THE FIRST YEAR

Matthias Felleisen
PLT
Northeastern University, Boston

Where it all began

- Starting in 1995 (January 26) @ Rice University
- The "How to Design" Project
  - TeachScheme!
  - DrScheme and ProfessorJ
- ... but in reality, it all began in 1978.

The Problem

PROGRAMMING 101

PROGRAMMING 101

- in 1978 @ Karlsruhe (Technische Universität): variables, assignments, printing, arrays, loops, procedures, classes and methods

PROGRAMMING 101

- in 1978 @ Karlsruhe (Technische Universität): variables, assignments, printing, arrays, loops, procedures, classes and methods
- in 2007 @ Anywhere (College, University): variables, assignments, printing, arrays, loops, procedures, classes and methods, ...

PROGRAMMING 101

- in 1978 @ Karlsruhe (Technische Universität): variables, assignments, printing, arrays, loops, procedures, classes and methods
- in 2007 @ Anywhere (College, University): variables, assignments, printing, arrays, loops, procedures, classes and methods, ...
- ... and perhaps interfaces and inheritance.

Programming 101

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

PROGRAMMING 101

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

1978 - 2007:
9 languages, 29 years:

PROGRAMMING 101

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

1978 - 2007:

9 languages, 29 years:

Are we really just a fashion industry?

PROGRAMMING 101

Is your story any better?

PROGRAMMING 101

- Why are we still thrashing around when it comes to the first year?
- Do we really lack a "product" to sell?
- Would a Physicist ask "how do I make the introductory course more ***Glamour***-ous?"

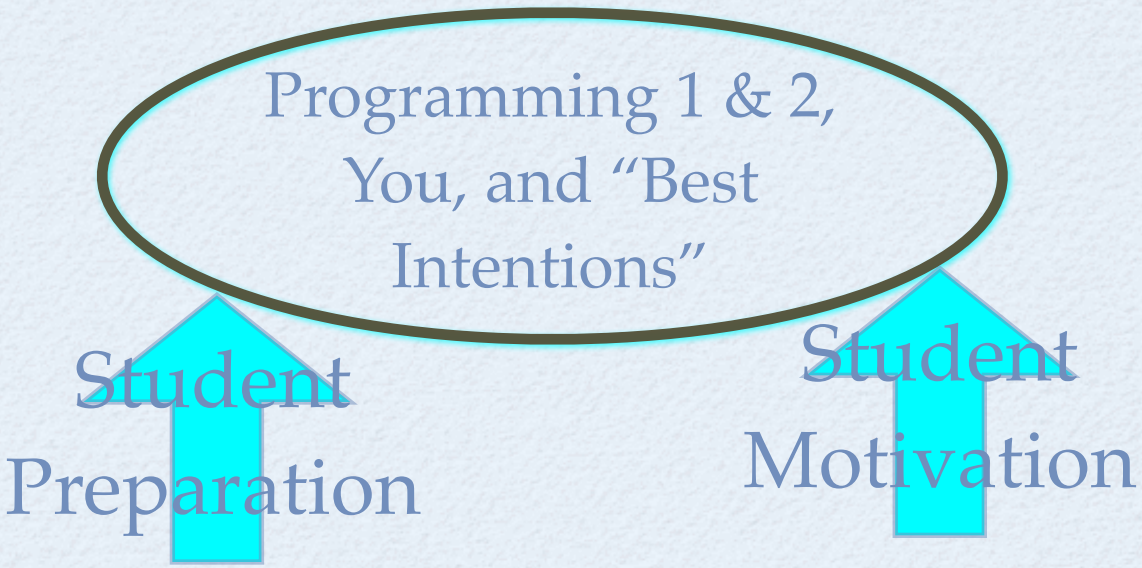
The Constraints

CONSTRAINTS

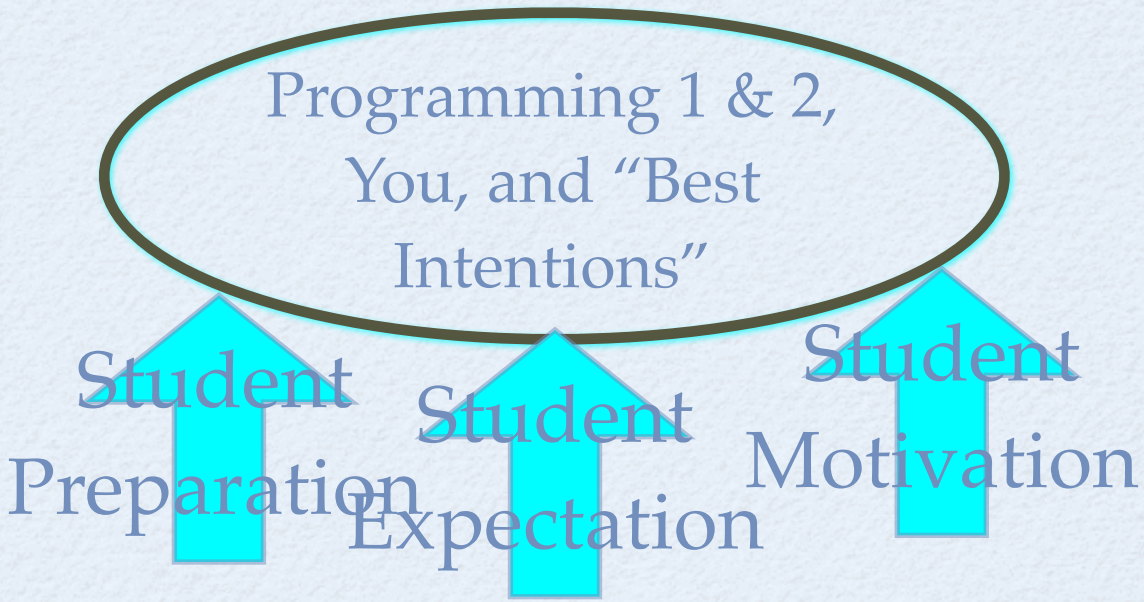
Programming 1 & 2,
You, and "Best
Intentions"

