Coding Theory

CSE 445/545

January 28, 2019

Let’s do some introductions

Atri Rudra

319 Davis Hall
atri@buffalo.edu
645-2464
Office hours: Tue, 2-2:45pm

Handouts for today

Syllabus

Linked from the course webpage

Feedback polls

Up on piazza

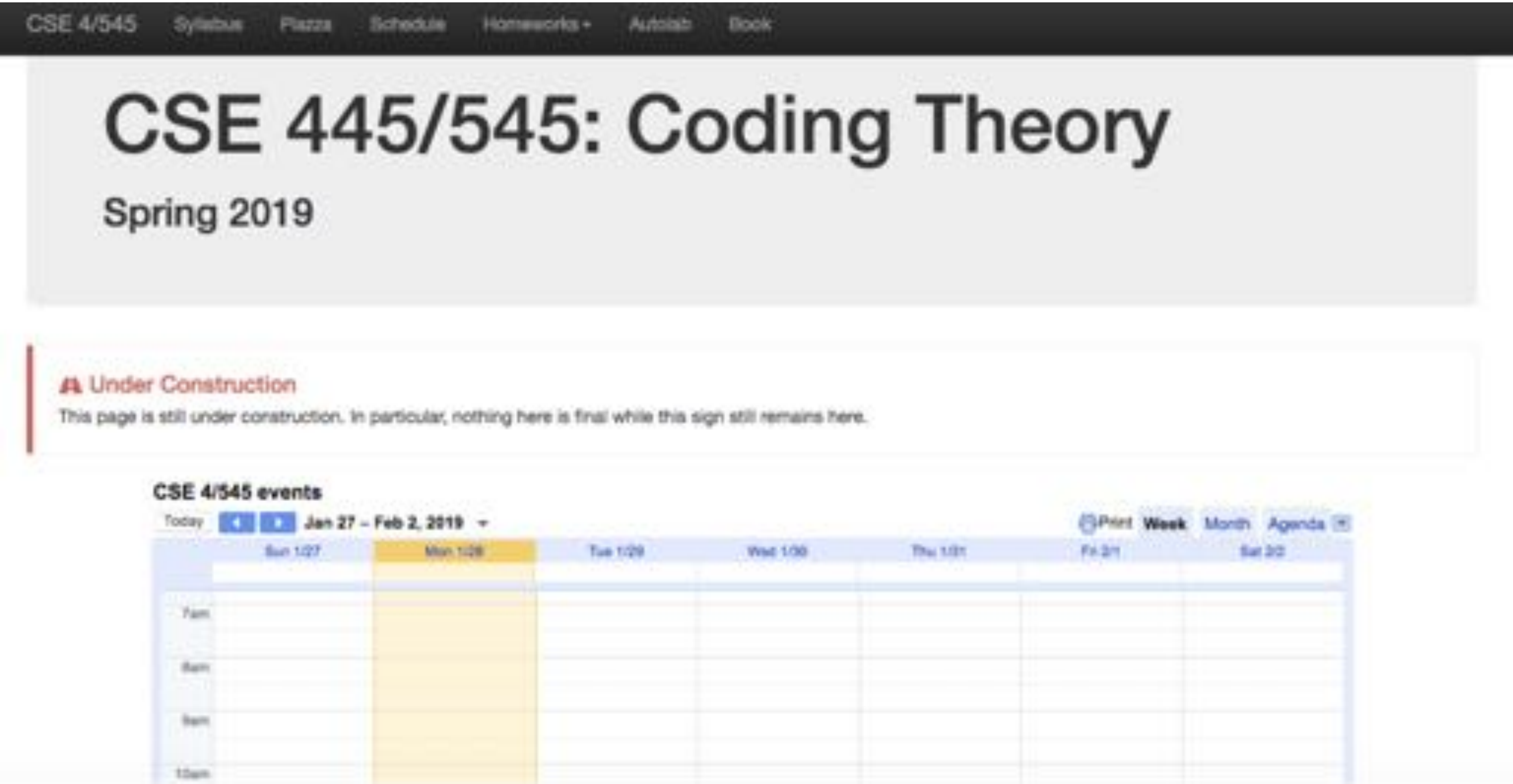
Plug for feedback polls

Completing the form is voluntary & anonymous

Purpose of the form

- For me to get an idea of your technical background

One Stop Shop for the course



CSE 445/545: Coding Theory

Spring 2019

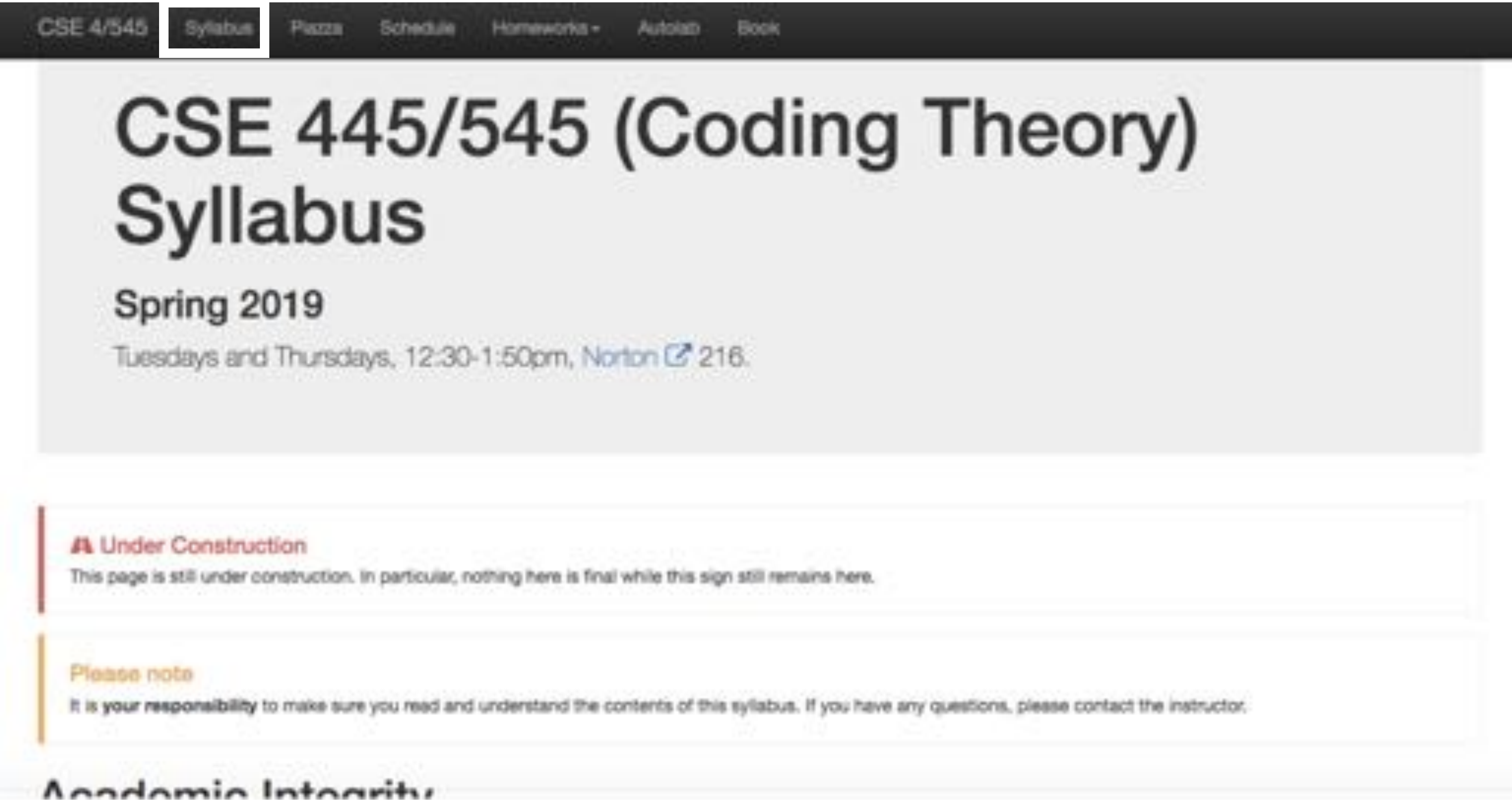
