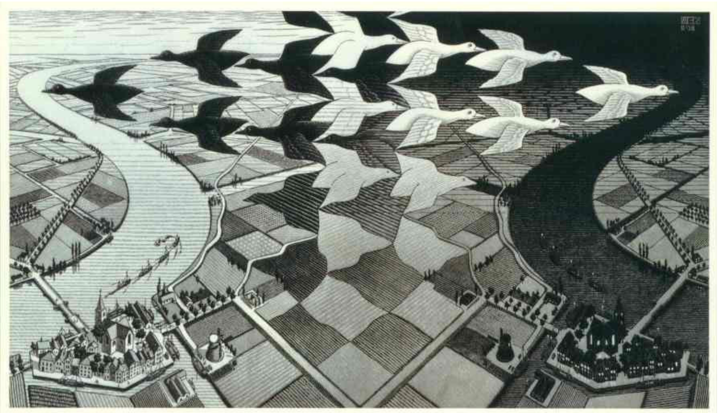


Lecture 18: Changing State

cs1120 Fall 2009
David Evans
http://www.cs.virginia.edu/evans

Sounds of Colossus: http://pixelh8.co.uk/discography/

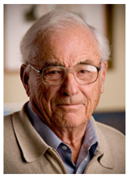
Menu

- Computing with Electrons
- The Story So Far
- Introduction Mutation

Yesterday's Nobel Prize in Physics

Charles K. Kao
Standard Telecommunication Laboratories, United Kingdom

fiberoptics: using light to transmit data



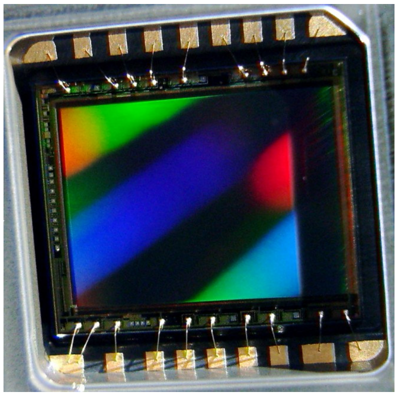
Willard S. Boyle

George E. Smith

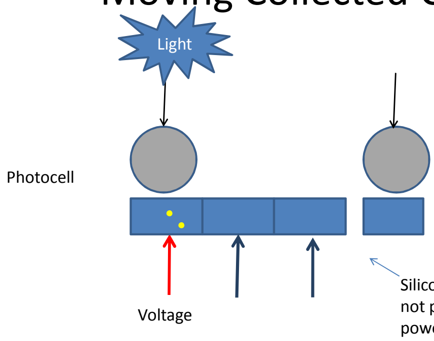
Bell Labs, New Jersey

Charge-Coupled Device (1969)

Charge-Coupled Device (CCD)



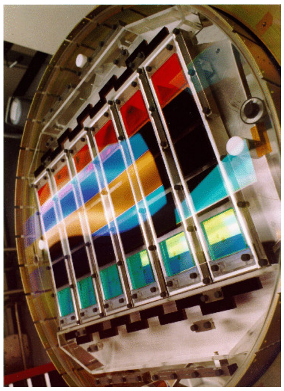
Moving Collected Charge



Photocell: photoelectric effect (Einstein's 1921 nobel prize): photons hit silicon plate and knock out electrons (more light, more electrons)

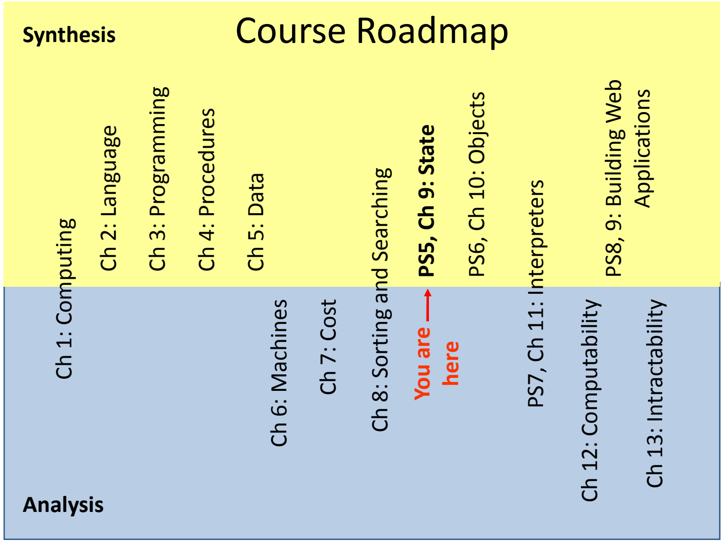
Silicon substrate:
not powered, conducts well
powered, does not conduct