CONSTRAINTS

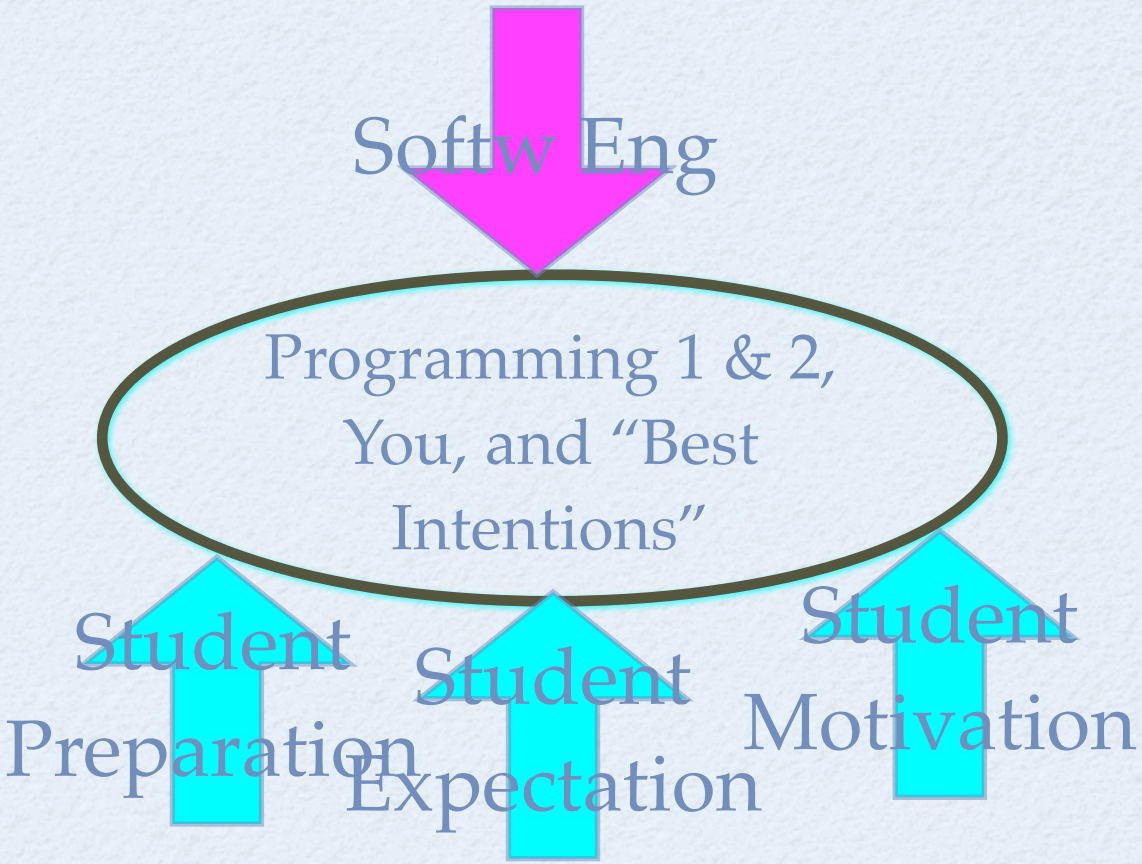
CONSTRAINTS



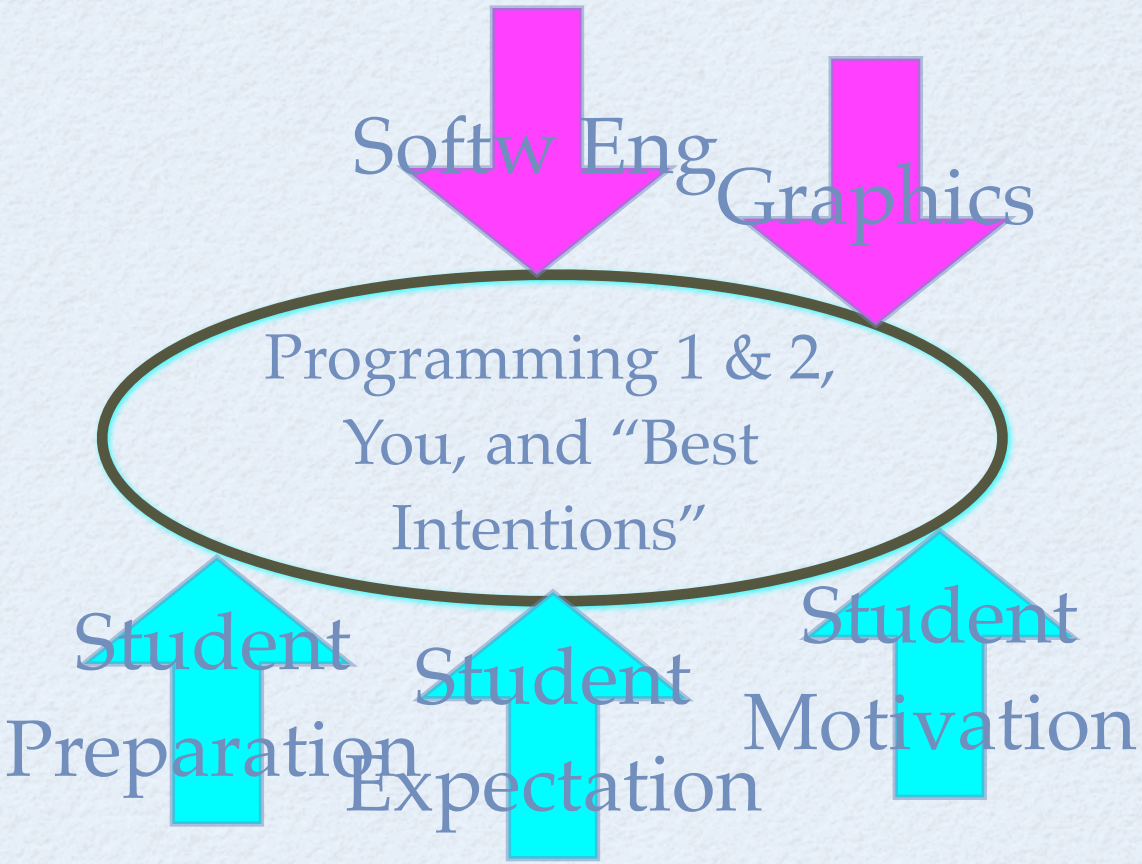
CONSTRAINTS



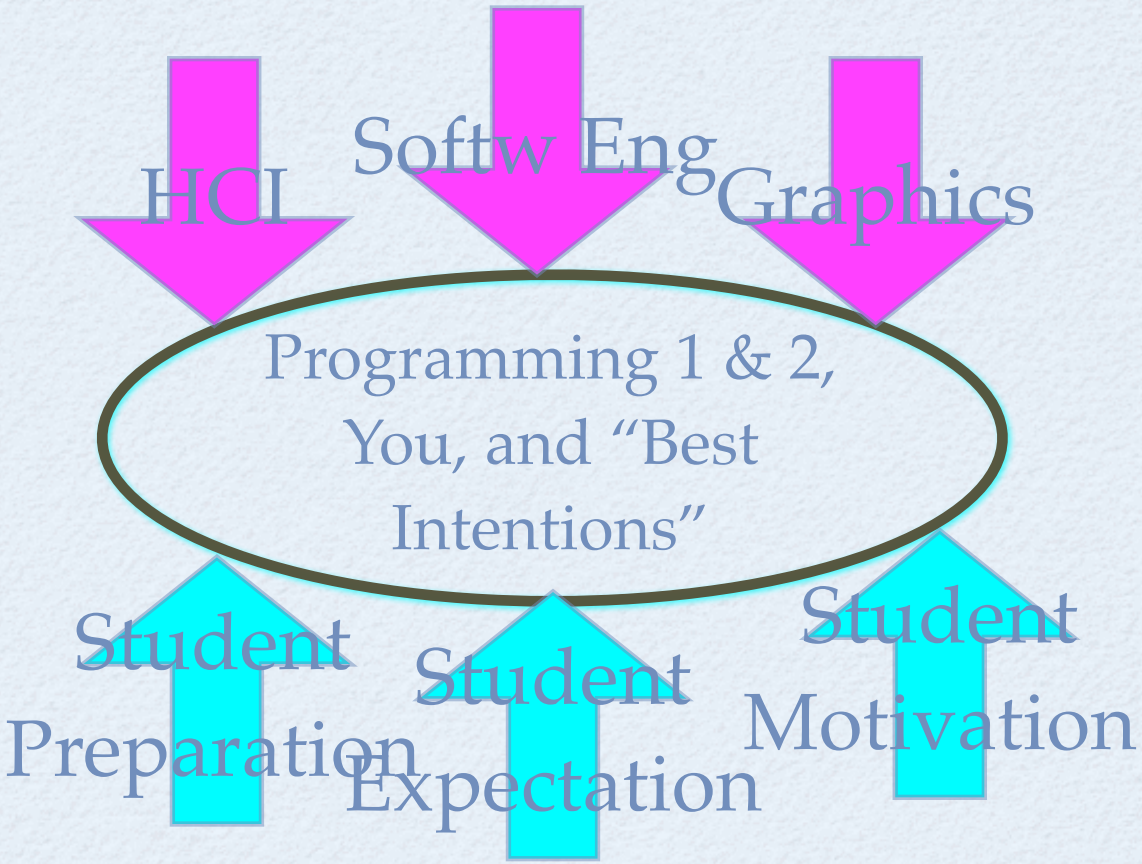
CONSTRAINTS



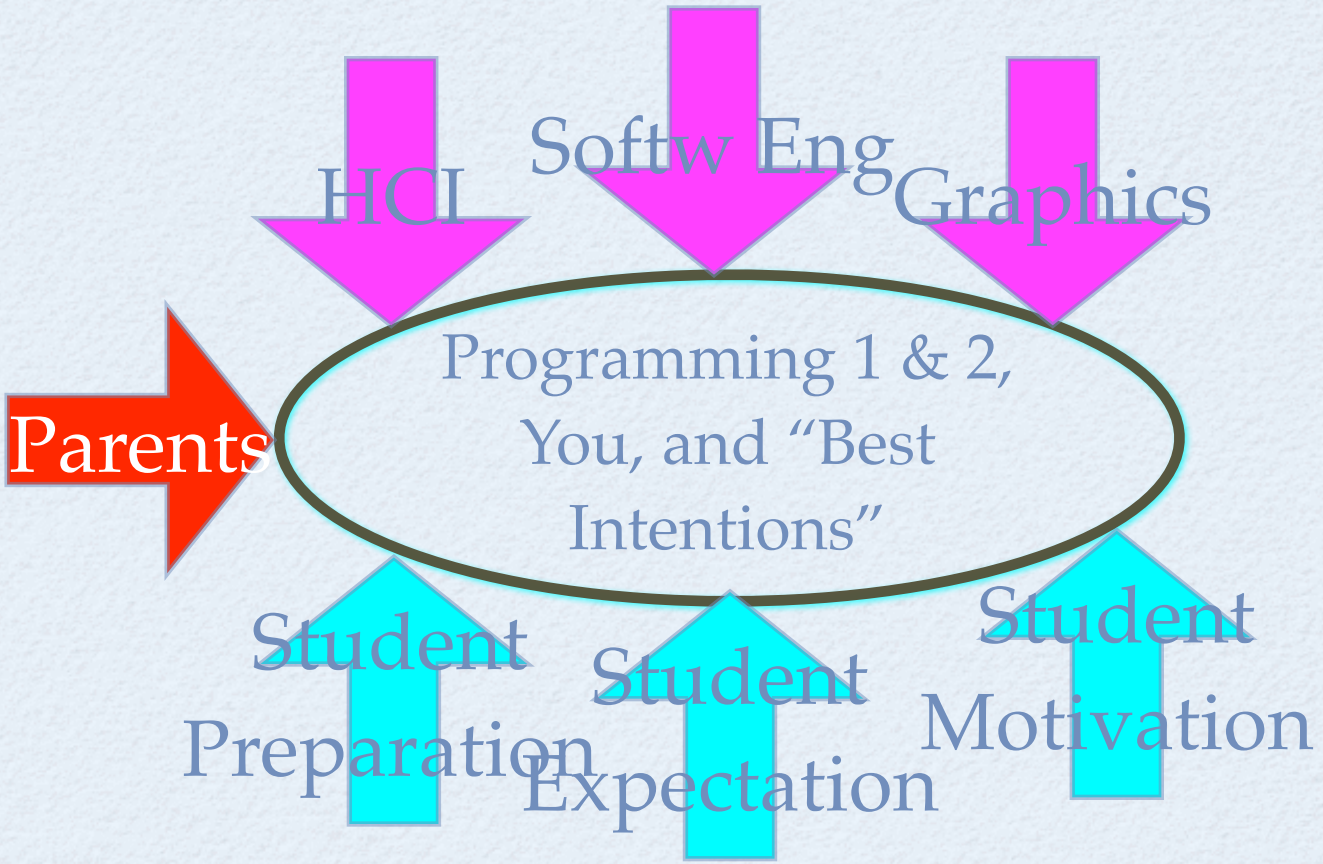
CONSTRAINTS



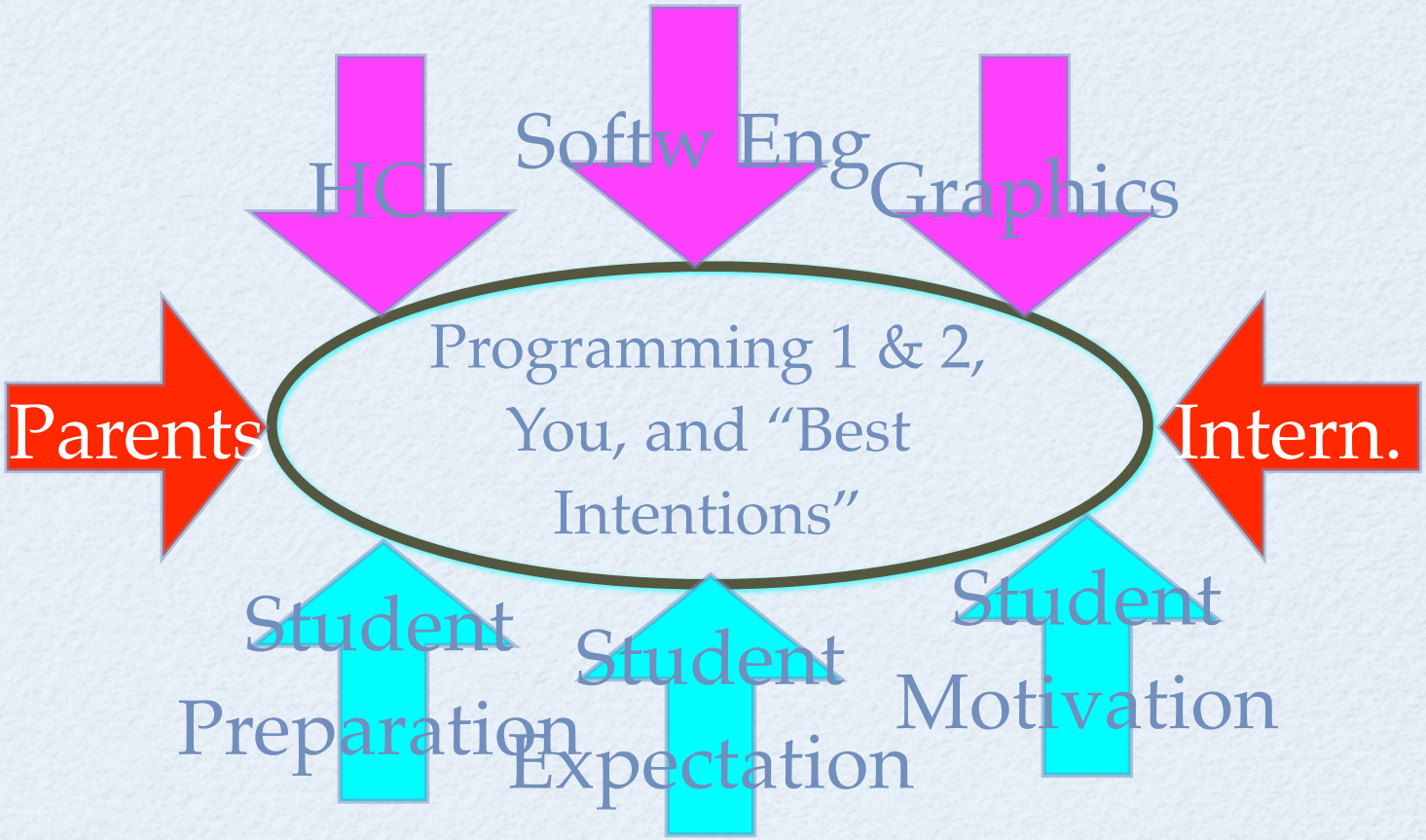
CONSTRAINTS



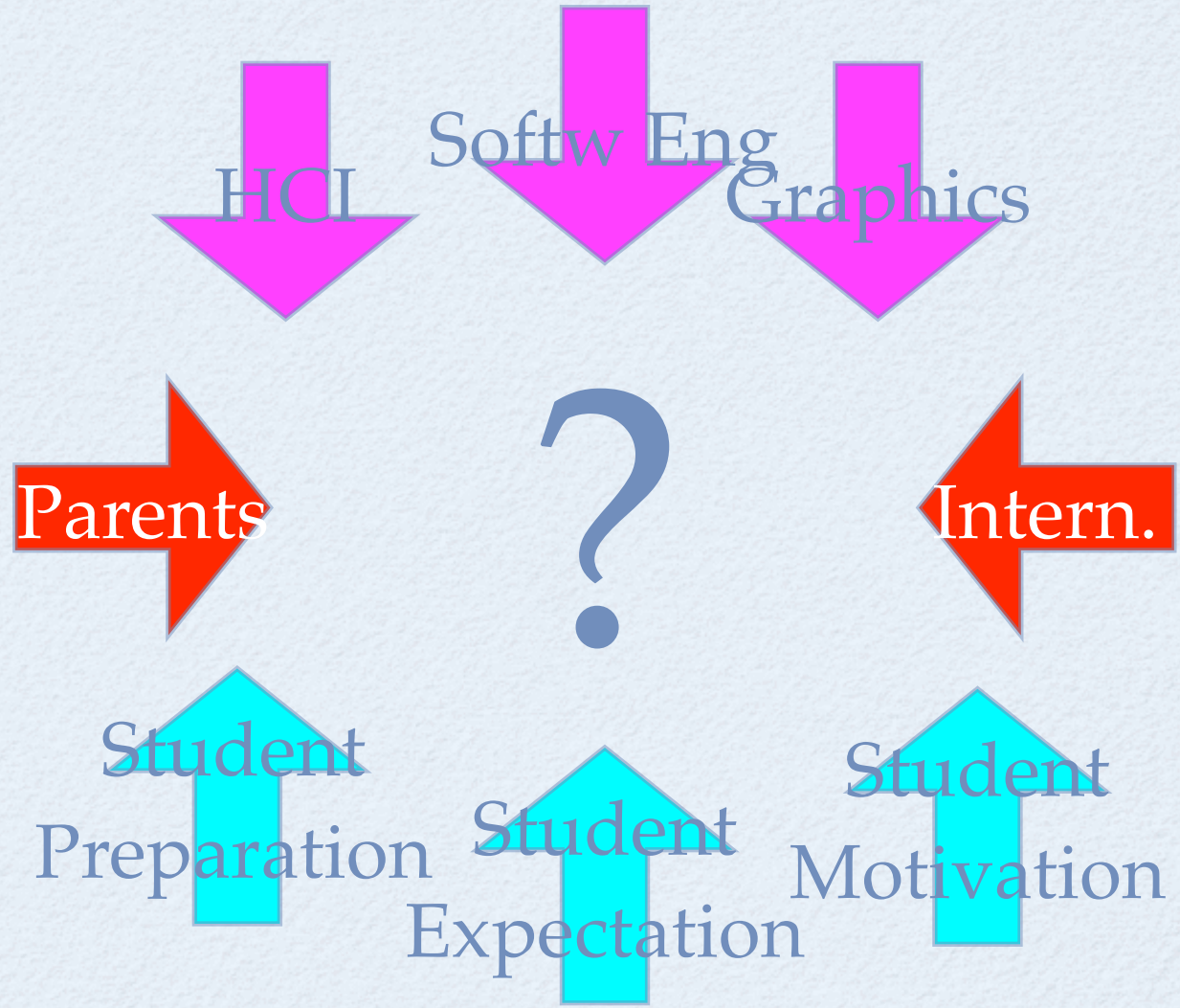
CONSTRAINTS



CONSTRAINTS



CONSTRAINTS

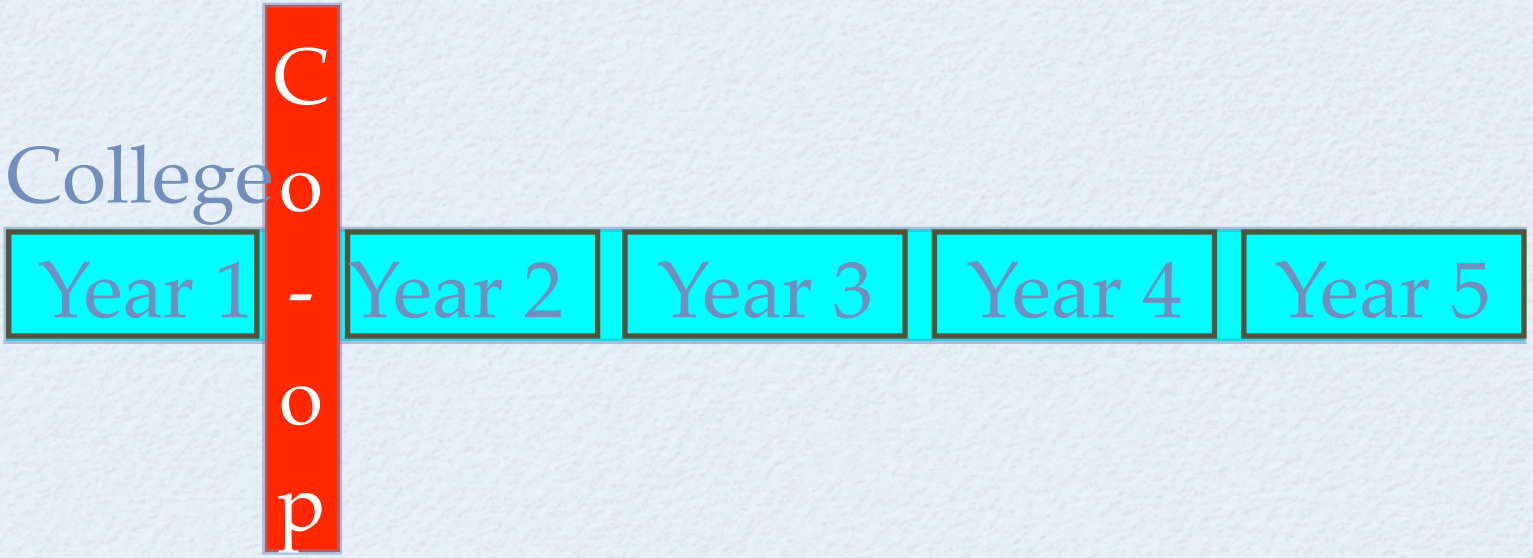


THE COLLEGE TIMELINE

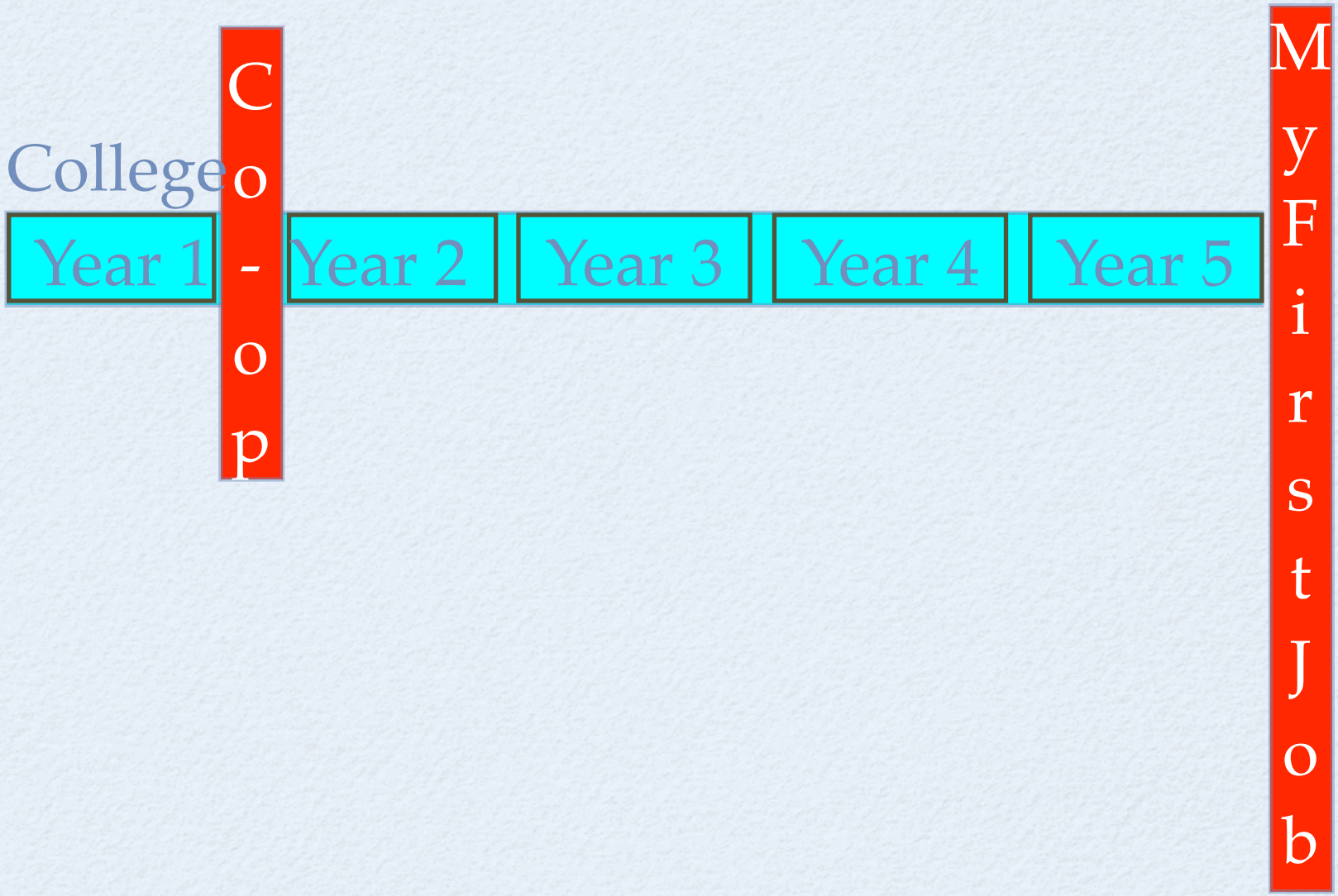
College

Year 1	Year 2	Year 3	Year 4	Year 5

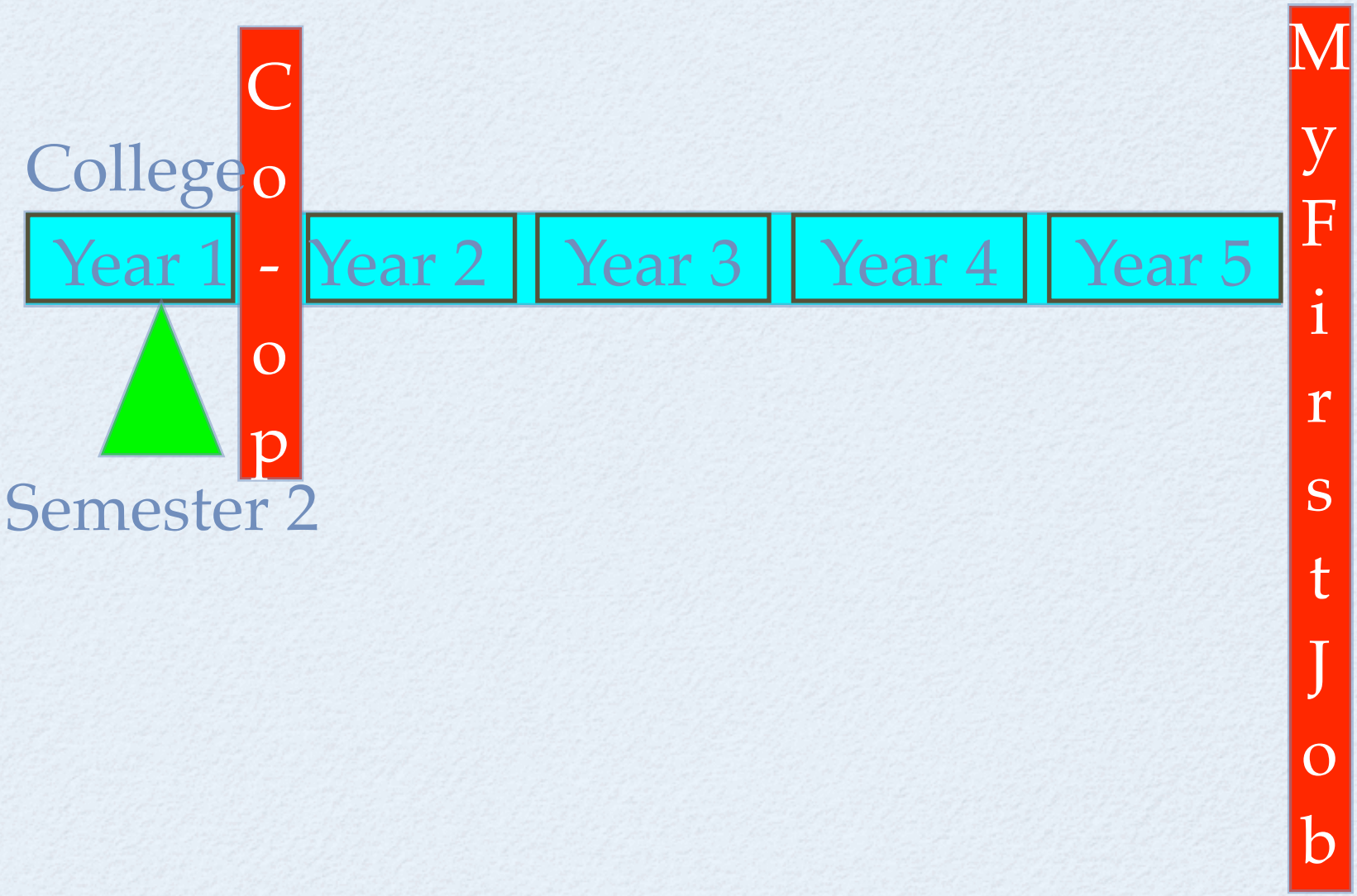
THE COLLEGE TIMELINE



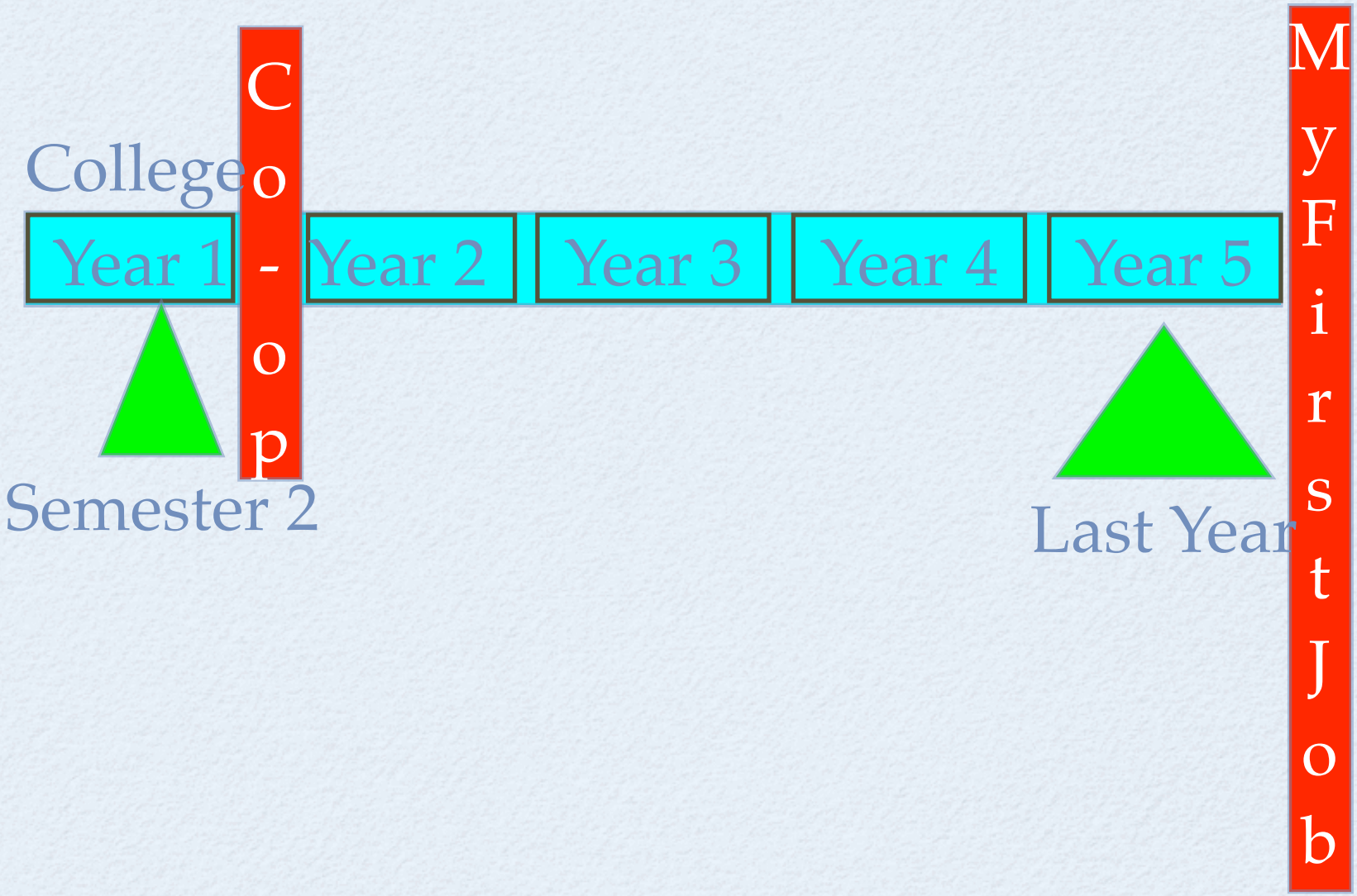
THE COLLEGE TIMELINE



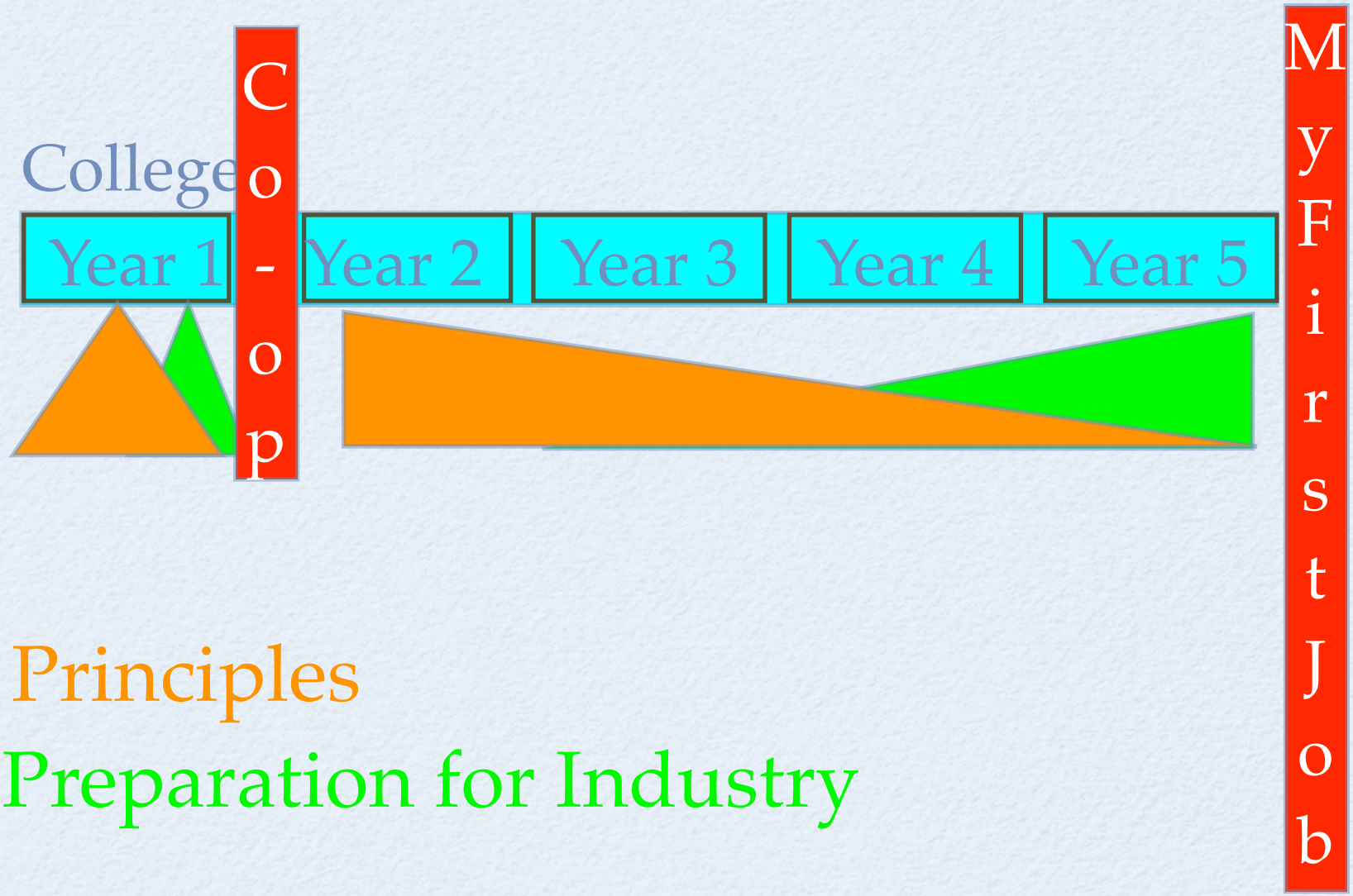
THE COLLEGE TIMELINE



THE COLLEGE TIMELINE



THE COLLEGE TIMELINE



The Principles

COLLEGE

COLLEGE

- College empowers ***life-long learning***.

COLLEGE

- College empowers ***life-long learning***.
- Students need ***early exposure to best-practices***.

COLLEGE

- College empowers ***life-long learning***.
- Students need ***early exposure to best-practices***.
- Faculty ***aims high and has spine*** to stick to principles.