Under Construction

This page is still under construction. In particular, nothing here is final while this sign still remains here.

CSE 4/545 events

https://cse.buffalo.edu/faculty/atri/courses/coding-theory/webpage/spr19/

Syllabus



CSE 445/545 (Coding Theory) Syllabus

Spring 2019

Tuesdays and Thursdays, 12:30-1:50pm, Norton 216.

Under Construction

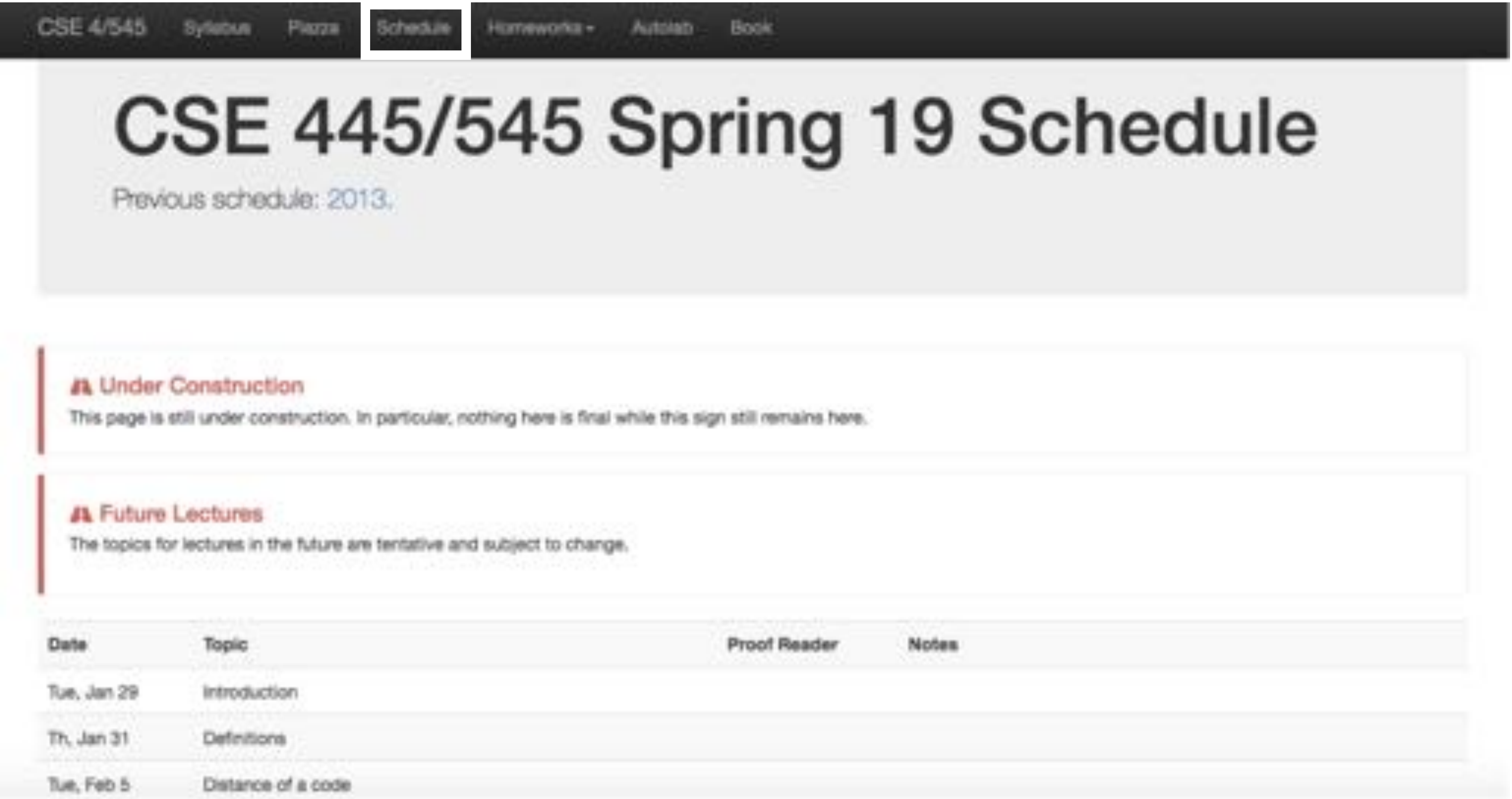
This page is still under construction. In particular, nothing here is final while this sign still remains here.