CCDs Today



Sloan Digital Sky Survey (1998): array of 30, ~4Mpixel CCDs

cs1120 Story so Far



Course Roadmap

Synthesis

Analysis

Ch 1: Computing

Ch 2: Language

Ch 3: Programming

Ch 4: Procedures

Ch 5: Data

Ch 6: Machines

Ch 7: Cost

Ch 8: Sorting and Searching

You are here →

PS5, Ch 9: State

PS6, Ch 10: Objects

PS7, Ch 11: Interpreters

Ch 12: Computability

PS8, 9: Building Web Applications

Ch 13: Intractability

Computer Science: cs1120 so far

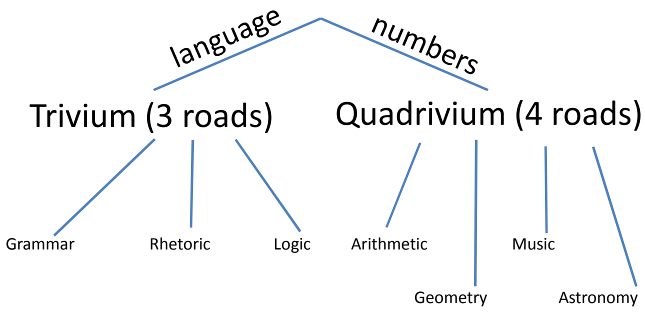
- How to describe information processes by defining procedures (Chapters 3, 4, 5)
  - Programming with procedures, lists, recursion
- How to predict properties about information processes (Chapter 6, 7)
  - Predicting how running time grows with input size
- How to efficiently implement information processes (not much on this)
  - Chapter 3 (rules of evaluation)
  - Chapter 6 (machines)

cs1120 Upcoming

- How to describe information processes by defining procedures
  - Programming with state (Ch 9), objects (Ch 10), languages (Ch 11)
- How to predict properties about information processes
  - Are there problems which can't be solved by algorithms? (Ch 12)
  - What is the fastest process that can solve a given problem? (Ch 13)
- How to efficiently implement information processes
  - How to implement a Scheme interpreter (Ch 11)

From Chapter 1/Lecture 1:

The Liberal Arts



language

numbers

Trivium (3 roads)

Quadrivium (4 roads)

Grammar

Rhetoric

Logic

Arithmetic

Geometry

Music

Astronomy

Liberal Arts Checkup

Trivium

- Grammar: study of meaning in written expression

  BNF, RTN, rules of evaluation for meaning

- Rhetoric: comprehension of verbal and written discourse

  Not much yet…interfaces between components (PS6-9), program and user (PS8-9)

- Logic: argumentative discourse for discovering truth

  Rules of evaluation, if, recursive definitions

Quadrivium

- Arithmetic: understanding numbers

  Not much yet… wait until November

- Geometry: quantification of space

  Curves as procedures, fractals (PS3)

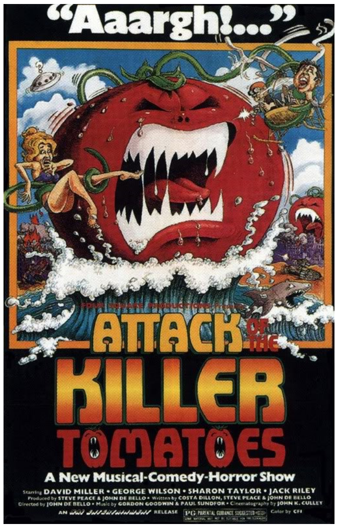
- Music: number in time

  Yes, listen to "Hey Jude!"

- Astronomy

  Soon: read Neil deGrasse Tyson's essay

Introducing Mutation


"Aaargh!..."
ATTACK OF THE KILLER TOMATOES
A New Musical-Comedy-Horror Show

From Lecture 3:

Evaluation Rule 2: Names

A *name* expression evaluates to the value associated with that name.

```
> (define two 2)
> two
2
```

This has been more-or-less okay so far, since the value associated with a name never changes...

Names and Places

- A name is not just a value, it is a **place** for storing a value.
- **define** creates a new place, associates a name with that place, and stores a value in that place

(define x 3)   x: 3

Bang!

set! ("set bang") changes the value associated with a place

```
> (define x 3)
> x
3
> (set! x 7)
> x
7
```

x: 7

set! should make you nervous

```
> (define x 2)
> (nextx)
3
> (nextx)
4
> x
4
```

Before **set!** all procedures were functions (except for some with side-effects). The value of (f) was the same every time you evaluate it. Now it might be different!

Defining nextx

```
(define (nextx)
 (set! x (+ x 1))
 x)
```

syntactic sugar for

```
(define nextx
  (lambda ()
    (begin
      (set! x (+ x 1))
      x))))
```

Evaluation Rules

```
> (define x 3)
> (+ (nextx) x)
7
or 8
> (+ x (nextx))
9
or 10
```

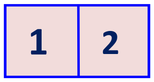
DrScheme evaluates application subexpressions left to right, but Scheme evaluation rules allow any order.

Mutable Cons Cell

mcons – creates a mutable cons cell

(mcar m) – first part of a mutable cons cell

(mcdr m) – second part of a mutable cons cell



(mcons 1 2)

set-mcar! and **set-mcdr!**

(set-mcar! *p v*)

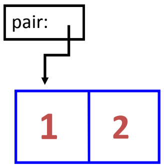
Replaces the car of mutable cons *p* with *v*.

(set-mcdr! *p v*)

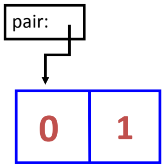
Replaces the cdr of mutable cons *p* with *v*.

These should scare you even more then set!!

```
> (define pair (mcons 1 2))
> pair
(1 . 2)
```



```
> (define pair (mcons 1 2))
> pair
(1 . 2)
> (set-mcar! pair 0)
> (mcar pair)
0
> (mcdr pair)
2
> (set-mcdr! pair 1)
> pair
(0 . 1)
```



Impact of Mutation

- We will need to revise our evaluation rules for names and application expressions: substitution model of evaluation no longer works since values associated with names **change**
- We need to be much more careful in our programs to think about **when** things happen: order matters since values change

Charge

- PS5: posted now, due ~~next Wednesday~~

  Monday, 19 October

WAHOO! Auctions

- Read Chapter 9
- Friday: return Exam 1, Revising our Evaluation Rules to handle mutation