PROGRAMMING

PROGRAMMING

- Best practices means "***get it right.***"

PROGRAMMING

- Best practices means "***get it right.***"
- Best practices means "***think it through.***"

PROGRAMMING

- Best practices means *"**get it right.**"*
- Best practices means *"**think it through.**"*
- Best practices means *"**to program is to design.**"*

PROGRAM. LANGUAGE

PROGRAM. LANGUAGE

- Object-oriented programming has won.

PROGRAM. LANGUAGE

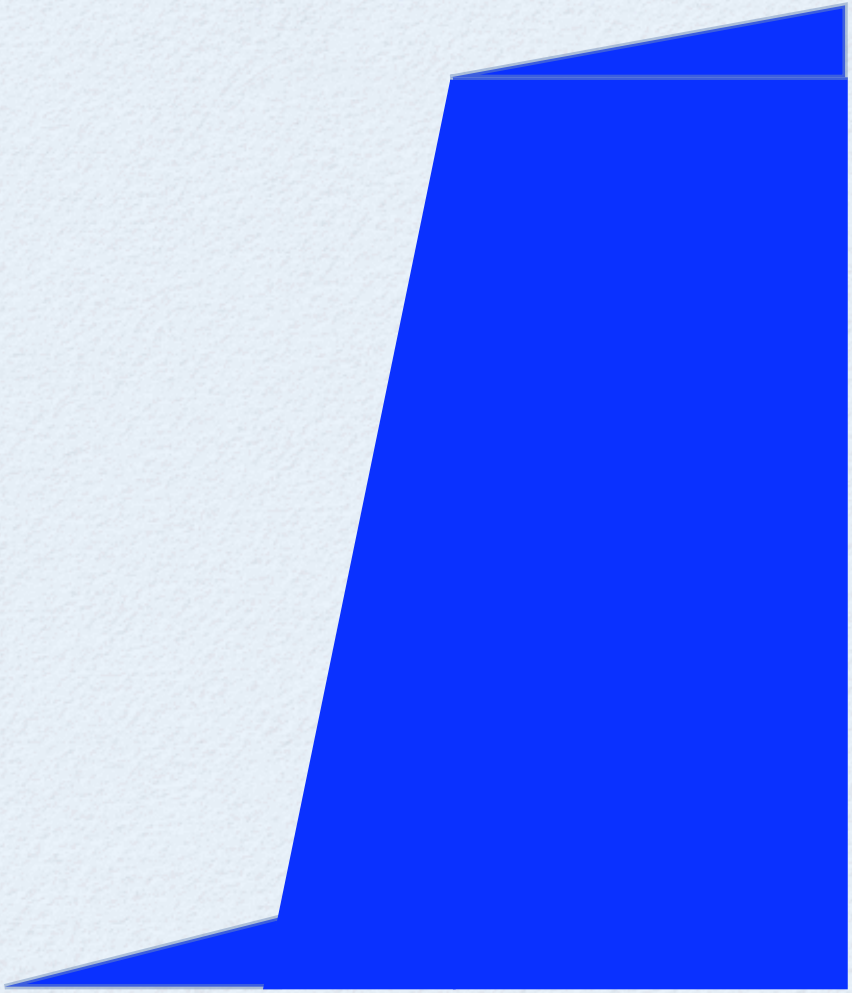
- Object-oriented programming has won.
- So has scripting (light-weight programming).

PROGRAM. LANGUAGE

- Object-oriented programming has won.
- So has scripting (light-weight programming).
- The quality of programs depends on how often the programmer has switched languages and programming paradigms. It does ***not*** depend on the numbers of years spent with ***one*** language [PPig].

THE FIRST LANGUAGE

Current:

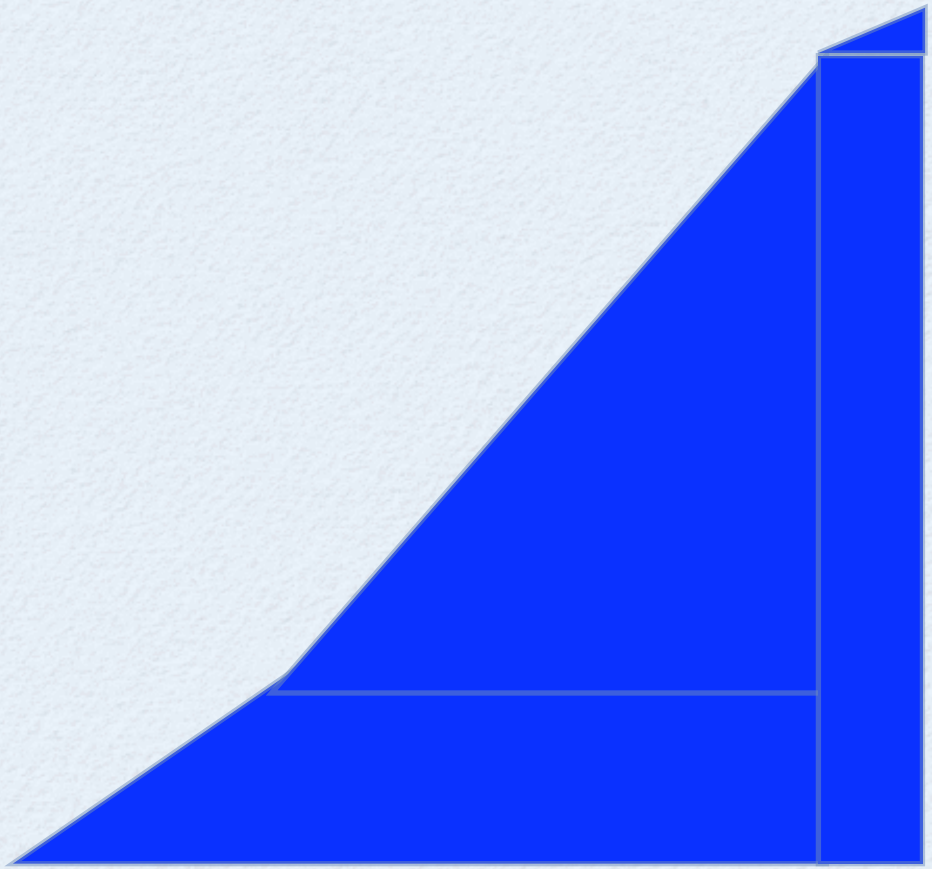


variables, assignments,
printing, arrays,
loops, procedures, ...

10th grade math

THE FIRST LANGUAGE

Wanted:



???

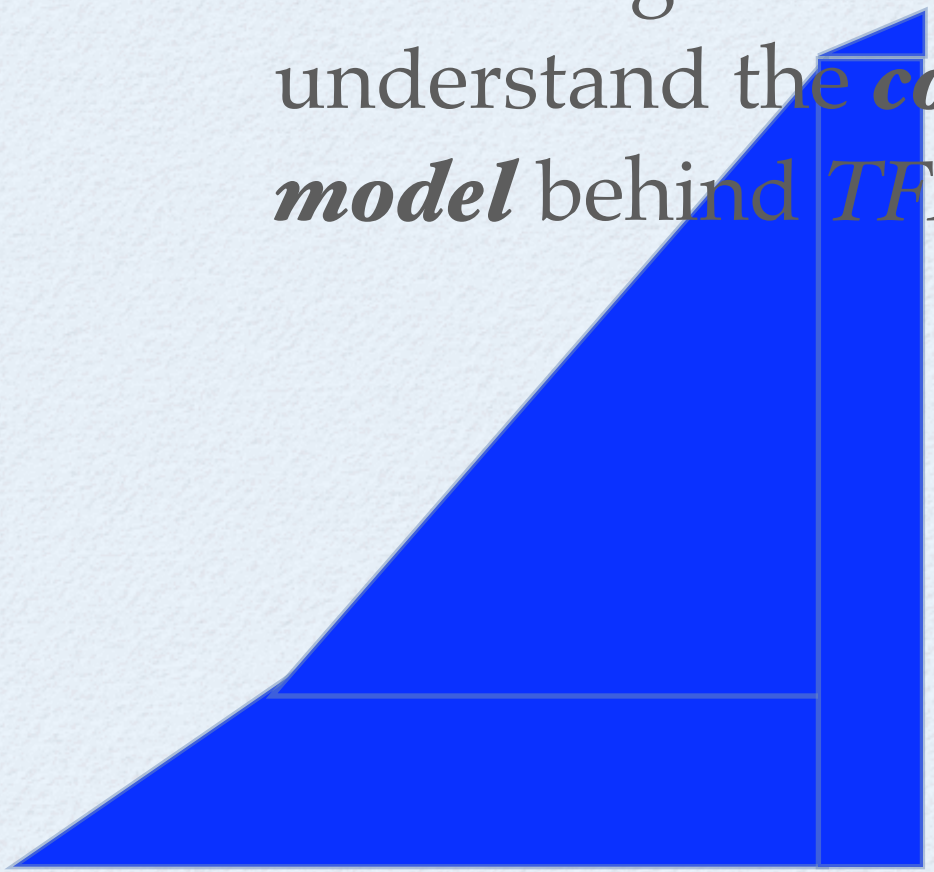
10th grade math

THE FIRST LANGUAGE

Wanted:

An tenth grader should naturally understand the ***computational model*** behind *TFL*.

???



10th grade math

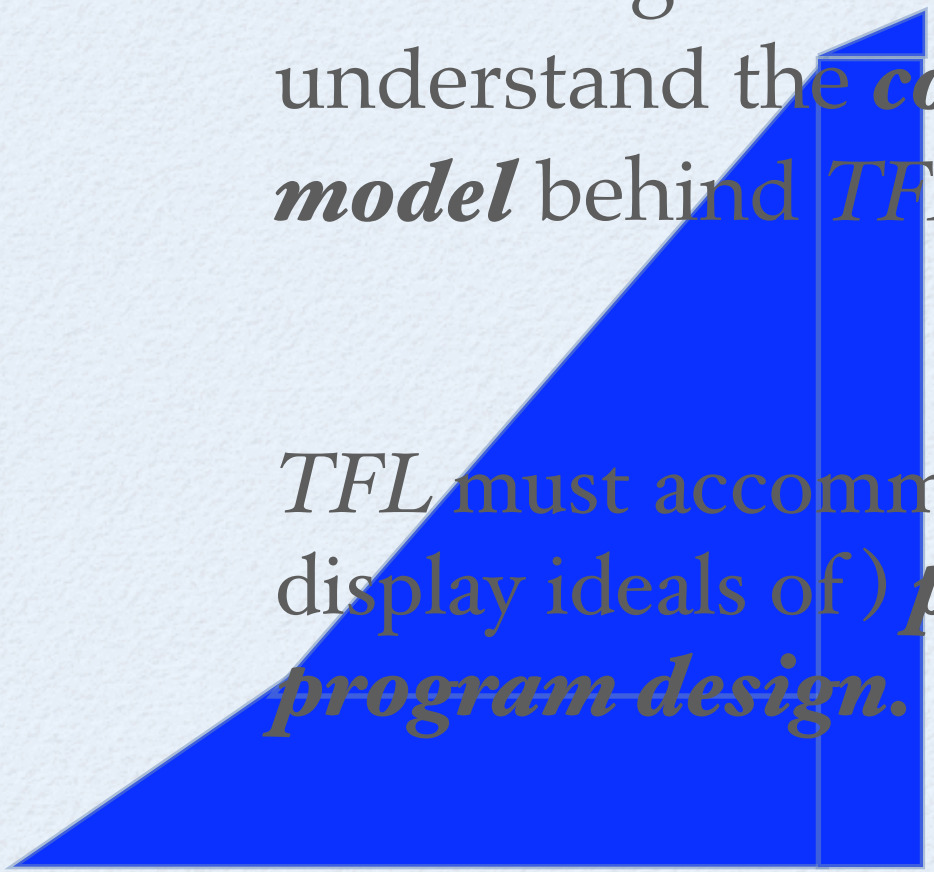
THE FIRST LANGUAGE

Wanted:

An tenth grader should naturally understand the ***computational model*** behind *TFL*.

???

TFL must accommodate (and display ideals of) ***principled program design.***



10th grade math

The Second Language

- The second language must prepare students for *co-op* and should be *fashionable*.
- Use the second language to demonstrate that principles work *everywhere*.

PRINCIPLES & CONSTRAINTS

principles:
- programming
- computation
- easy!

principles for OO
- programming
- computation
- relevant

Linguistics I

THE FIRST LANGUAGE

THE FIRST LANGUAGE

Sather

Eiffel

Java

C#

C++

VB

C

Basic

imperative or OOP

THE FIRST LANGUAGE

JavaScript

Ruby

Tcl/Tk

Python

Perl

scripting

Sather

Eiffel

Java

C#

C++

VB

C

Basic

imperative or OOP

THE FIRST LANGUAGE

It's Scheme!

JavaScript

Ruby

Tcl/Tk

Python

Perl

scripting

Sather

Eiffel

Java

C#

C++

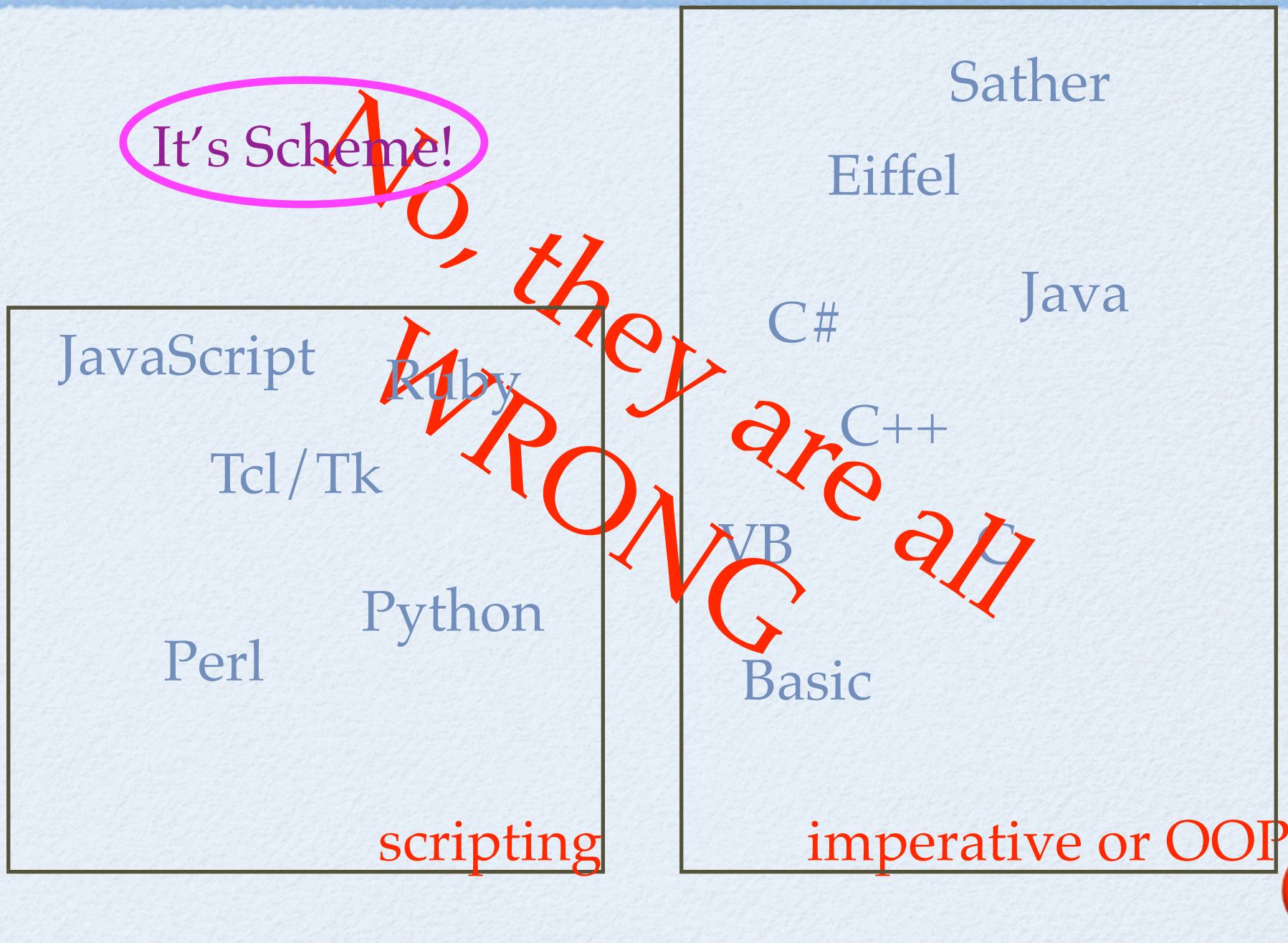
VB

C

Basic

imperative or OOP

THE FIRST LANGUAGE



THE FIRST LANGUAGE

```
// a first C++ program
main() {
  double price;
  int quantity;
  double total;
  ...
  price * quantity = total;
  ...
}
```

THE FIRST LANGUAGE

```
// a first C++ program
main() {
  double price;
  int quantity;
  double total;

  ...
  price * quantity = total;
  ...
}
```

LHS value expected here!

THE FIRST LANGUAGE

```
// a first C++ program
main() {
  double price;
  int quantity;
  double total;

  ...
  price * quantity = total;
  ...
}
```

LHS value expected here!

- 100s of hours of observations
- inner city, urban, suburban, public, private high schools, college freshmen
- ***Not a C++ problem.***
- ***Every language*** suffers.

THE FIRST LANGUAGE

```
// my first Java program with extends
class UP {
 UP(int x) {}
}

class DOWN extends UP {
 DOWN(int y) {}
}
```

THE FIRST LANGUAGE

```
// my first Java program with extends
class UP {
 UP(int x) {}
}

class DOWN extends UP {
 DOWN(int y) {}
}
```

constructor-error-message.java:6: cannot find symbol:
constructor UP()

THE FIRST LANGUAGE

```
// my first Java program with extends
class UP {
 UP(int x) {}
}

class DOWN extends UP {
 DOWN(int y) {}
}
```

Quick: What does this mean?

constructor-error-message.java:6: cannot find symbol:
constructor UP()

THE FIRST LANGUAGE

"You have to know everything all at once."

THE FIRST LANGUAGE

"You have to know everything all at once."

- We **can't** change the "all at once" part.

THE FIRST LANGUAGE

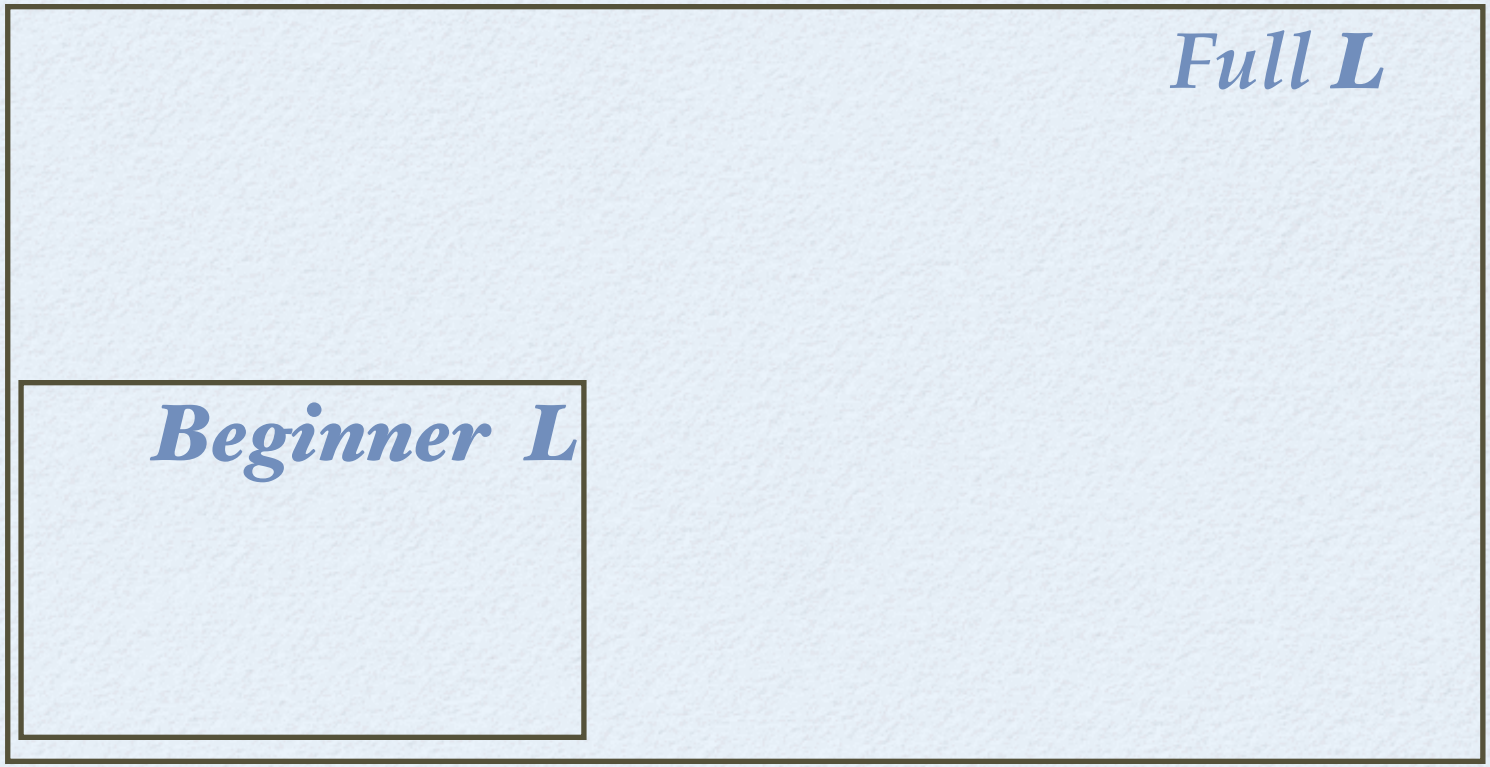
"You have to know everything all at once."

- We **can't** change the "all at once" part.
- We ***can*** change the "everything" part.