Please note

It is **your responsibility** to make sure you read and understand the contents of this syllabus. If you have any questions, please contact the instructor.

Academic Integrity

Schedule



CSE 445/545 Spring 19 Schedule

Previous schedule: 2013.

Under Construction

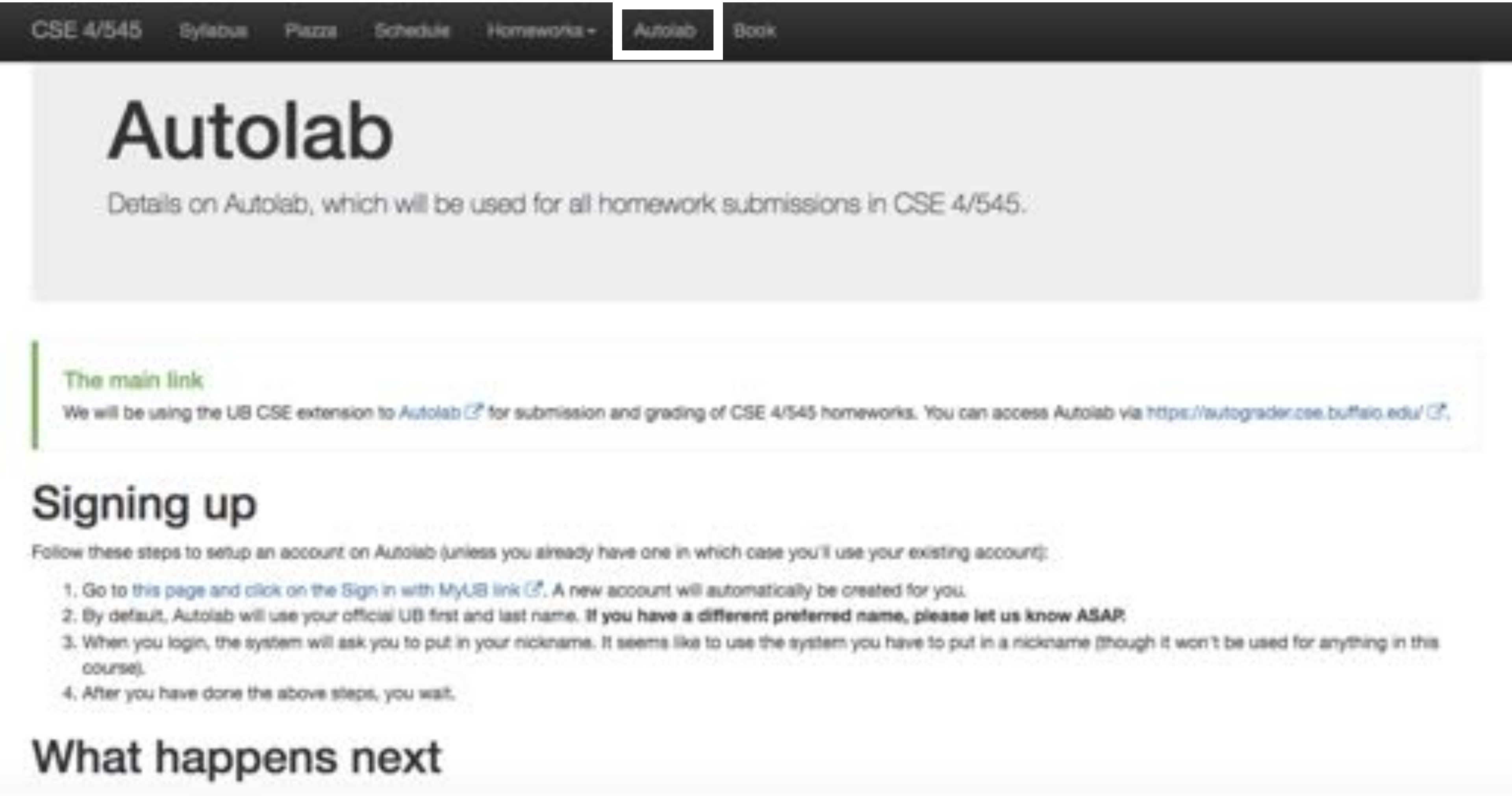
This page is still under construction. In particular, nothing here is final while this sign still remains here.

