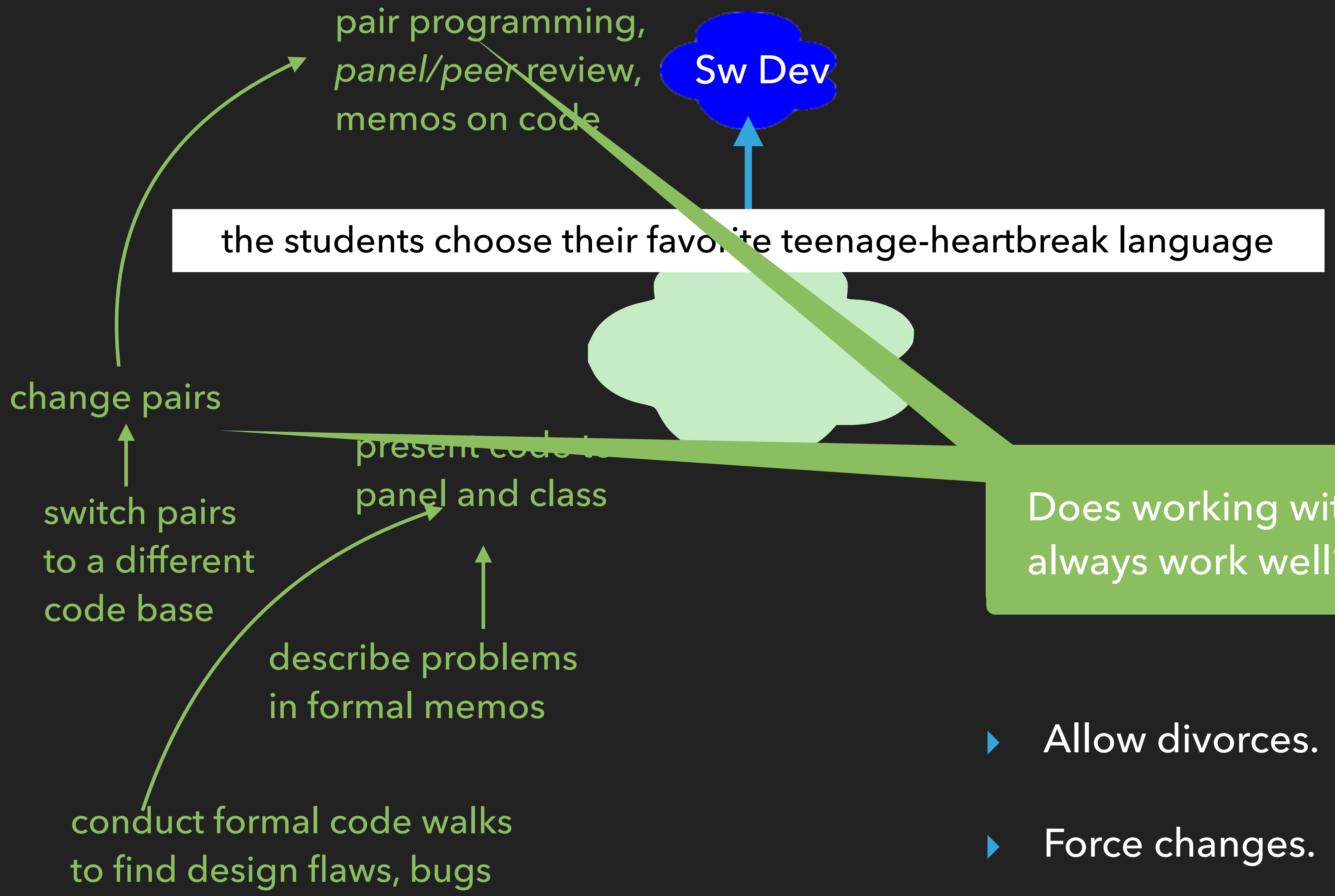
describe problems in formal memos

conduct formal code walks to find design flaws, bugs

How do we check whether and what panelists learn?

- Secretaries write memos.
- Grades come in four flavors: ok+, ok, ok-, zero.
- Students vote what the grades mean.