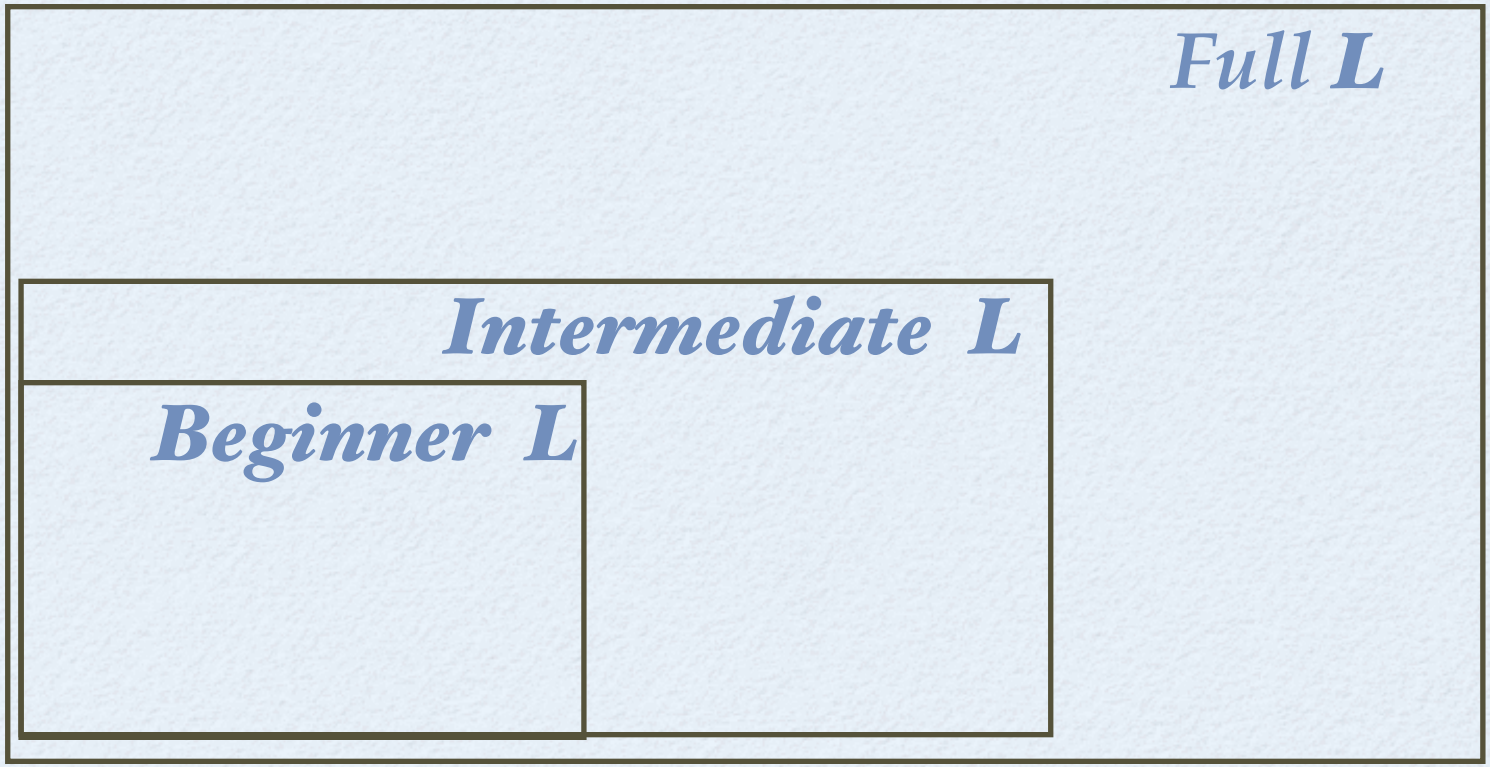
THE FIRST LANGUAGE

Full ***L***

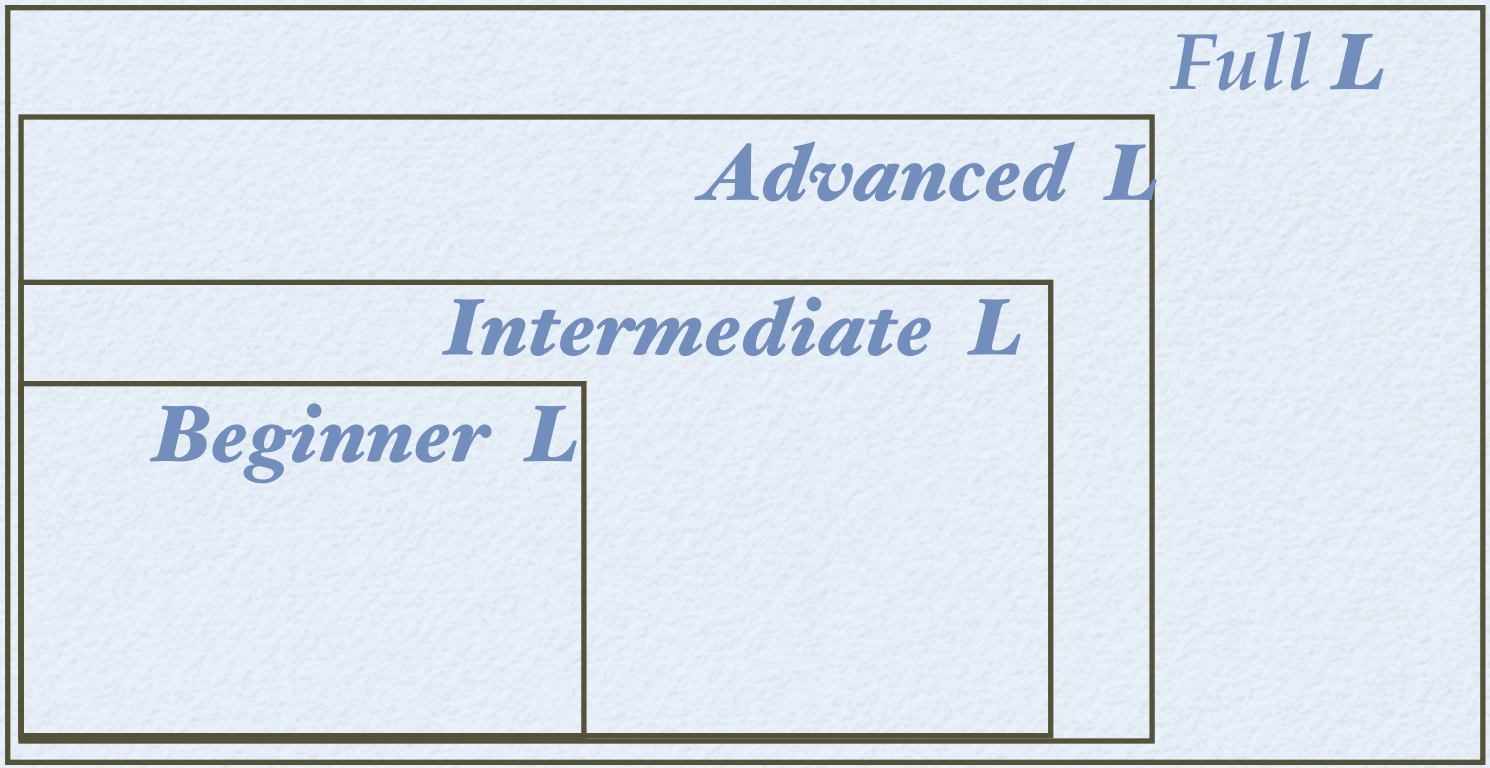
THE FIRST LANGUAGE



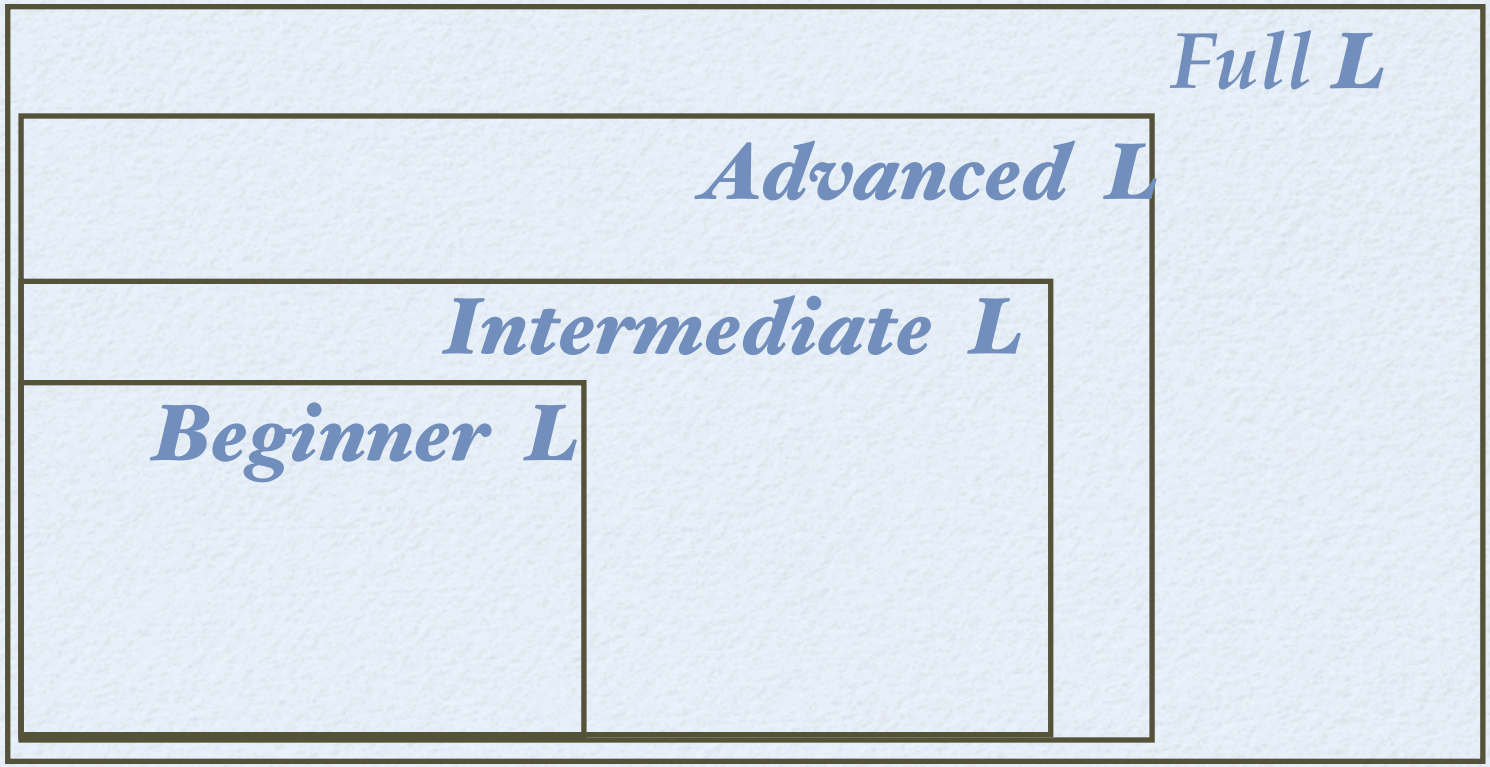
THE FIRST LANGUAGE



THE FIRST LANGUAGE



THE FIRST LANGUAGE



... and all of them are ***enforced and checked.***

The First Language

- Our choice: Scheme, a broken language
- Five subsets: *Beginning, Beginning with Quote, Intermediate, Intermediate with Lambda, Advanced*.
- ***Beginning Scheme*** is 8th grade algebra, plus structures
- *The others chosen based on* **design principles**.

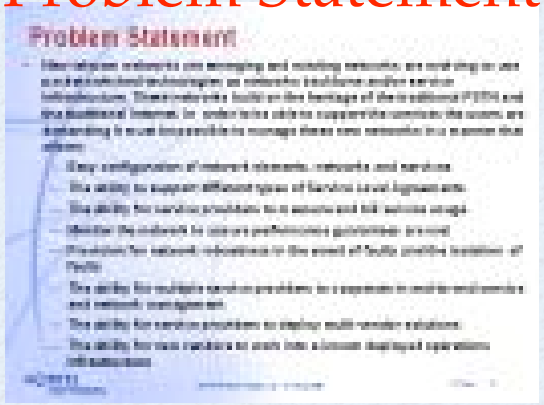
Design (HtDP)

HTDP

- ***Design*** is what happens before, while, and after you program.
- ***Design*** means to create programs in a systematic manner.
- ***Design*** is the opposite of "tinker with examples until it works."

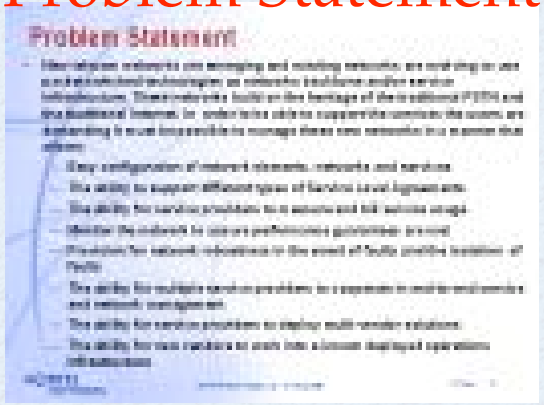
HTDP

Problem Statement



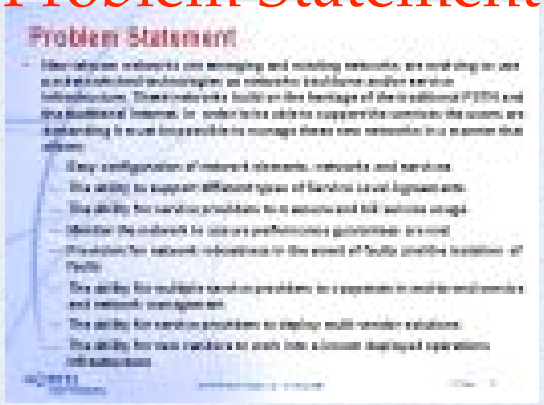
HTDP

Problem Statement


Problem Statement

HTDP

Problem Statement



How do you
“unstick”

HtDP

- A design method for beginners must unstick students at ***almost any point.***
- A design method must therefore ***guide students step by step*** from problem statement to program.
- It is only step-by-step if it is ***continuous*** (a small change in the problem statement leads to a systematic change to the function).

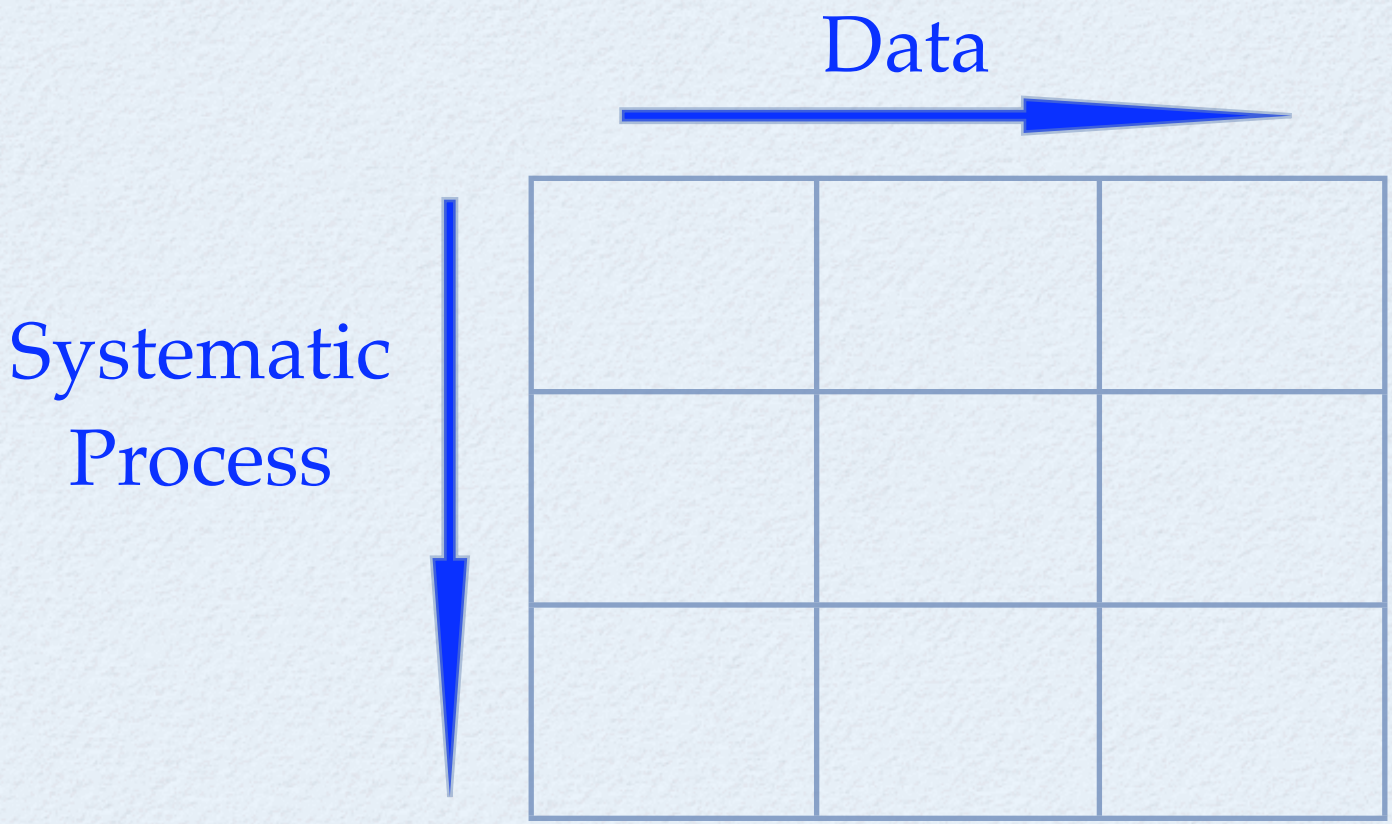
HTDP

- We are designing functions: ...
- ... things that consume data ...
- ... and produce data.

HtDP

- A design method must start from and must exploit the nature of input/output ***data***.
- A design method must do this ***systematically.***

HTDP: DESIGN RECIPES



HtDP

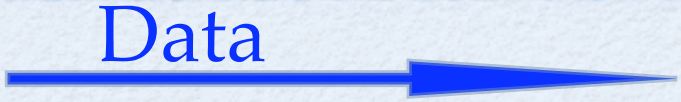
- The Design Recipes:
- (1) a series of steps from problem statement to solution expressed as questions;
- (2) the steps are fine-tuned to the nature of data that the function consumes.

HTDP

Systematic Process

- read the problem, extract a data description, illustrate the data description with examples
- state a purpose and type signature
- illustrate with behavioral examples
- translate the data descr. into a program organization (aka “take inventory”)

HtDP



- Data comes in sets (misnamed “classes”).
- Naive set theory guides development:
  - atomic (numbers, characters, ...)
  - unions (intervals, classification)
  - products (structures, records)
  - “inductive” (arbitrary size, ...)

HTDP: EXAMPLE

Problem:

Control a UFO via Keystrokes on Screen

Move the anchor point of a geometric shape along the horizontal or vertical according to key presses by the user.

HTDP: EXAMPLE

Data Descriptions

A ***Key*** is one of:
- char (#\a)
- 'left
- 'right
- 'up
- 'down
- other symbol

(**define-struct** point (x y))

A ***Point*** is a structure:
(make-point Number Number)

Examples:
-- 'left
-- (make-posn 20 30)

HTDP: EXAMPLE

Contract:

;; Point Key -> Point

Purpose Statement:

;; +/- *Xdelta* to *p* if *key* is 'right or 'left

;; +/- *Ydelta* to *p* if *key* is 'up or 'down

Function Signature:

(define (move *p key*) ...)

HTDP: EXAMPLE

Functional Examples:

move((makePoint 10 20),'left)
should be (make-point (+ ... 10) 20)

move((make-point 10 20),'up)
should be (make-point 10 (+ ... 20))

HtDP: Example

Inventory:

```
(define (move p key)
  (cond
    [ "key is a char" ... p ... ]
    [ "key is 'left"   ... (point-x p) ... (point-y p) ...]
    [ "key is 'right"  ... (point-x p) ... (point-y p) ...]
    [ "key is 'up"     ... (point-x p) ... (point-y p) ...]
    [ "key is 'down"   ... (point-x p) ... (point-y p) ...]
    [ "otherwise"      ... p ...]
```

HTDP: EXAMPLE

The Program:

from here, the program is self-evident

HTDP: EXAMPLE

Tests:

```
(check-expect (move (make-posn 10 20) 'left)
              (make-posn 5 20))
```

...

HTDP: FUN

- Run: the function is used via a callback hook: (on-key-event move)
- Students write GUI-controlled programs from week 2.
- And they get to know the distinction between ***running*** and ***testing*** a program.

HTDP: ANOTHER EXAMPLE

Problem:

Process the Files in A Directory ...

Design a function that finds put whether a file with some given name exists in a folder structure.

HTDP: ANOTHER EXAMPLE

Data Description:

A ***folder*** is a structure with two fields:
- name, list of files and folders.

A ***list of files and folders*** is one of:
- the empty list
- a list of files and folders extended with a file
- a list of files and folders extended with a folder

A ***file*** is a name.

HTDP: ANOTHER EXAMPLE

Data Description:

A ***folder*** is a structure with two fields:
- name, list of files and folders.

A ***list of files and folders*** is one of:
- the empty list
- a list of files and folders extended with a file
- a list of files and folders extended with a folder.

A ***file*** is a name.

HtDP: Another Example

Socratic Scripts:

- Does the data description consist of several disjoint subsets? How many?
- For each subset (subclass): Is it a structure? What are its fields?
- Does the data description refer to itself? Make the function recursive in analogous places!

HtDP: Another Example

Socratic Scripts:

- Does the data description consist of several disjoint subsets? How many?
- For each subset (subclass): Is it a structure? What are its fields?
- Does the data description refer to itself? Make the function recursive in analogous places!

This is material for the 7th week for total novices at NU.