Future Lectures

The topics for lectures in the future are tentative and subject to change.

Date	Topic	Proof Reader	Notes
Tue, Jan 29	Introduction		
Th, Jan 31	Definitions		
Tue, Feb 5	Distance of a code		

Autolab



Autolab

Details on Autolab, which will be used for all homework submissions in CSE 4/545.

The main link

We will be using the UB CSE extension to Autolab for submission and grading of CSE 4/545 homeworks. You can access Autolab via https://autograder.cse.buffalo.edu/.

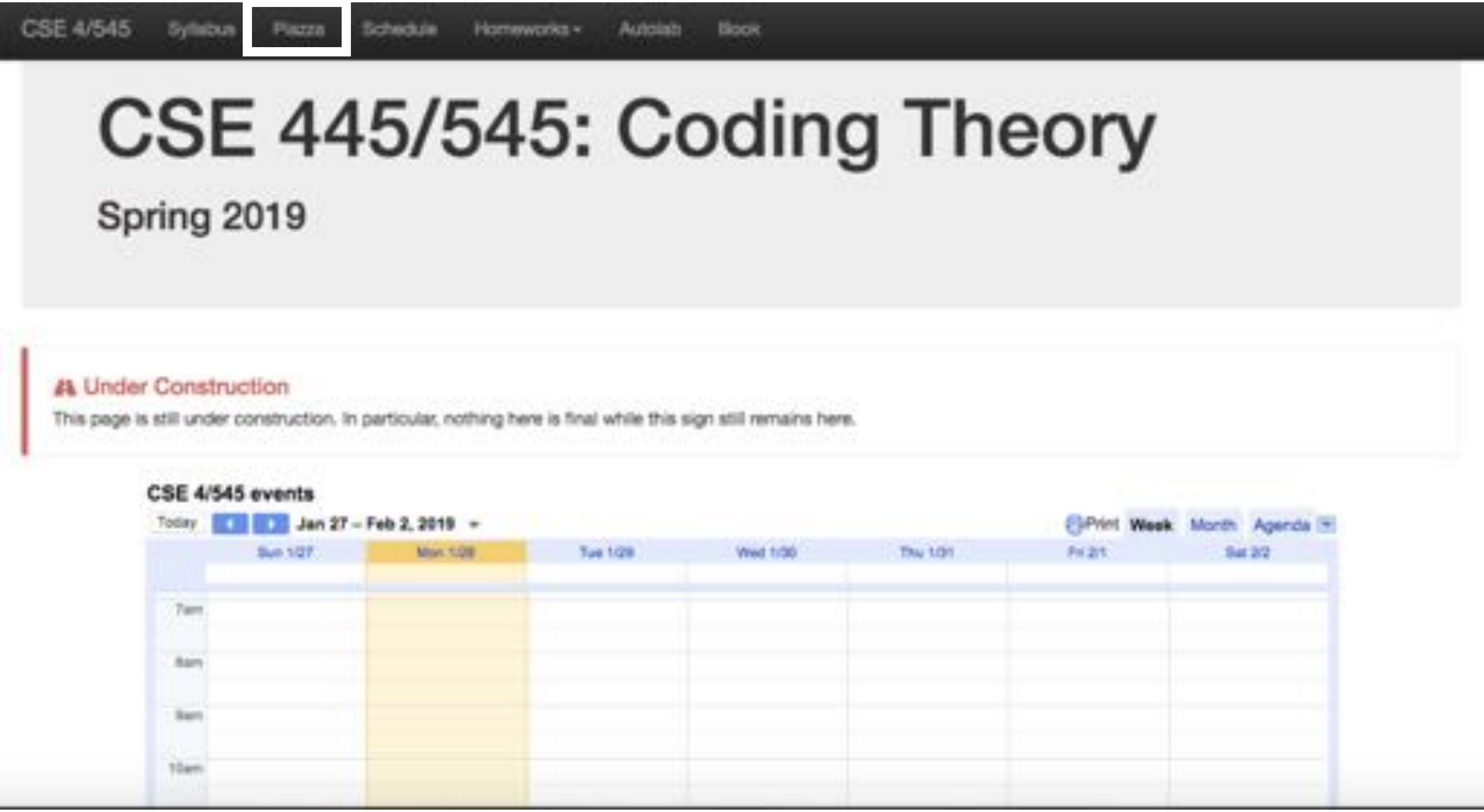
Signing up

Follow these steps to setup an account on Autolab (unless you already have one in which case you'll use your existing account):

1. Go to this page and click on the Sign in with MyUB link. A new account will automatically be created for you.
2. By default, Autolab will use your official UB first and last name. **If you have a different preferred name, please let us know ASAP.**
3. When you login, the system will ask you to put in your nickname. It seems like to use the system you have to put in a nickname (though it won't be used for anything in this course).
4. After you have done the above steps, you wait.