SOFTWARE DEVELOPMENT, THE COURSE



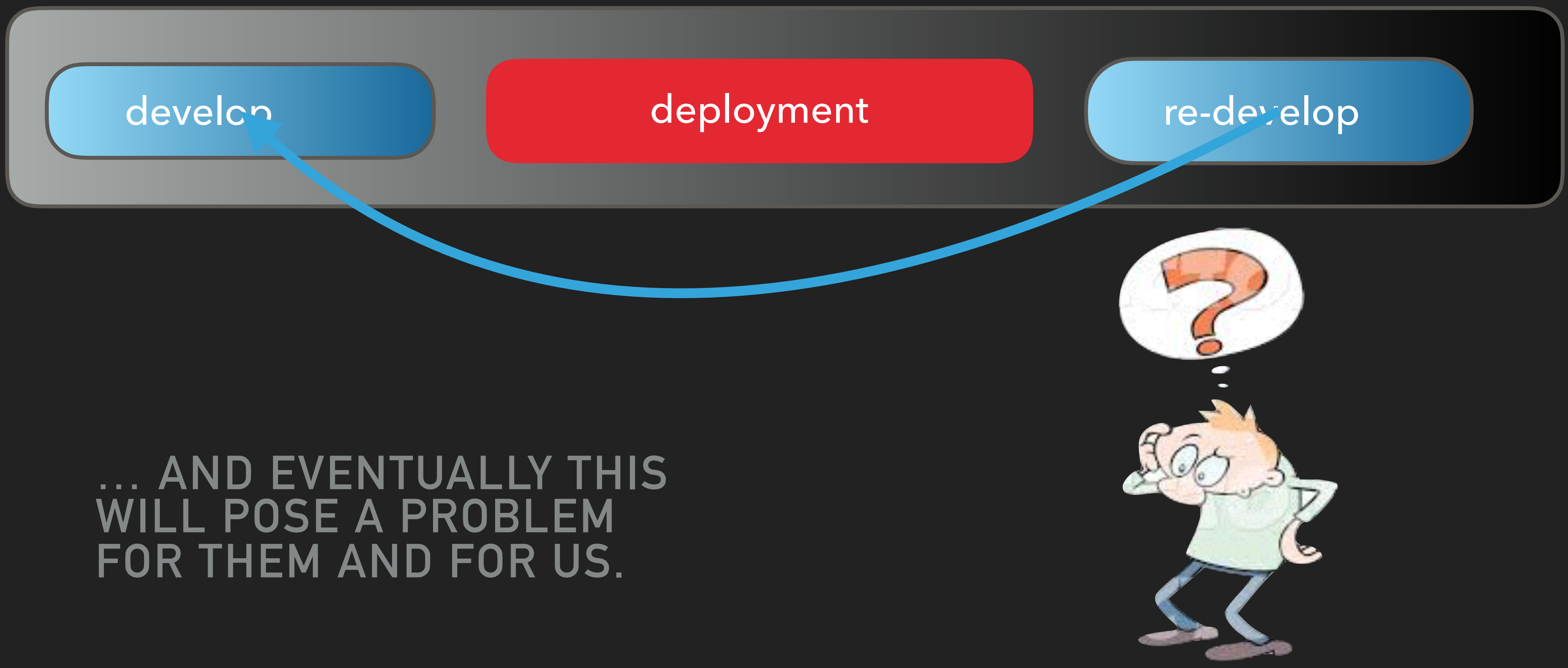
Does working with partners always work well?

- Allow divorces.
- Force changes.
- Let students vote on “choice of partner” or “choice of code base.”

TAKE AWAY

TAKE AWAY

DEVELOPMENT COST IS HIGH FOR DEVELOPERS AND EMPLOYERS

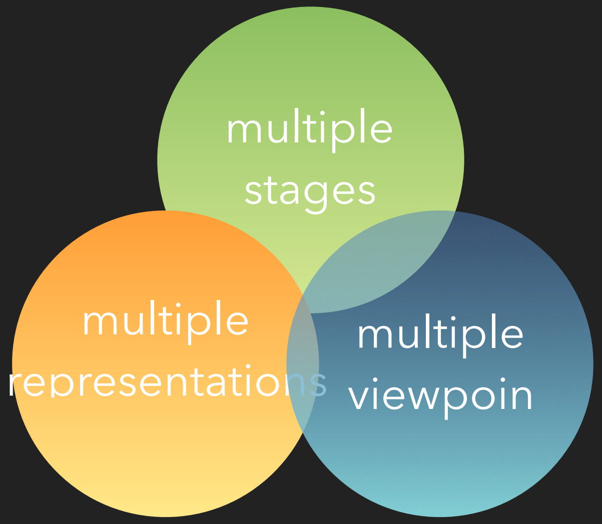


... AND EVENTUALLY THIS WILL POSE A PROBLEM FOR THEM AND FOR US.

TAKE AWAY

STUDENTS NEED TECHNICAL DESIGN SKILLS,

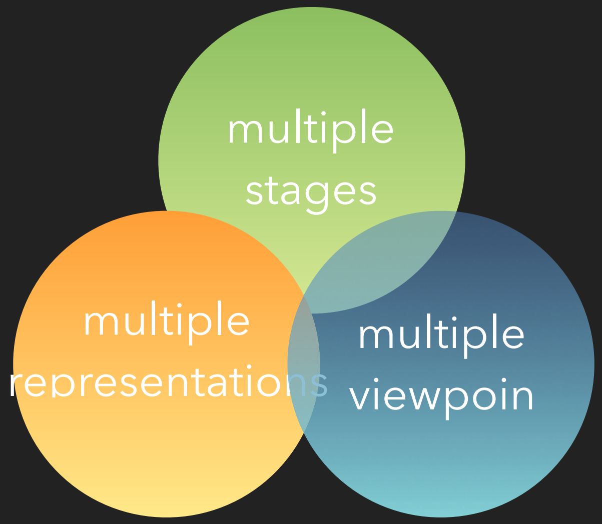
- Teach systematic design explicitly.
- Teach it in several courses.
- Teach it at increasingly large scales.
- Teach it in different languages & contexts.
- Teach it until it becomes second nature.