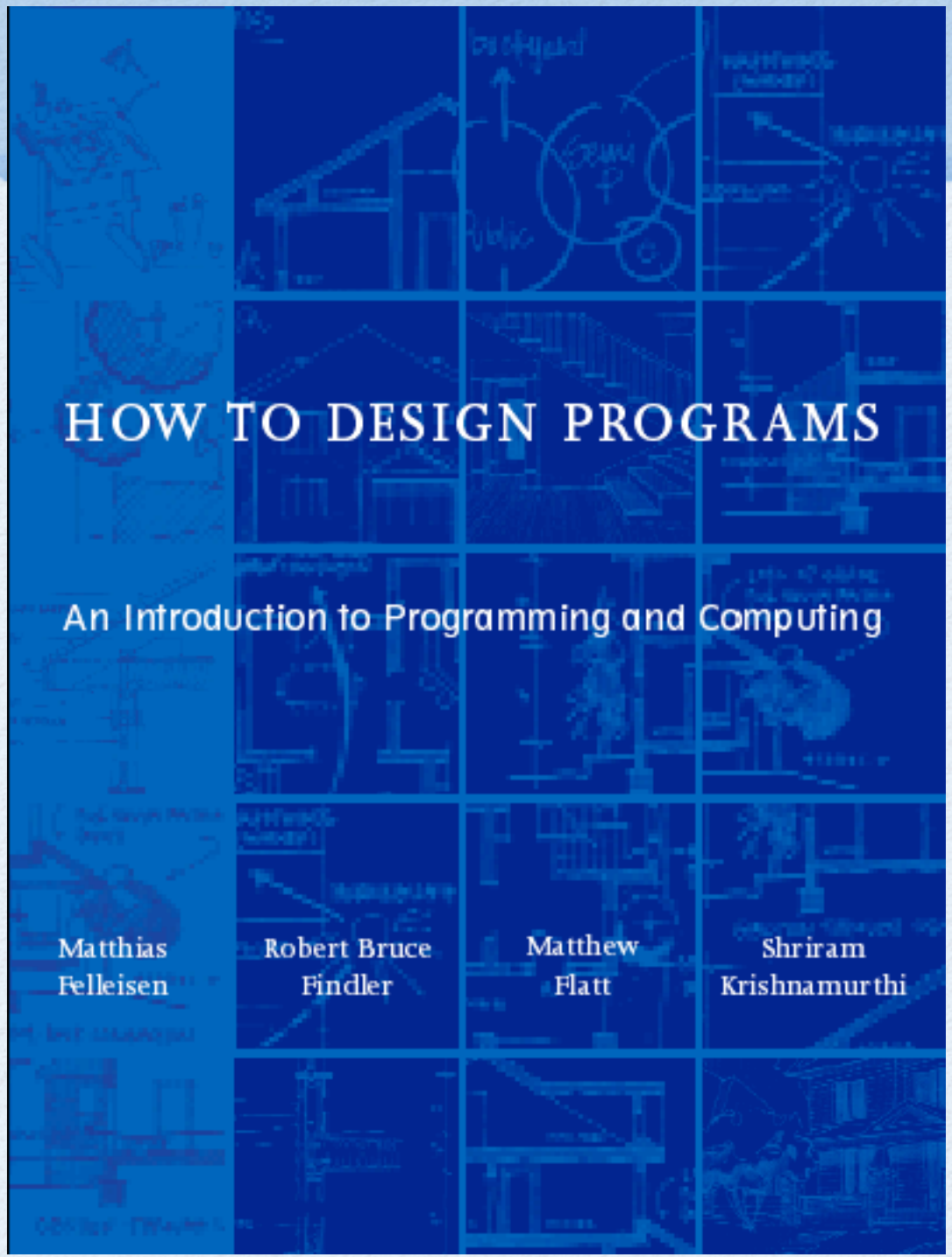
HTDP: More Design

- iterative refinement (data modeling)
- abstraction over similar data (polym.)
- abstraction over similar functions (h.o.)
- generative recursion (graph traversal)
- functions with accumulators
- memory (state and assignment)

HTDP

Details too much for a talk.

Warning: Commercial Plug

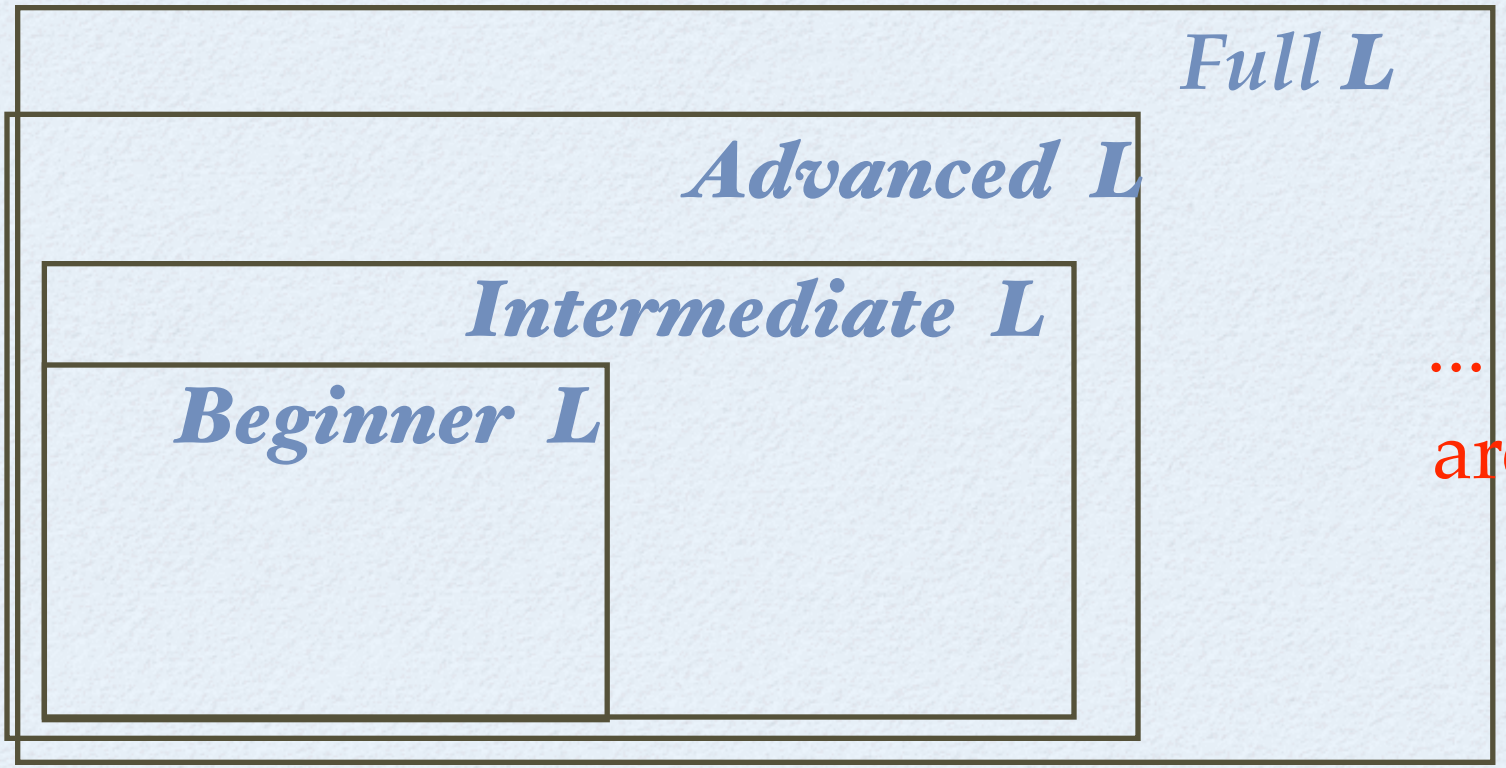

HOW TO DESIGN PROGRAMS
An Introduction to Programming and Computing
Matthias Felleisen
Robert Bruce Findler
Matthew Flatt
Shriram Krishnamurthi

HTDP

Dewarning: http://www.htdp.org/

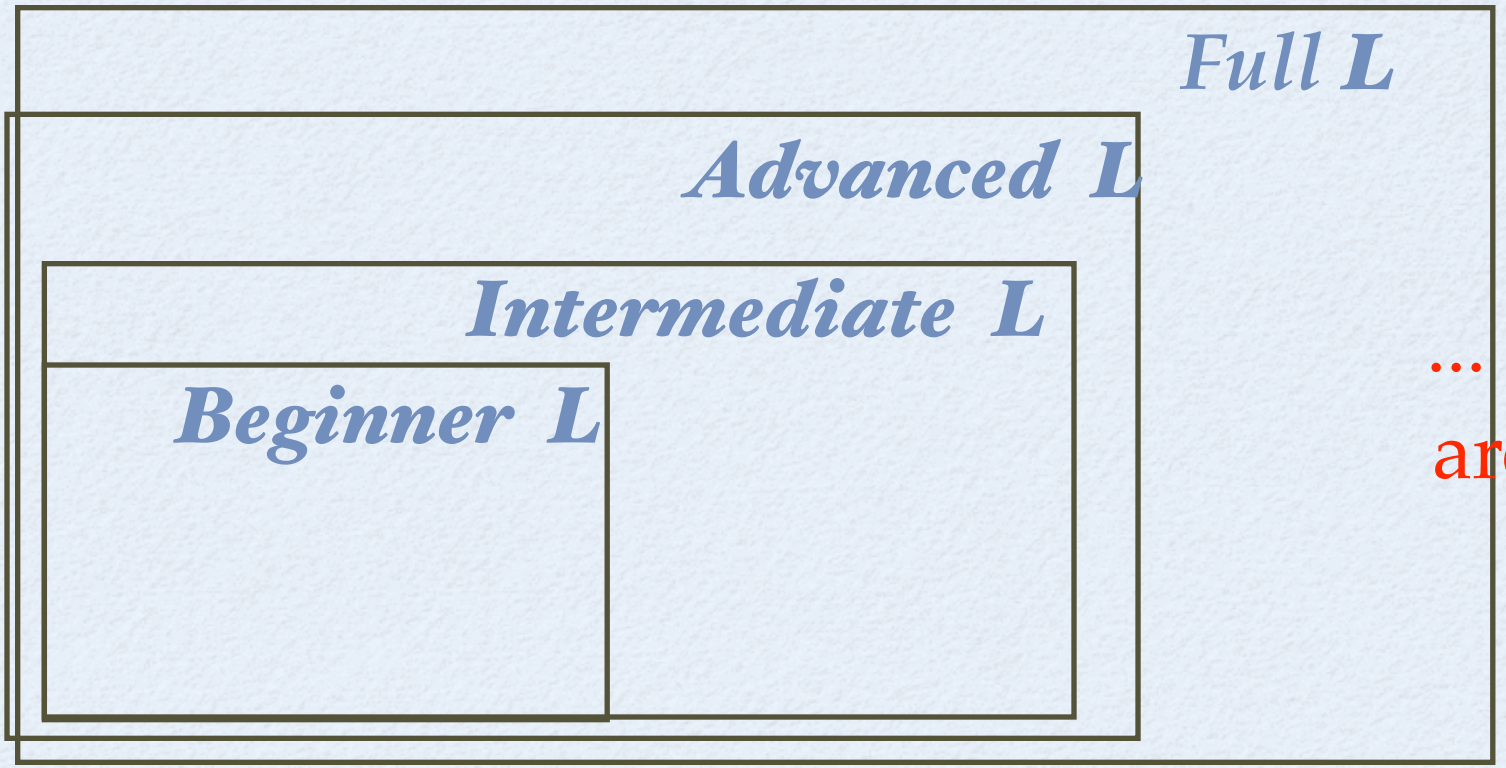
Linguistics 2

THE FIRST LANGUAGES



... and all of them are ***enforced and checked.***

THE FIRST LANGUAGES



... and all of them are ***enforced and checked.***

How do you enforce and support them?

THE FIRST LANGUAGES

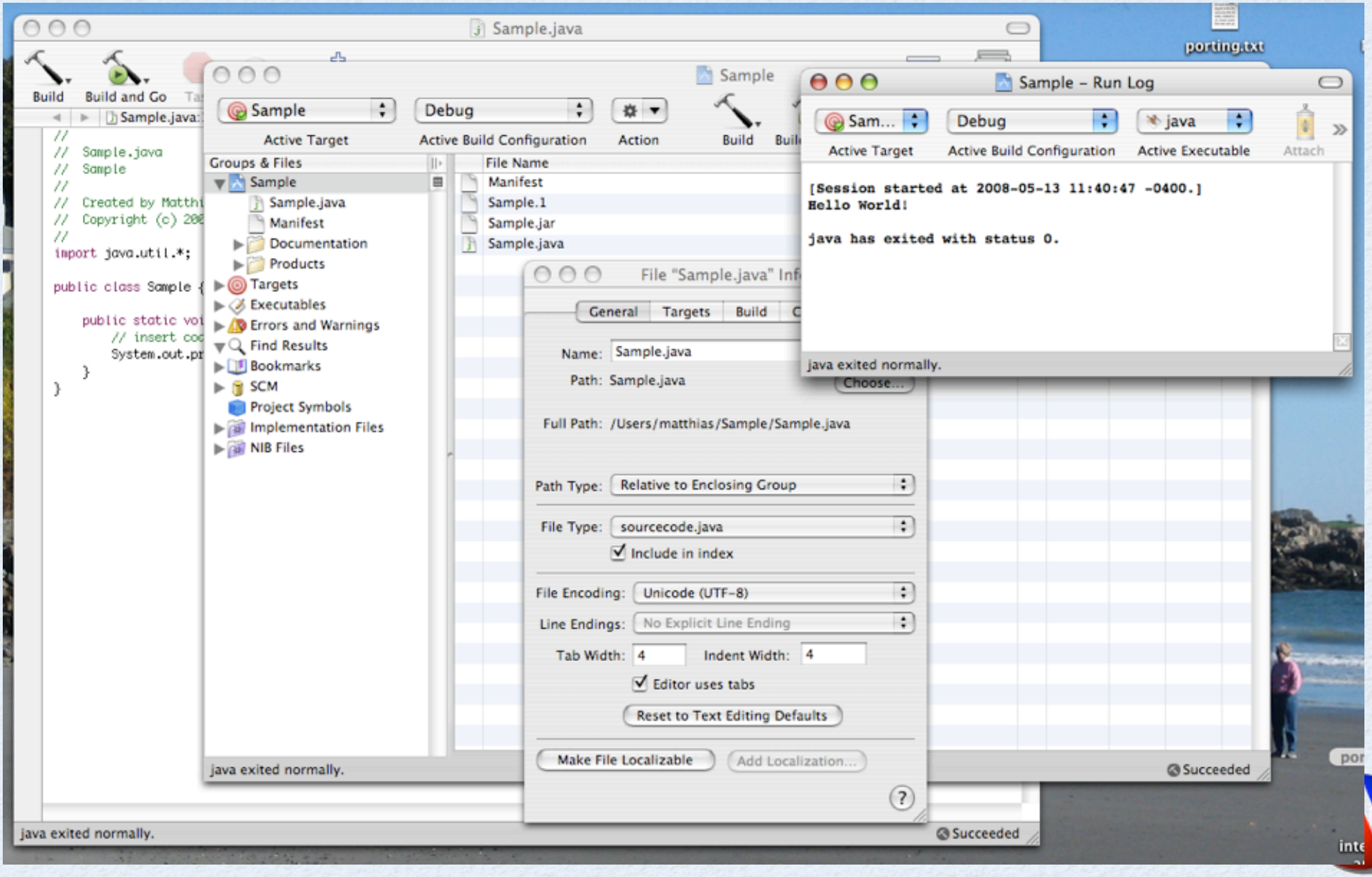
- Each language comes with a compiler and its own error reporting.
- Each language comes with an **algebraic** stepper.
- Each language comes with *interactive* REPL.

THE FIRST LANGUAGES

- Each language comes with a compiler and its own error reporting.
- Each language comes with an **algebraic** stepper.
- Each language comes with *interactive* REPL.

So we did it for all five of them, in one programming environment.

THE FIRST ENVIRONMENT



THE FIRST ENVIRONMENT

Thanks Viera Proulx

A GOOD FIRST IDE

Our Response to the 747-IDE

first-year.ss – DrScheme

first-year.ss
(define ...)

Step | Check Syntax | Run | Stop

```
;; draw : Position -> Image
;; create a drawing of the rocket at the proper position
(define (draw p)

  (place-image  XX p (empty-scene WIDTH HEIGHT)))

;; move : Position -> Position
;; move a rocket straight up
(define (move p)
  (+ 3 p))

;; tests:

(check-expect (move 10) 13)

(check-expect (draw 10) (place-image  XX 10 (empty-scene WIDTH HEIGHT)))
(generate-report)
```

Welcome to DrScheme, version 369.8-svn26feb2007 [3m].
Language: Beginning Student.
Teachpack: /Users/matthias/plt/collects/teachpack/htdp/testing.ss.
Teachpack: /Users/matthias/plt/collects/teachpack/htdp/world.ss.
>

Programming language:
Beginning Student

25:16 | GC | 66,476,132 | Read/ Write | not running

first-year.ss - DrScheme

first-year.ss
(define ...)
Save
Step
Check Syntax
Run
Stop

```
;; draw : Position -> Image
;; create a drawing of the rocket at the proper position
(define (draw p)
  (place-image XX p (empty-scene WIDTH HEIGHT)))

;; move : Position -> Position
;; move a rocket straight up
(define (move p)
  (+ 3 p))

;; tests:

(check-expect (move 10) 13)

(check-expect (draw 10) (place-image XX 10 (empty-scene WIDTH HEIGHT)))
(generate-report)

;; run program run
```

Test Results

Recorded 2 checks. All checks succeeded!

Welcome to DrScheme, version 369.8-svn26feb2007 [3m].
Language: Beginning Student.
Teachpack: /Users/matthias/plt/collects/teachpack/htdp/testing.ss.
Teachpack: /Users/matthias/plt/collects/teachpack/htdp/world.ss.
true
> (draw 20)

Programming language:
Beginning Student
18:72
GC
66,476,132
Read/
Write
not running

first-year.ss – DrScheme

first-year.ss

(define ...)

Step | Check Syntax | Run | Stop

```
(define WIDTH  300)
(define HEIGHT 300)
(define XX     100)

;; draw : Position -> Image
;; create a drawing of the rocket at the proper positi
(define (draw p)
```

Test Results

Recorded 2 checks. All che

DrScheme

Stop | Images

```
(check-expect (move 10) 15)
```

Welcome to DrScheme, version 369.8-svn26feb2007 [3m].
Language: Beginning Student.
Teachpack: /Users/matthias/plt/collects/teachpack/htdp/testing.ss.
Teachpack: /Users/matthias/plt/collects/teachpack/htdp/world.ss.

```
true
true
true
true
>
```

Programming language:
Beginning Student

GC | 66,476,132 | not running

Stepper

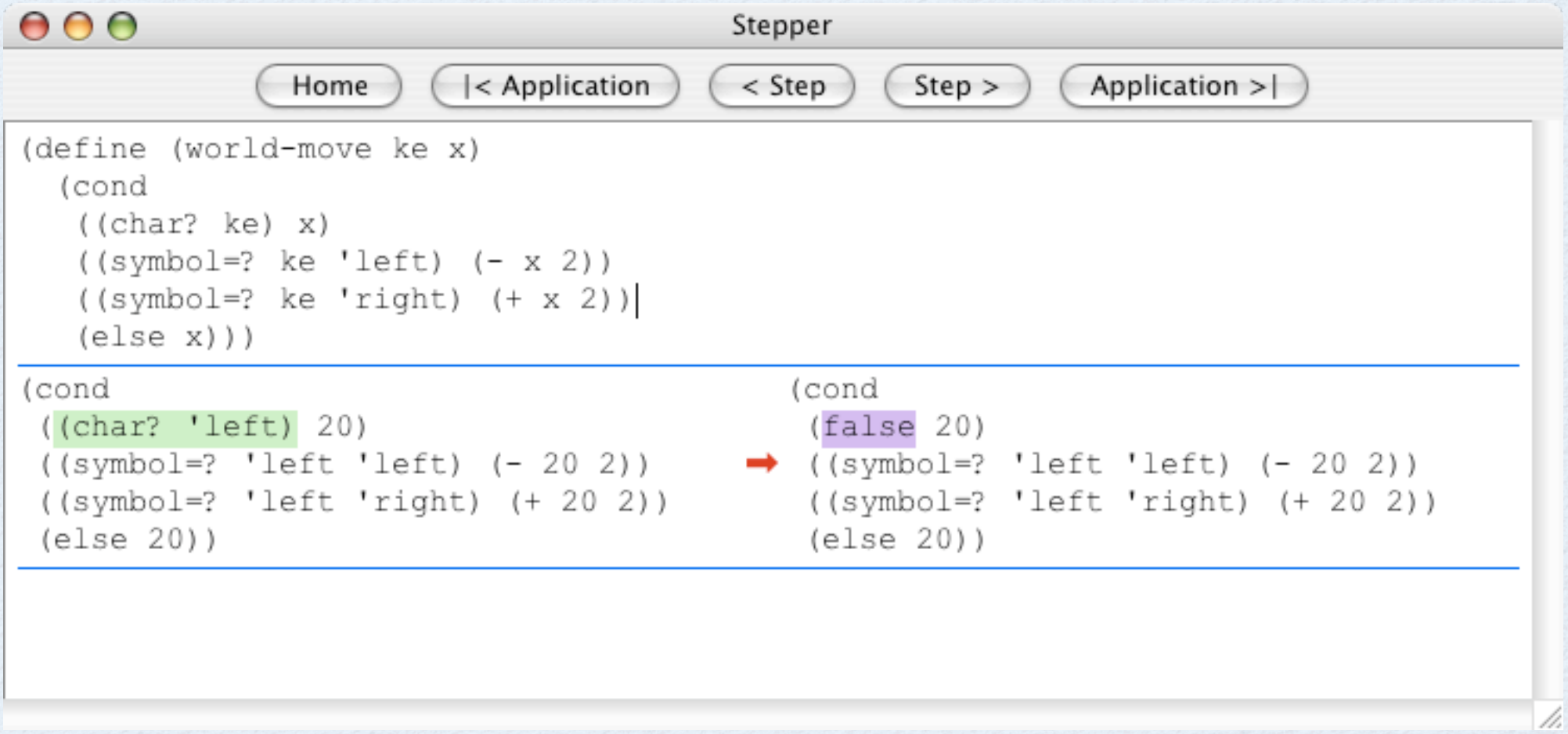
Home | |< Application | < Step | Step > | Application >|

```
(define (world-move ke x)
  (cond
   ((char? ke) x)
   ((symbol=? ke 'left) (- x 2))
   ((symbol=? ke 'right) (+ x 2))
   (else x)))
```

```
(world-move 'left 20)
```

➡

```
(cond
 ((char? 'left) 20)
 ((symbol=? 'left 'left) (- 20 2))
 ((symbol=? 'left 'right) (+ 20 2))
 (else 20))
```


Stepper
Home
|< Application
< Step
Step >
Application >|
(define (world-move ke x)
(cond
((char? ke) x)
((symbol=? ke 'left) (- x 2))
((symbol=? ke 'right) (+ x 2))
(else x)))
(cond
((char? 'left) 20)
((symbol=? 'left 'left) (- 20 2))
((symbol=? 'left 'right) (+ 20 2))
(else 20))
(cond
(false 20)
((symbol=? 'left 'left) (- 20 2))
((symbol=? 'left 'right) (+ 20 2))
(else 20))