What happens next

Piazza



CSE 445/545: Coding Theory

Spring 2019

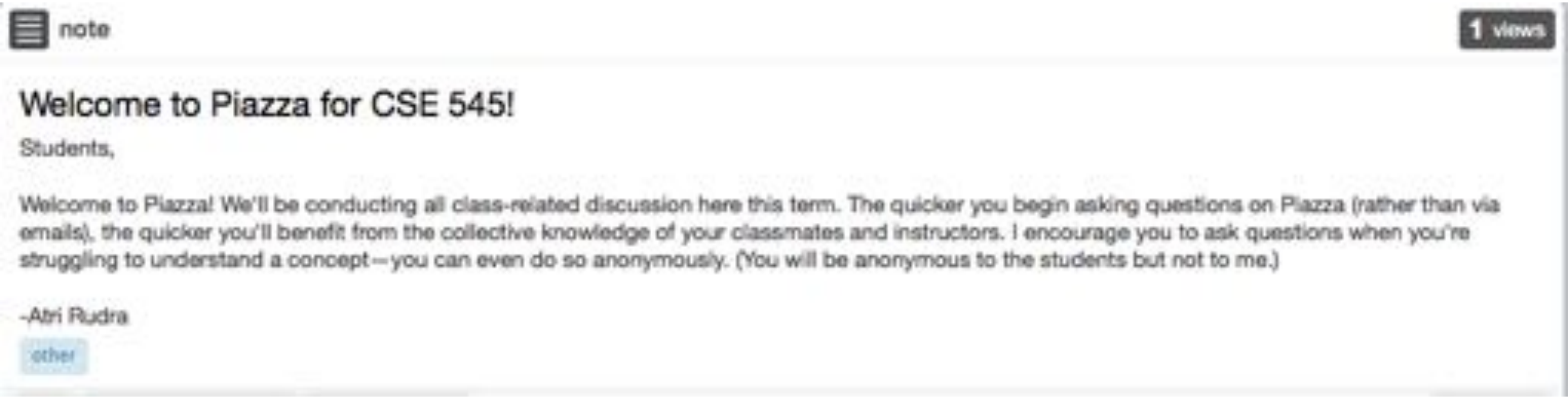
Under Construction

This page is still under construction. In particular, nothing here is final while this sign still remains here.

CSE 4/545 events

Piazza for discussion

Please use your UB email ID to sign up



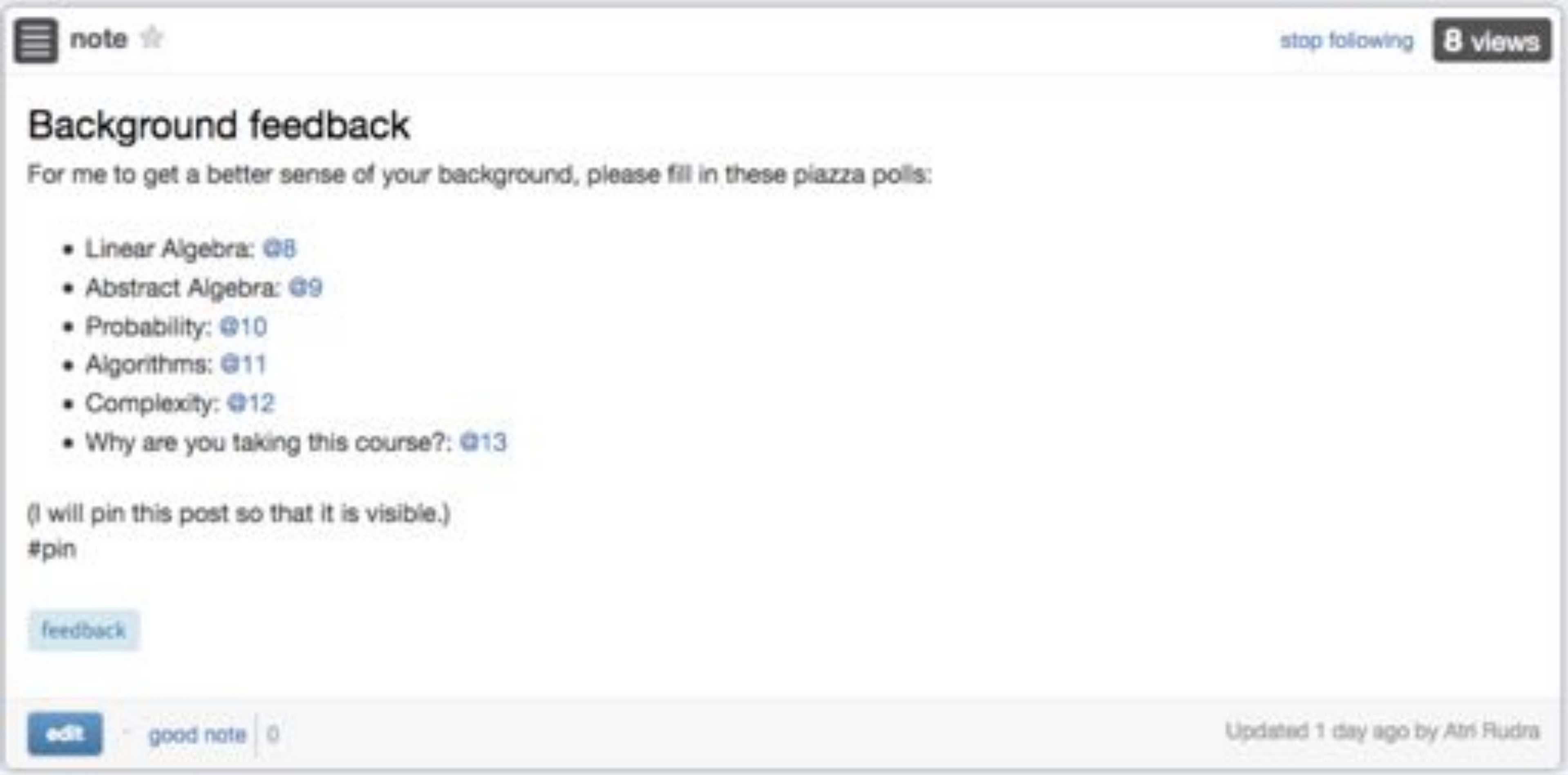
Welcome to Piazza for CSE 545!