TAKE AWAY

STUDENTS NEED TECHNICAL COMMUNICATION SKILLS.

- Teach programming as communication of thoughts.
- Teach it in several courses.
- Teach it in different contexts.
- Teach it in for pairs and in class.
- Teach it until it becomes second nature.



multiple stages

multiple representations

multiple viewpoin

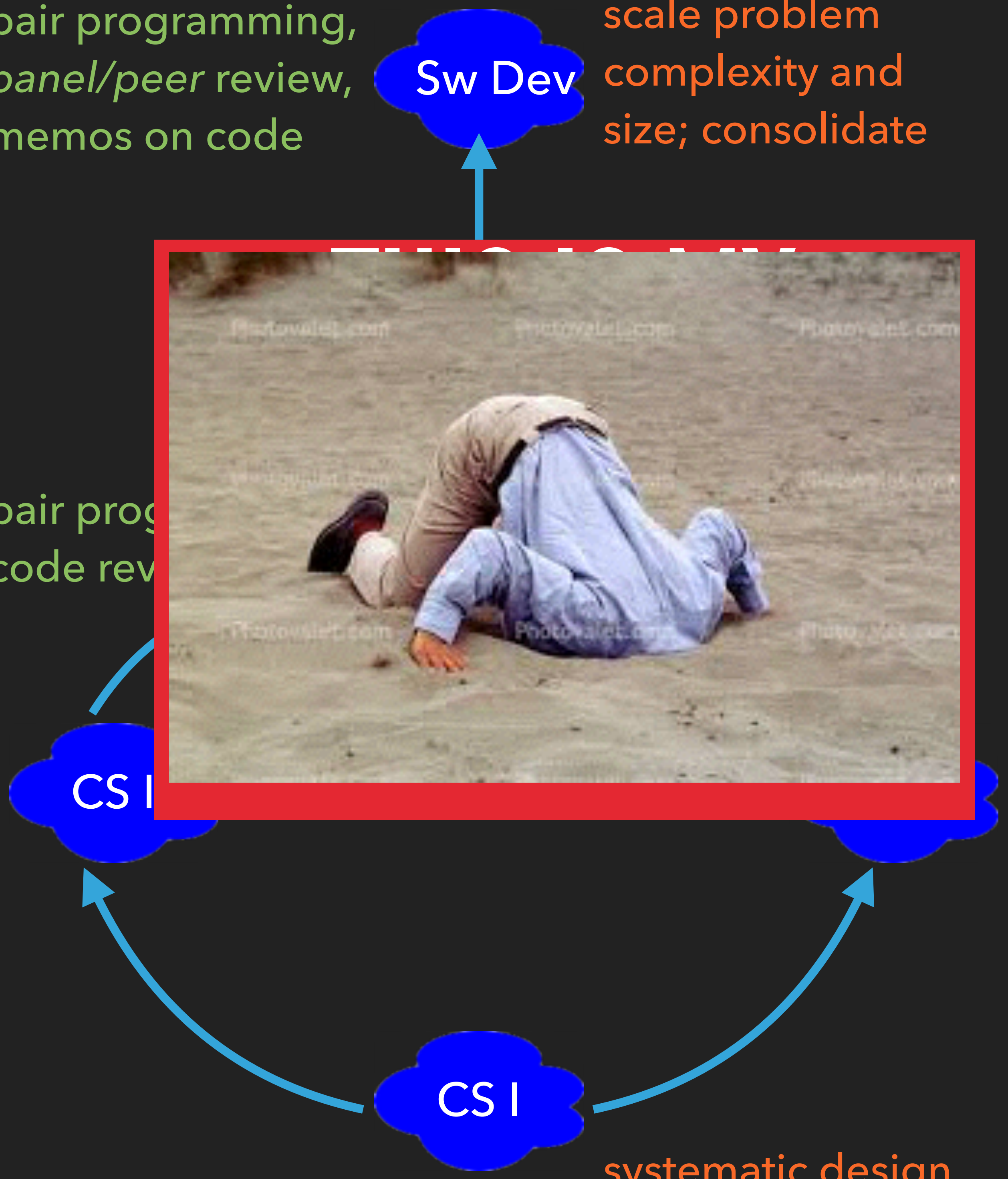
TAKE AWAY

Your students and their employers will appreciate these skills in time.

HOW?

pair programming,
panel/peer review,
memos on code

scale problem
complexity and
size; consolidate



pair prog
code rev

systematic design,
typed & OOPL (Java)

pair programming,
code review

thinking about code,
dual to systematic design

pair programming

pair programming

systematic design,
“student languages”

THE END

- Robby Findler, for co-creating "Hell" and pointing me in the right direction
- Shriram Krishnamurthi and Kathi Fisler, for many exchanges on design and planning
- Matthew Flatt, for teaching me the value of rapid feedback in design
- .. and many others for discussions and push-back and telling me how wrong I was and often am

QUESTIONS?