Stepper

Home | |< Application | < Step | Step > | Application >|

```
(define (world-move ke x)
  (cond
   ((char? ke) x)
   ((symbol=? ke 'left) (- x 2))
   ((symbol=? ke 'right) (+ x 2))
   (else x)))
```

```
(cond
 (false 20)
 ((symbol=? 'left 'left) (- 20 2))
 ((symbol=? 'left 'right) (+ 20 2))
 (else 20))
```

```
(cond
 ((symbol=? 'left 'left) (- 20 2))
 ((symbol=? 'left 'right) (+ 20 2))
 (else 20))
```

Stepper

Home | |< Application | < Step | Step > | Application >| | End

```
(define XX 100)
(define (draw p)
  (place-image
```

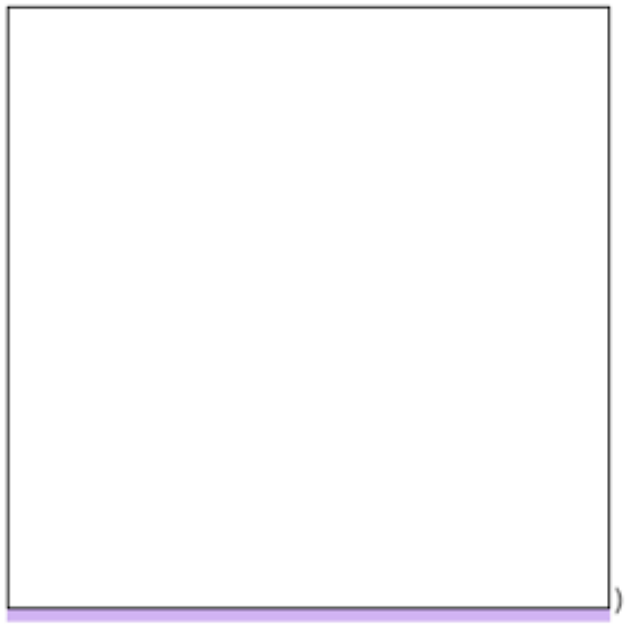
```
   XX
   p
    (empty-scene WIDTH HEIGHT)))
(place-image
```

```
  100
  42
  (empty-scene 300 300))
```

```
(define XX 100)
(define (draw p)
  (place-image
```

```
   XX
   p
    (empty-scene WIDTH HEIGHT)))
(place-image
```

```
  100
  42
```

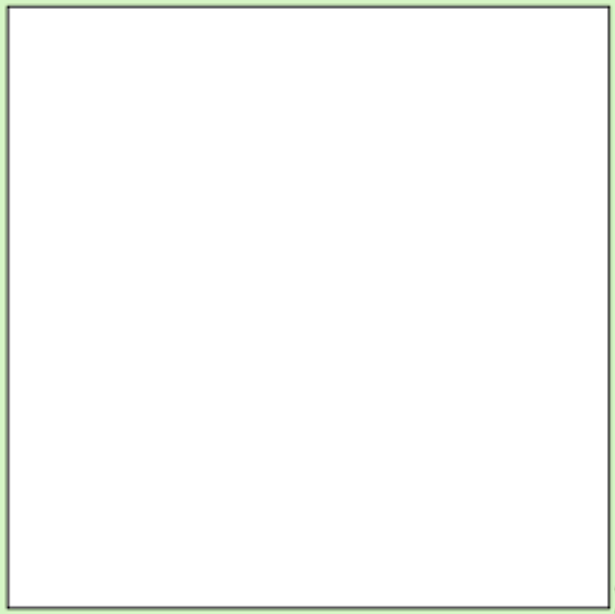


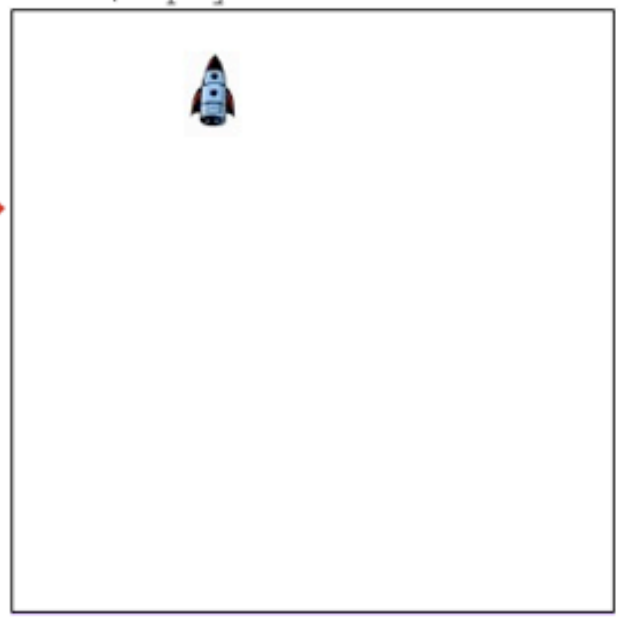
)

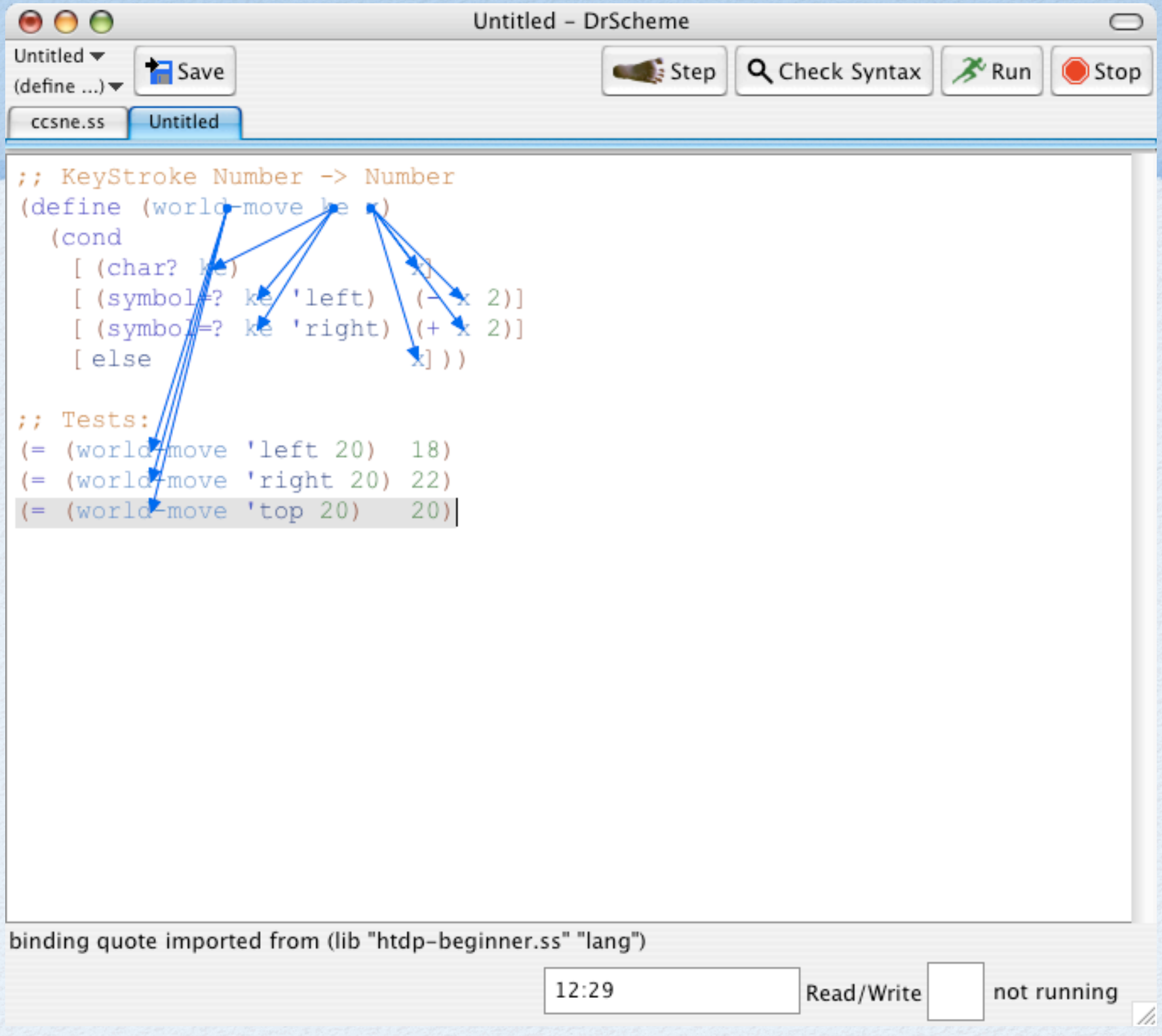
Stepper

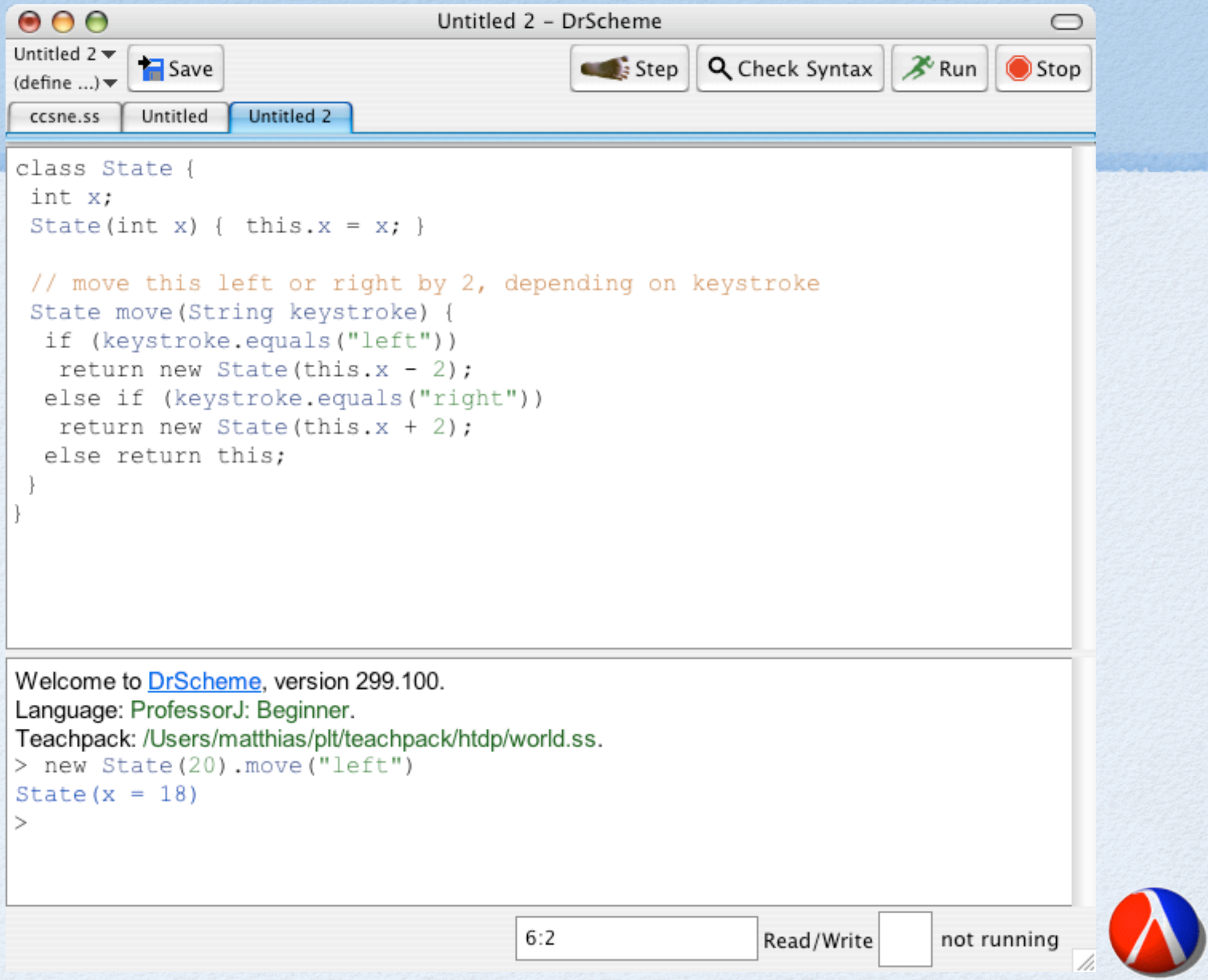
Home | |< Application | < Step | Step > | Application >| | End

```
(define XX 100)
(define (draw p)
  (place-image
   
   XX
   p
   (empty-scene WIDTH HEIGHT)))
```

```
(place-image
 
 100
 42
 
)
```




Untitled - DrScheme
Untitled
(define ...)
Save
Step
Check Syntax
Run
Stop
ccsne.ss
Untitled
;; KeyStroke Number -> Number
(define (world-move ke x)
(cond
[(char? ke) x]
[(symbol=? ke 'left) (- x 2)]
[(symbol=? ke 'right) (+ x 2)]
[else x]))
;; Tests:
(= (world-move 'left 20) 18)
(= (world-move 'right 20) 22)
(= (world-move 'top 20) 20)
binding quote imported from (lib "htdp-beginner.ss" "lang")
12:29
Read/Write
not running


Untitled 2 – DrScheme
Untitled 2
(define ...)
Save
Step
Check Syntax
Run
Stop
ccsne.ss
Untitled
Untitled 2
class State {
int x;
State(int x) { this.x = x; }
// move this left or right by 2, depending on keystroke
State move(String keystroke) {
if (keystroke.equals("left"))
return new State(this.x - 2);
else if (keystroke.equals("right"))
return new State(this.x + 2);
else return this;
}
}
Welcome to DrScheme, version 299.100.
Language: ProfessorJ: Beginner.
Teachpack: /Users/matthias/plt/teachpack/htdp/world.ss.
> new State(20).move("left")
State(x = 18)
>
6:2
Read/Write
not running

Design (HtDC)

Semester 2

HTDC

- Classes for Objects, Types to Check
- Does the Design Process Apply?
- Does Java Support it All?

HTDC: DESIGN RECIPE (1)

Split the Design Recipe

HtDC: Design Recipe (1)

Split the Design Recipe

Design Data:

- Diagrams & Classes
- Sample Instances
- Representation &

Interpretation

HTDC: DESIGN RECIPE (1)

Split the Design Recipe

Design Data:

- Diagrams & Classes
- Sample Instances
- Representation & Interpretation

Design Methods:

- Signature & Purpose
- Functional Examples
- Template = Chase the Diagram
- Program!
- Tests