Students,

Welcome to Piazza! We'll be conducting all class-related discussion here this term. The quicker you begin asking questions on Piazza (rather than via emails), the quicker you'll benefit from the collective knowledge of your classmates and instructors. I encourage you to ask questions when you're struggling to understand a concept—you can even do so anonymously. (You will be anonymous to the students but not to me.)

-Atri Rudra

Feedback polls already up



Background feedback

For me to get a better sense of your background, please fill in these piazza polls:

- Linear Algebra: @8
- Abstract Algebra: @9
- Probability: @10
- Algorithms: @11
- Complexity: @12
- Why are you taking this course?: @13

(I will pin this post so that it is visible.)
#pin

Updated 1 day ago by Atri Rudra

Questions/Comments?

If something doesn’t work (e.g. you cannot post a comment), let me know

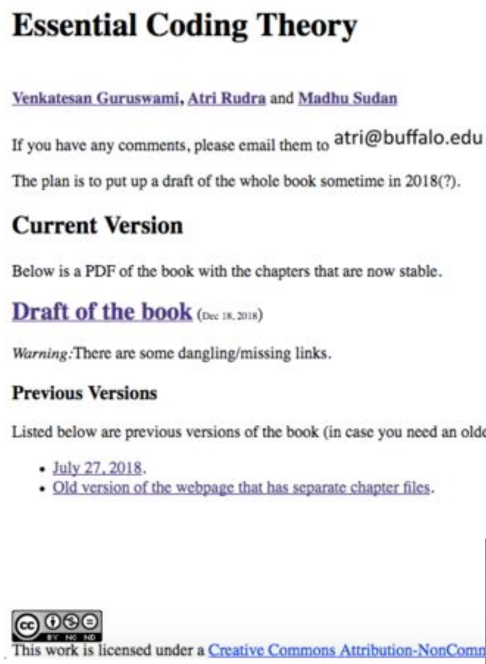
References

Draft of a book I’m writing

- With Guruswami+Sudan

Standard coding theory texts

- MacWilliams and Sloane
- van Lint
- Blahut
- Handbook of coding theory



Essential Coding Theory

Venkatesan Guruswami, Atri Rudra and Madhu Sudan

If you have any comments, please email them to atri@buffalo.edu

The plan is to put up a draft of the whole book sometime in 2018(?).

Current Version

Below is a PDF of the book with the chapters that are now stable.

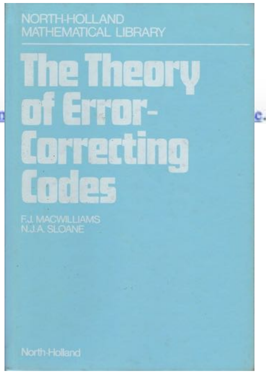
Draft of the book (Dec 18, 2018)

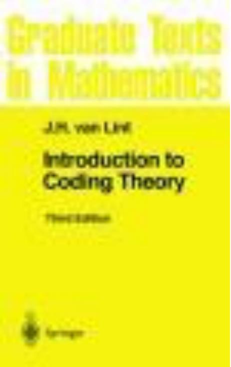
Warning: There are some dangling/missing links.

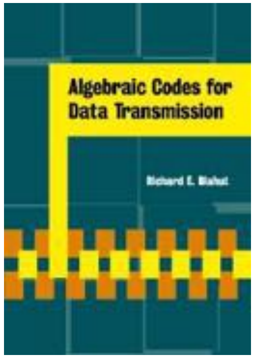
Previous Versions

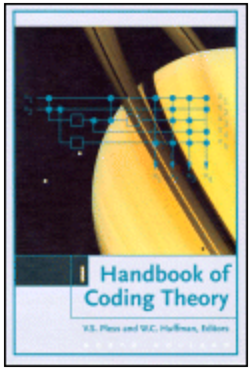
Listed below are previous versions of the book (in case you need an older version):

- July 27, 2018.
- Old version of the webpage that has separate chapter files.