HTDC: CLASS DESIGN

HTDC: CLASS DESIGN

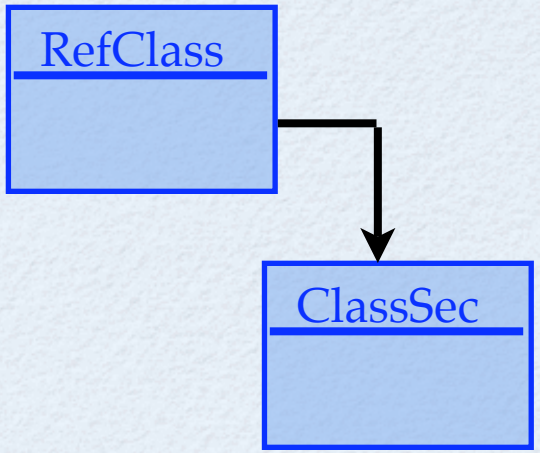
"atomic" types

HTDC: CLASS DESIGN

"atomic" types

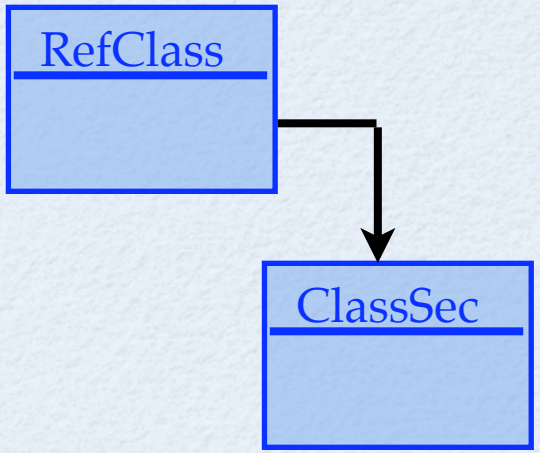
HTDC: CLASS DESIGN

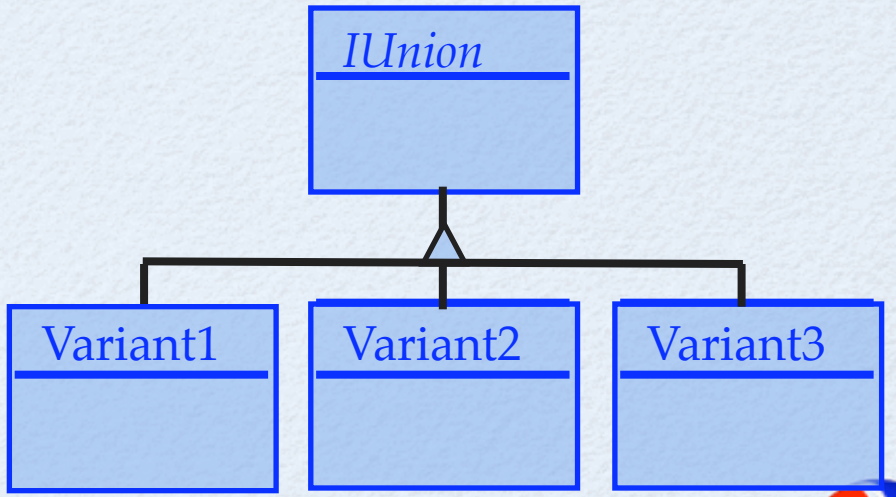
"atomic" types



HTDC: CLASS DESIGN

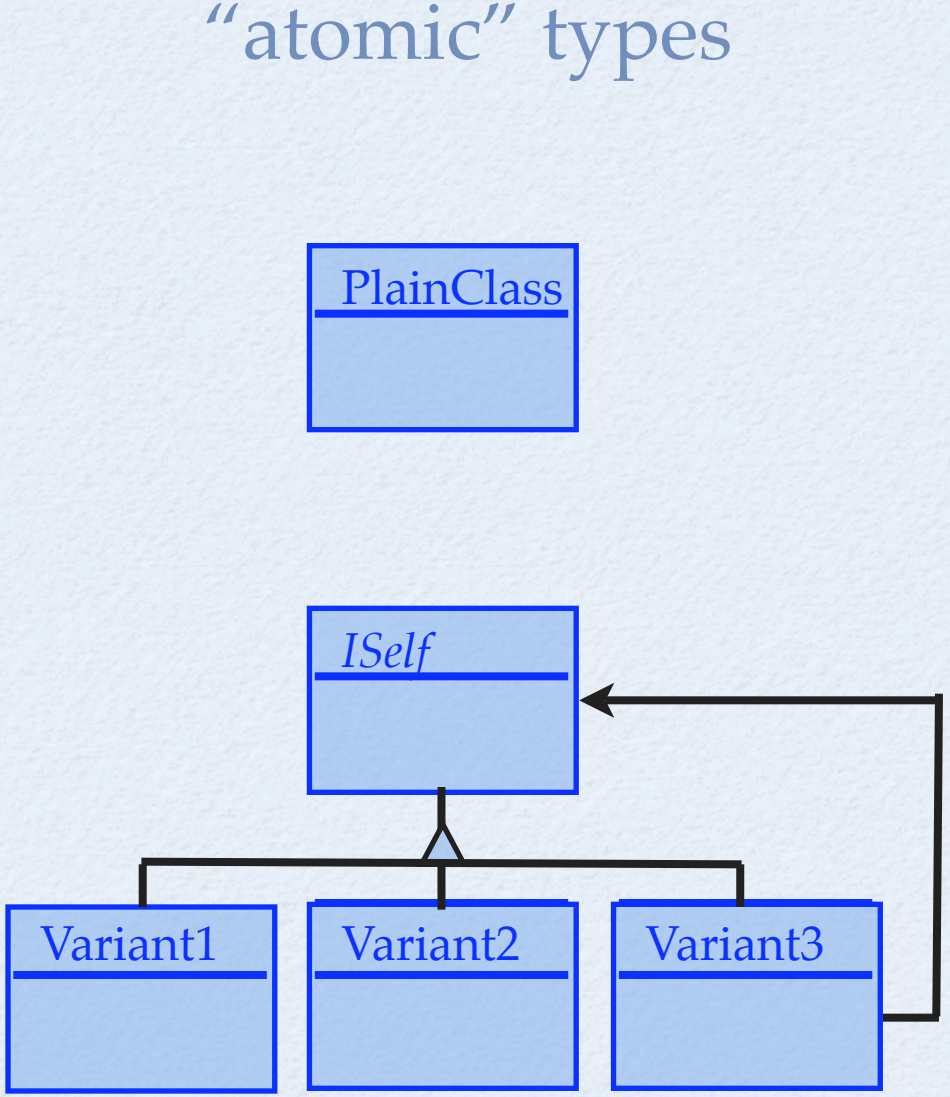
"atomic" types

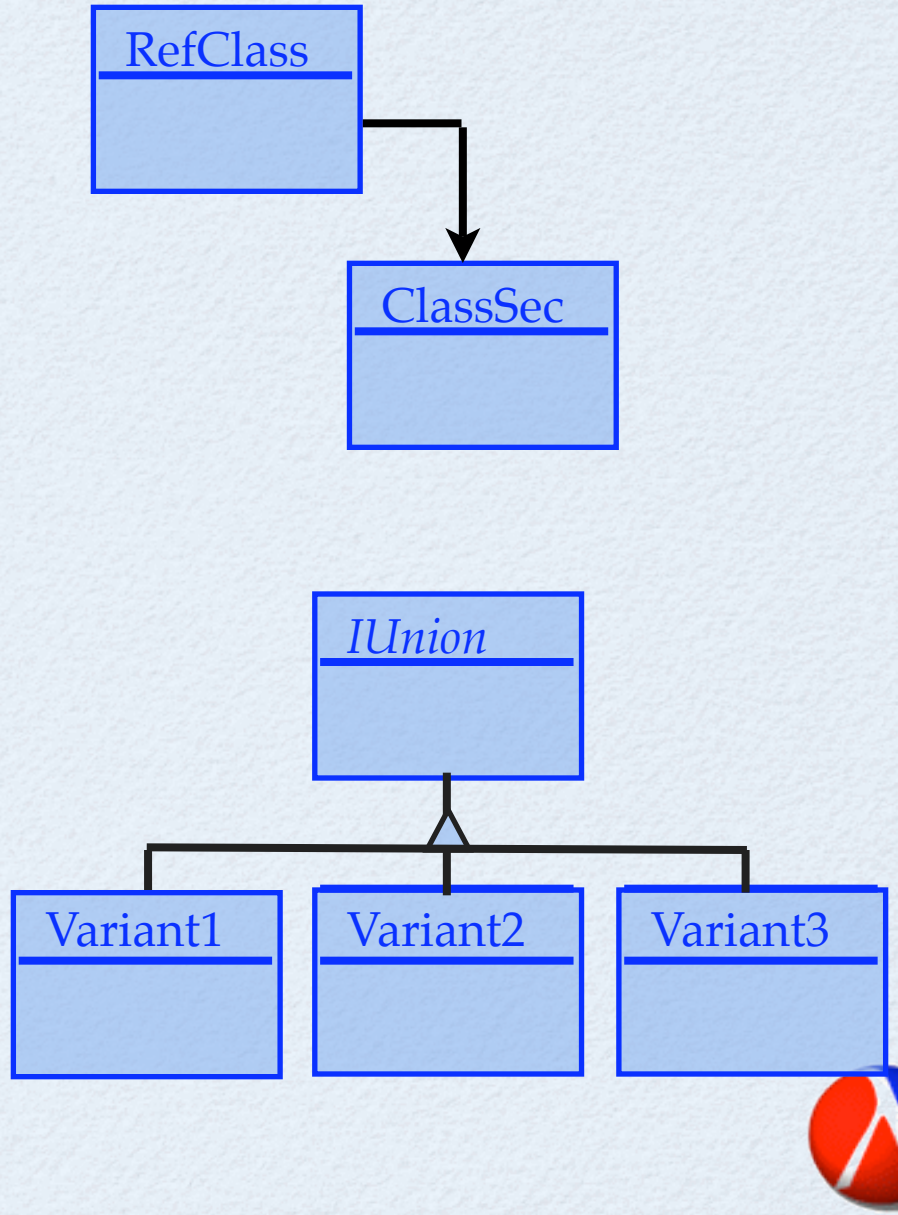




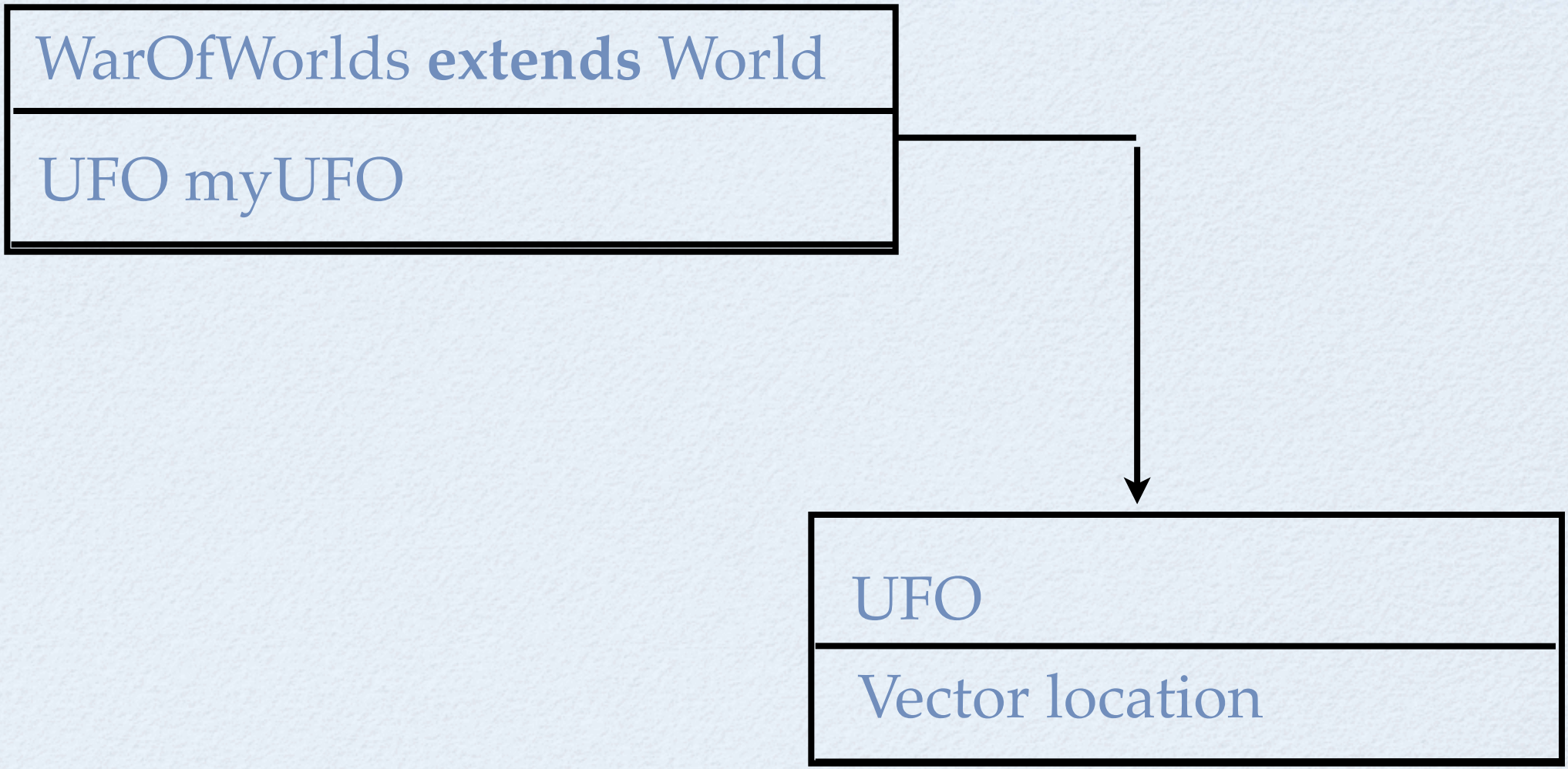
HTDC: CLASS DESIGN

"atomic" types





DESIGN EXAMPLE



DESIGN EXAMPLE

WarOfWorlds **extends** World

UFO myUFO

UFO

Vector location

Examples

Vector uLoc = ...
UFO u = ...
World w = ...

DESIGN EXAMPLE

WarOfWorlds
UFO myUFO
World onKeyEvent(String ke)

Examples
Vector uLoc = ... UFO u = ... World w = ...

UFO
Vector location
UFO move(Vector delta)

DESIGN EXAMPLE

WarOfWorlds
UFO myUFO
World onKeyEvent(String ke)

UFO
Vector location
UFO move(Vector delta)

Examples
Vector uLoc = ... UFO u = ... World w = ...

DESIGN EXAMPLE

WarOfWorlds

UFO myUFO

World onKeyEvent(String ke)

UFO

Vector location

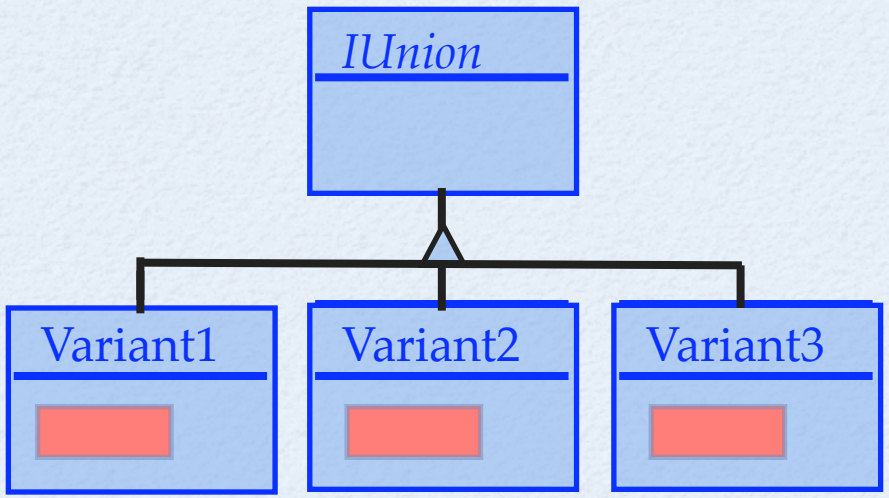
UFO move(Vector delta)

Examples

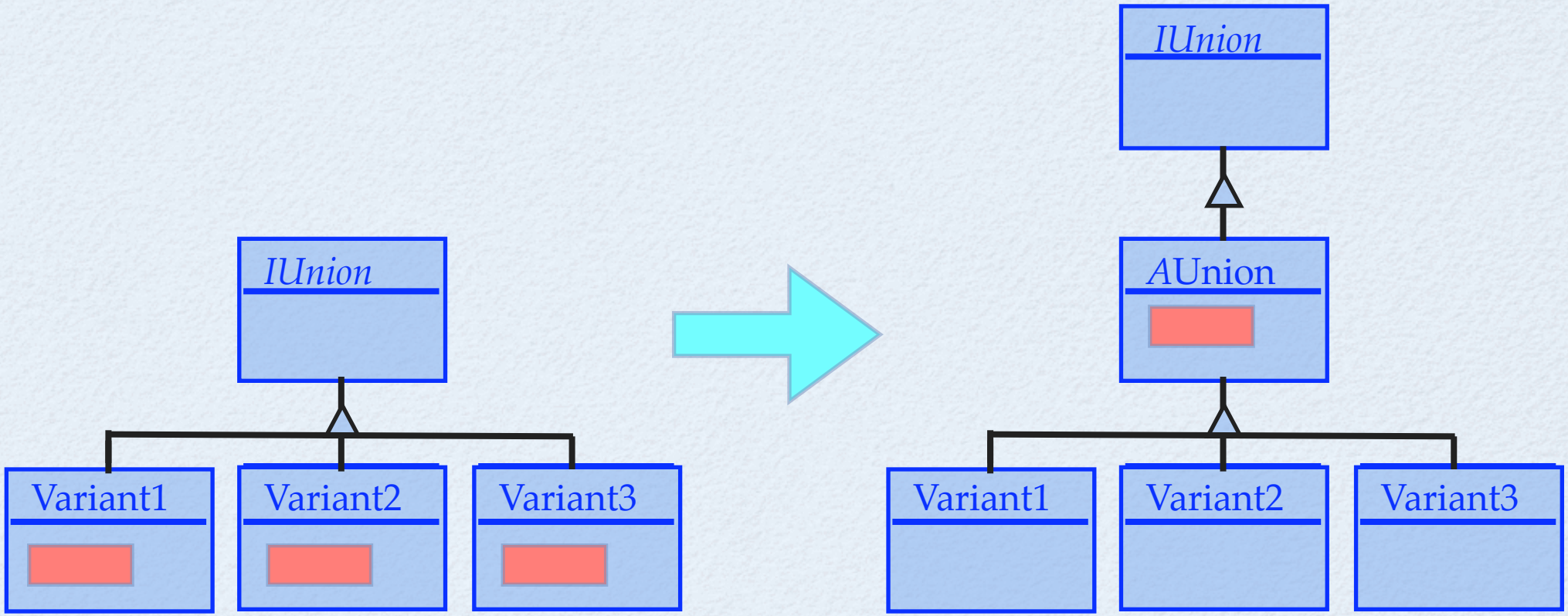
Vector uLoc = ...
UFO u = ...

World w = ...

HTDC: ABSTRACTION

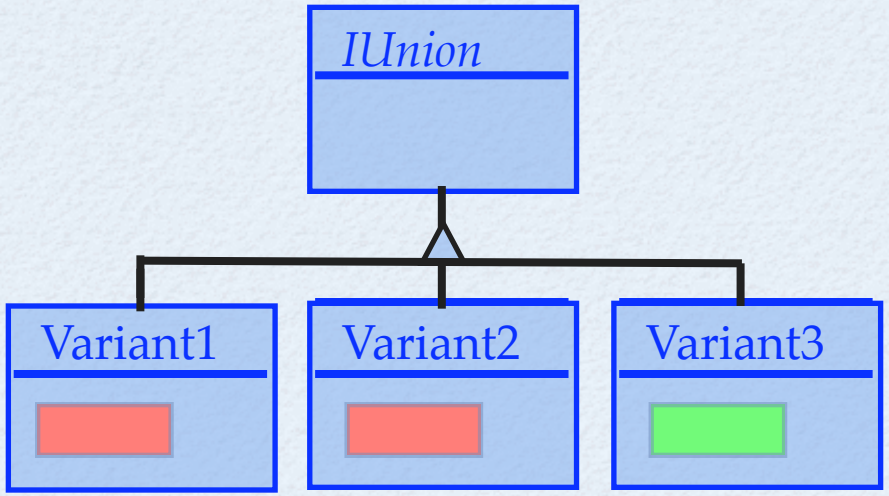


HTDC: ABSTRACTION

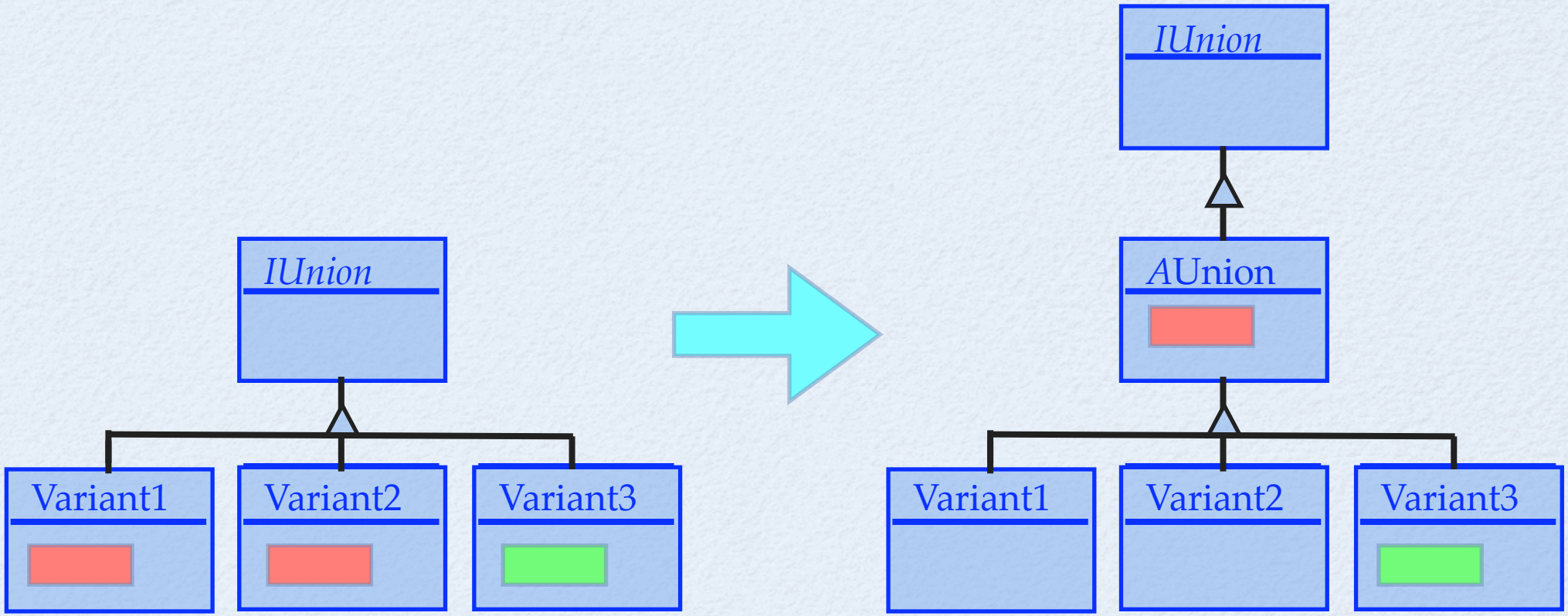


Inheritance

HTDC: ABSTRACTION

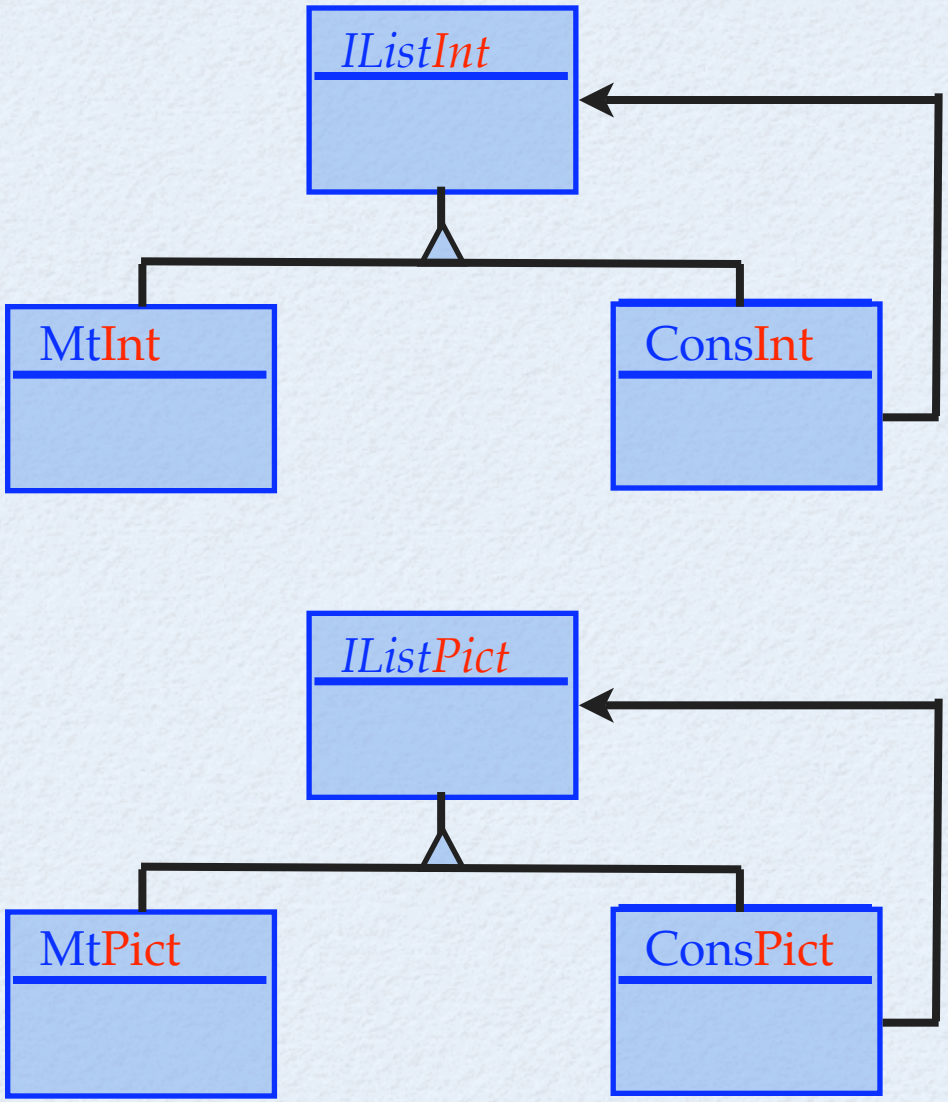


HTDC: ABSTRACTION

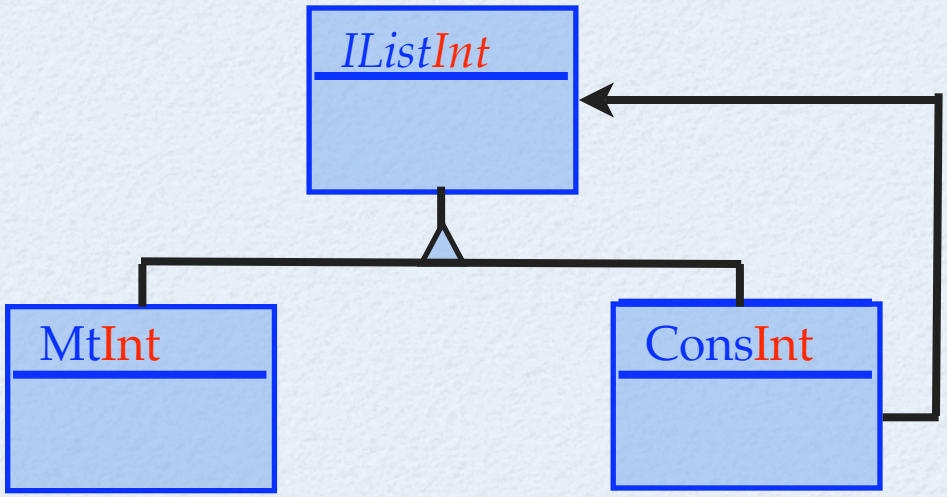


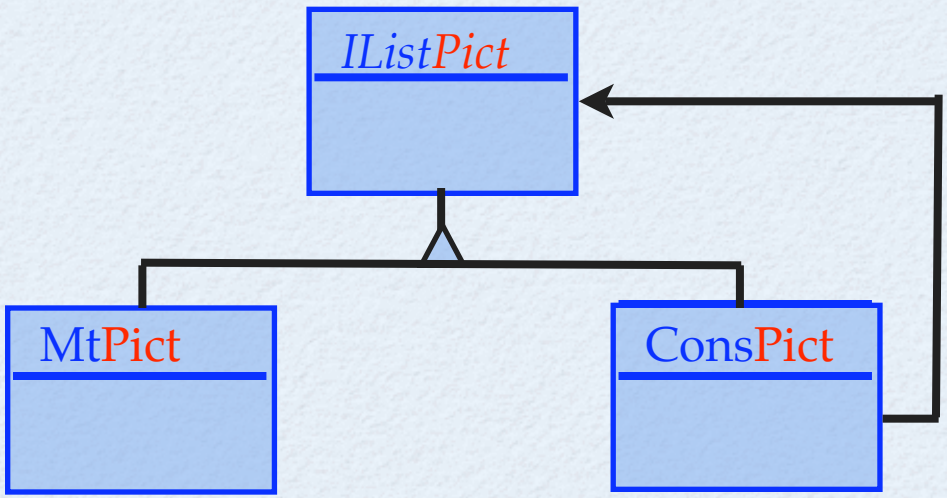
Inheritance and Overriding

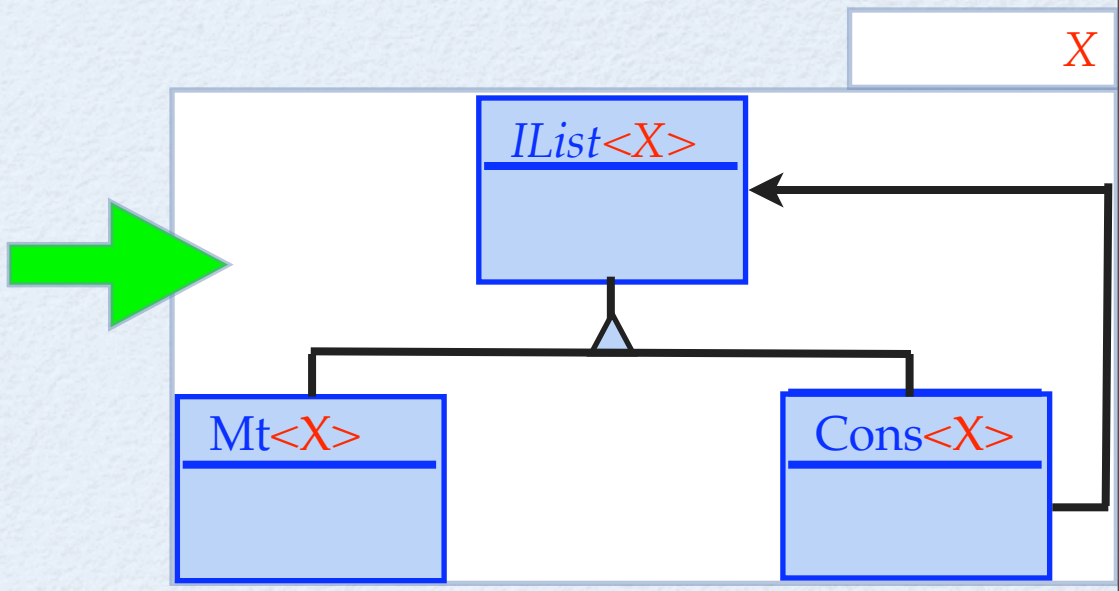
HTDC: ABSTRACTION



HTDC: ABSTRACTION







Generics

HTDC: ABSTRACTION

- detect similar but distinct traversals
- abstract over distinct action with “first-class” methods ...
- ... which don’t exist: represent as instances of command class