NORTH-HOLLAND MATHEMATICAL LIBRARY
The Theory of Error-Correcting Codes
F.J. MACWILLIAMS
N.J.A. SLOANE
North-Holland






Handbook of Coding Theory
V.S. Pless and W.C. Huffman, Editors

Pre-requisites

No formal pre-requisites for 545/ CSE 331 for 445

- Probably no one will have all the pre-req’ s

Mathematical maturity

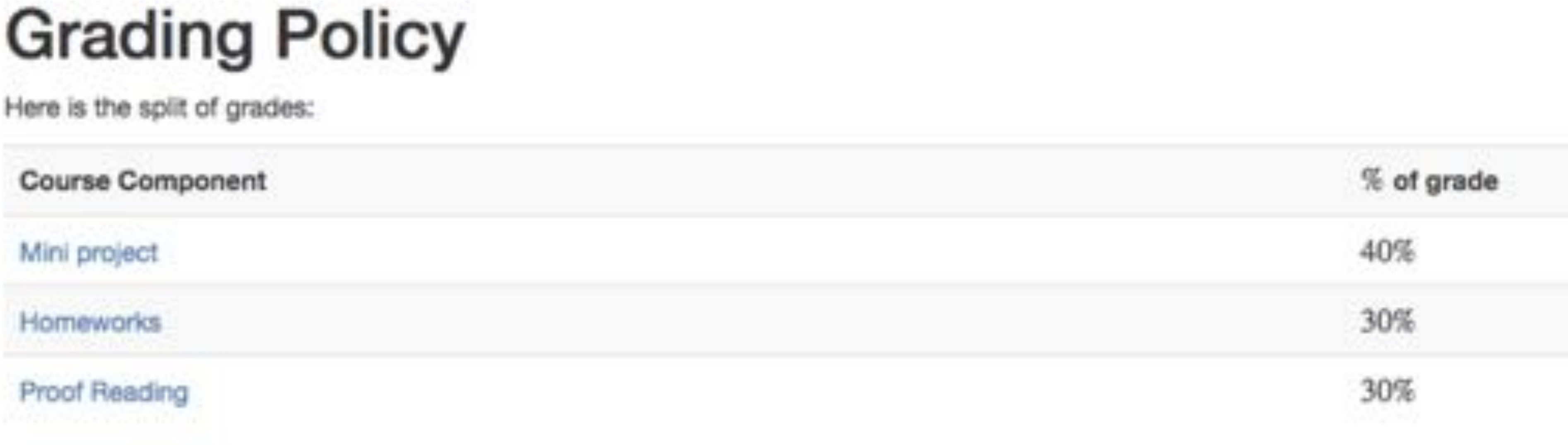
- Comfortable with proofs
- Willing to pick up basics of new areas

Will spend one lecture on the pre-req’s

- Linear Algebra
- Finite Fields
- Probability
- Algorithms/ Asymptotic Analysis

Go slower in the first half of the course

Grades and such like



Grading Policy

Here is the split of grades:

Course Component	% of grade
Mini project	40%
Homeworks	30%
Proof Reading	30%

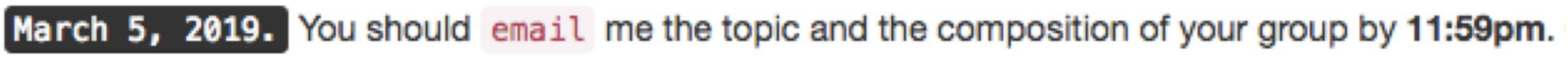
Mini Project

Groups of size <= 3

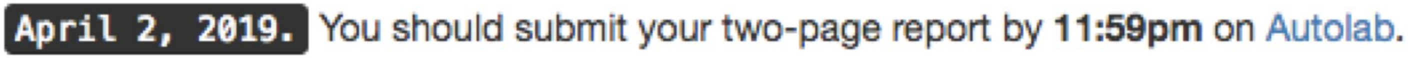
Create a Youtube video related to coding theory

Bunch of other details in syllabus

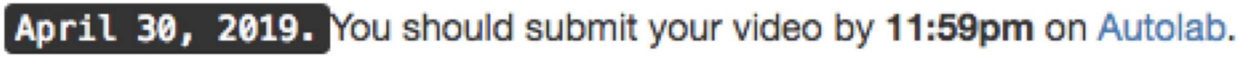
Deadlines



`March 5, 2019.` You should `email` me the topic and the composition of your group by **11:59pm**.



`April 2, 2019.` You should submit your two-page report by **11:59pm** on Autolab.



`April 30, 2019.` You should submit your video by **11:59pm** on Autolab.

Proof-reading

Proof-read relevant part of the book

3-4 during the course

- Depends on the class strength

Submit typos, suggestions for improvement

They are due in by noon before next lecture

Notes will be graded on timeliness & quality

Will ask for a volunteer

See syllabus for more details

Questions/Comments?

Check out the syllabus for more details

Homework

3 short ones (545)/ 2 short ones (445)

Collaboration generally allowed

- Work in groups of size at most 3
- Write up your own solutions
- Acknowledge your collaborators
- No source other than book and your notes
- Breaking these rules will be considered as cheating

More details when they are handed out

My homework philosophy for 545

NOT to make sure you understand what I teach in the lectures

Homework problems either

- Proofs that were not done in the class; or
- Material that is not covered in the class
    - Closely related to something that is

Questions/Comments?