HTDC: ABSTRACTION

- detect similar but distinct traversals
- abstract over distinct action with “first-class” methods ...
- ... which don’t exist: represent as instances of command class

Note: systematics “rediscovery” of patterns

HTDC: ENCAPSULATION

- idea: program for others, protect invariants
- for state, protection is encapsulation

HTDC: JAVA

- Java Teaching Languages: Beginning Student, Intermediate Student, Advanced Student
- switch to Eclipse and Java 1.5, with JUnit
- introduce Java run-time library

Results

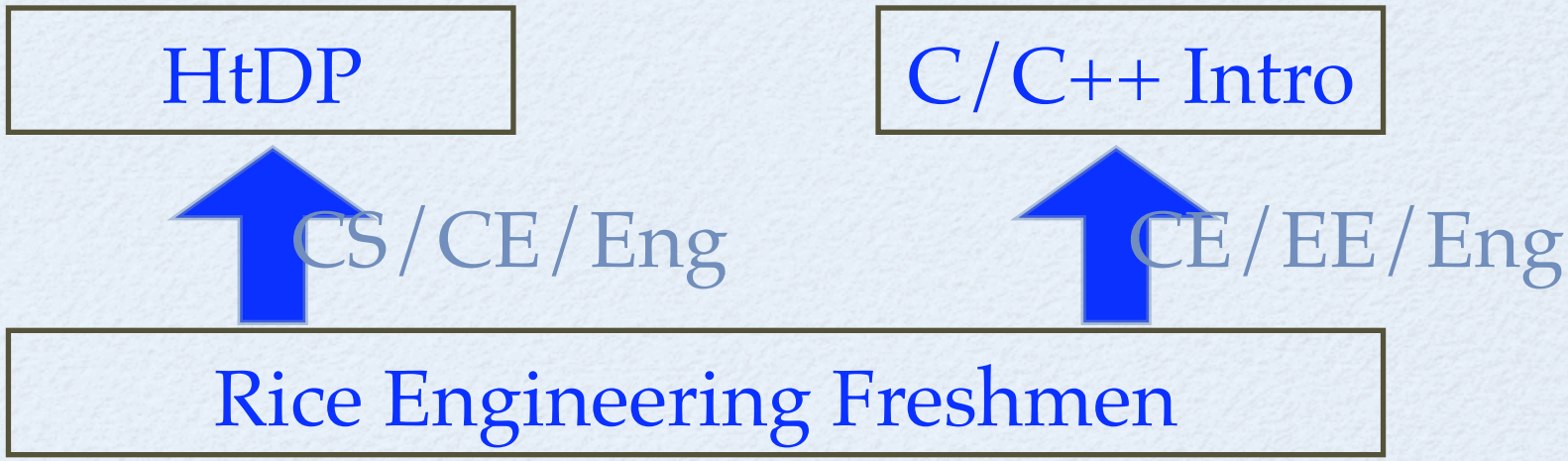
RESULTS: FIRST SEMESTER

- The first semester: Ten Years
- Evaluation with respect to C++ course(s)
- Hand-over test at Rice: five instructors
- Field tests at 20 local high-schools
- Now used at 12+ colleges and 200+ high schools

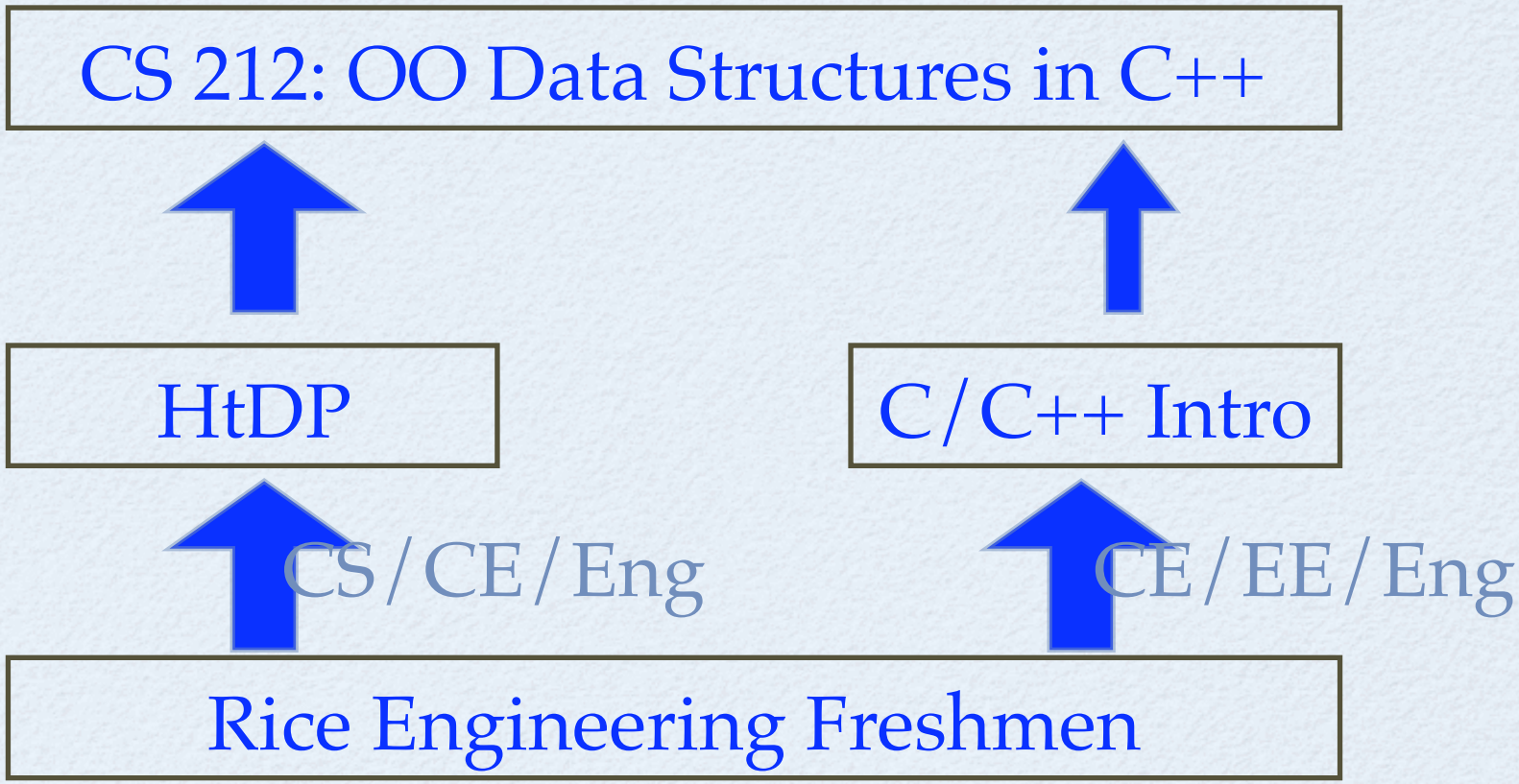
RESULTS: RICE

Rice Engineering Freshmen

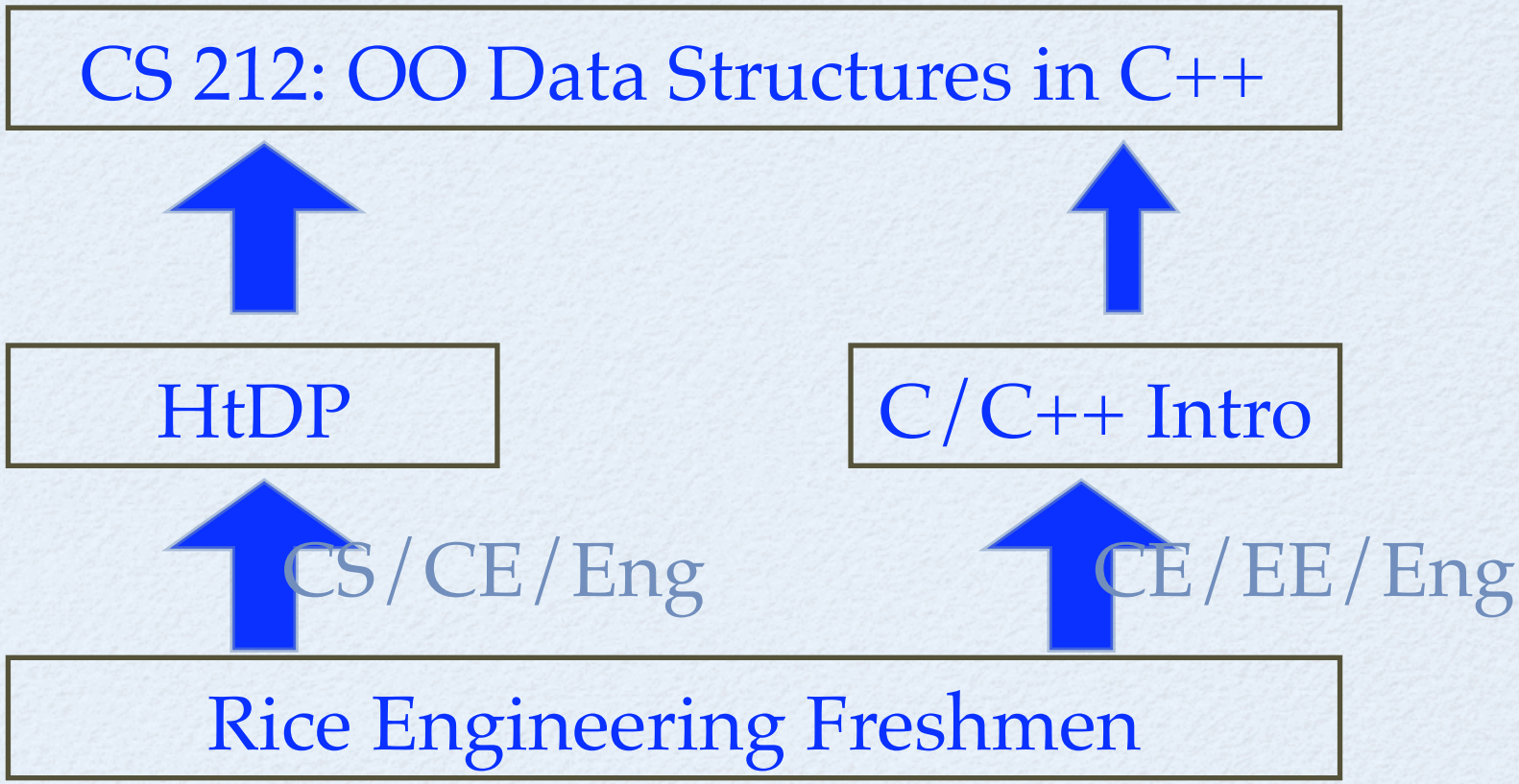
RESULTS: RICE



RESULTS: RICE



RESULTS: RICE

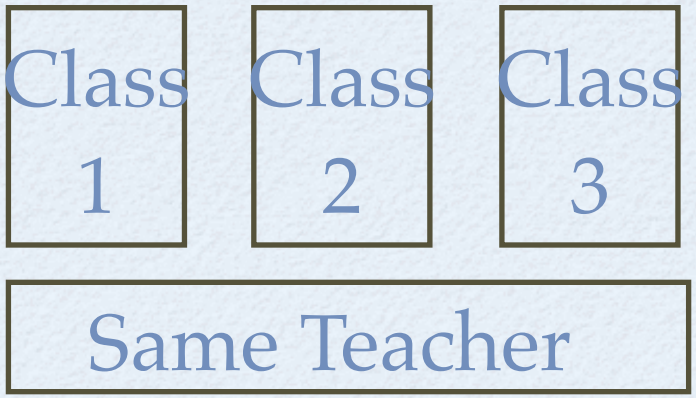


HtDP Students routinely outperform C/C++ students on C++

RESULTS: HIGH SCHOOL

Same Teacher

RESULTS: HIGH SCHOOL



RESULTS: HIGH SCHOOL

Same 2 Curricula

Class 1	Class 2	Class 3

Same Teacher

RESULTS: HIGH SCHOOL

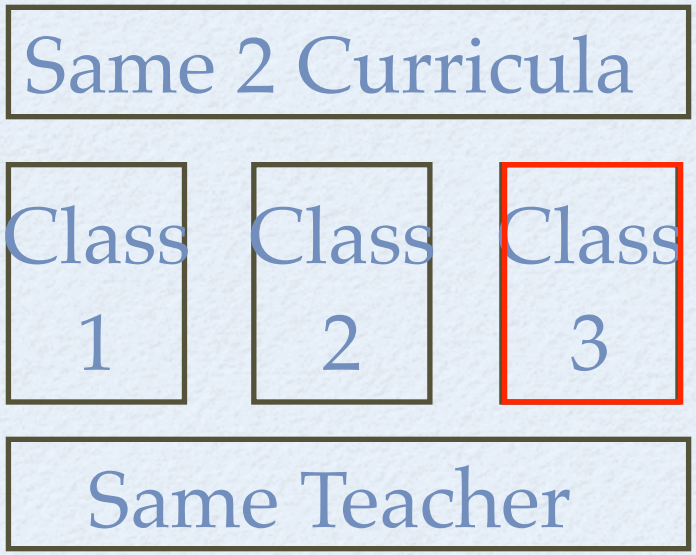
Same 2 Curricula

Class 1	Class 2	Class 3

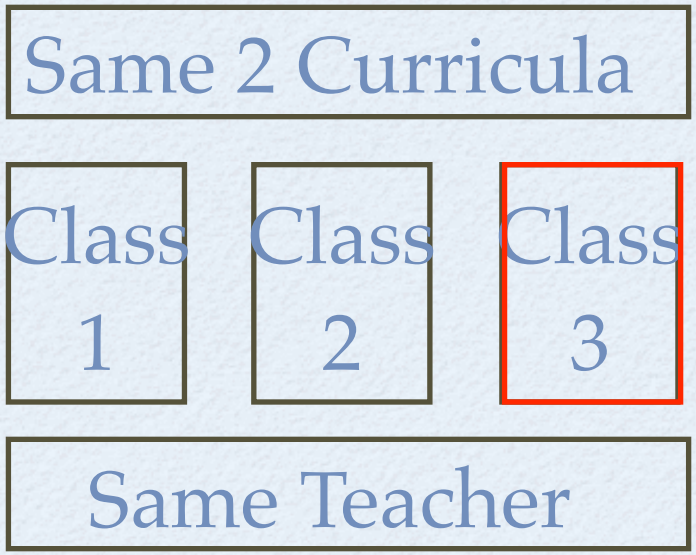
Same Teacher

RESULTS: HIGH SCHOOL

- All students: HtDP preferred by ~70%

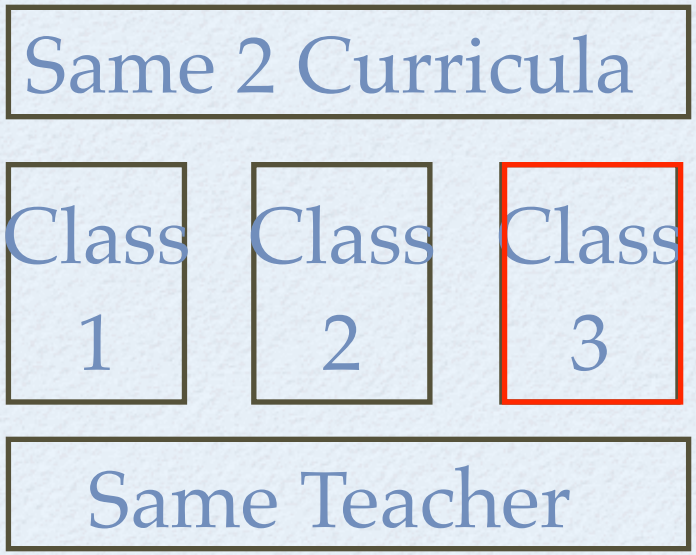


RESULTS: HIGH SCHOOL



- All students: HtDP preferred by ~70%
- The more C++, the more they prefer HtDP

RESULTS: HIGH SCHOOL



- All students: HtDP preferred by ~70%
- The more C++, the more they prefer HtDP
- Female students: prefer HtDP by a ratio over 4:1

RESULTS:SECOND SEMESTER

- The second semester: four full years at NU
- Three follow-up courses, 3 different instructors: ***all confirm that the students have made a quantum leap in programming performance.***
- Field tests: at a few high schools and colleges

RESULTS

Fall 2008: Bootcamp for MS

Conclusions, Future

FUTURE

	Programming	Mathematics
Semester 1	*How to Design Programs*	Discrete
Semester 2	*How to Design Classes*	*How to Prove Programs (ACL2)*

PRINCIPLES AND

Combining principles
with pragmatics in
the first year is

PRINCIPLES AND

Combining principles with pragmatics in the first year is

- ... feasible

PRINCIPLES AND

Combining principles with pragmatics in the first year is

- ... feasible
- ... effective

PRINCIPLES AND

Combining principles with pragmatics in the first year is

- ... feasible
- ... effective
- ... productive

PRINCIPLES AND

Combining principles with pragmatics in the first year is

- ... feasible
- ... effective
- ... productive
- It is the ***right*** thing!

What It *Really* Takes

Design Principles

What it *Really* takes

Design Principles

Series of PLs

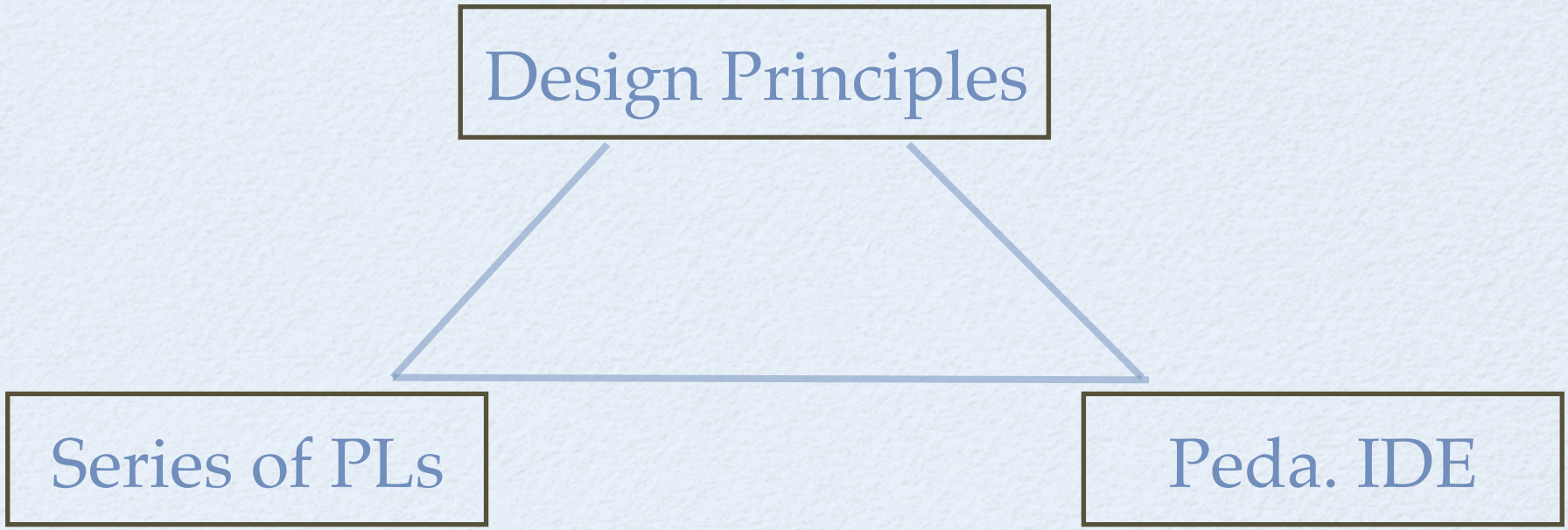
What it *Really* takes

Design Principles

Series of PLs

Peda. IDE

What it *Really* takes



What it *Really* takes

Design Principles

5 - 10 Years of Development Work
with between 3 and 20 people

Series of PLs

Peda. IDE

What it *Really* takes

Design Principles

Good Luck!

5 - 10 Years of Development Work
with between 3 and 20 people

Series of PLs

Peda. IDE

Thank You!

Matthew Flatt
Robert Findler
Shriram Krishnamurthi
Kathi Fisler
Kathy Gray
Viera Proulx
John Clements
and many, many more