Check out the syllabus for more details

Some comments

- Decide on a Video topic **early**
  - Different topics might need different prep. work
  - Come talk to me
- Homeworks might take time
  - Do not wait for the last moment

Academic Dishonesty

All your submissions must be your own work

Penalty:

- Minimum: An **grade reduction in course**
- Possible: **F** (or higher penalty) if warranted

YOUR responsibility to know what is cheating, plagiarism etc.

- If not sure, come talk to me

Excuses like “I have a job,” “This was OK earlier/in my country,” “This course is hard,” etc. **WON’T WORK**

I DO NOT HAVE ANY PATIENCE WITH ANY CHEATING :

YOU WILL GET A GRADE REDUCTION IN THE COURSE

FOR YOUR FIRST MISTAKE

If grades are all you care about

You’ll be fine if

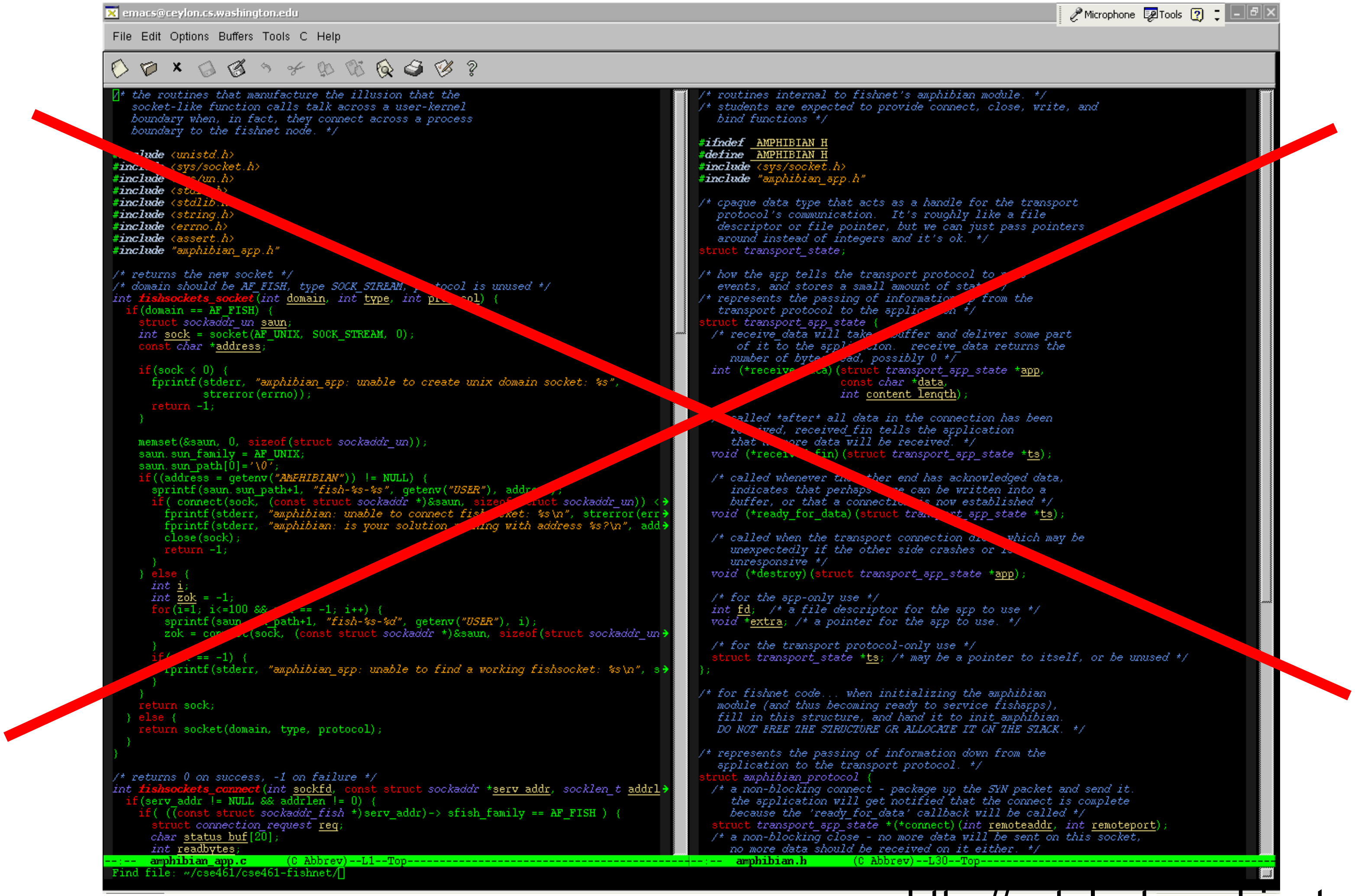
- You do your assignments **honestly**
- Make a reasonable attempt at them

Questions/Comments?

Check out the syllabus for more details

Let the fun begin!

Coding theory



```c
/* the routines that manufacture the illusion that the
   socket-like function calls talk across a user-kernel
   boundary when, in fact, they connect across a process
   boundary to the fishnet node. */

#include <unistd.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <assert.h>
#include "amphibian_app.h"

/* returns the new socket */
/* domain should be AF_FISH, type SOCK_STREAM, protocol is unused */
int fishsockets_socket(int domain, int type, int protocol) {
  if(domain == AF_FISH) {
    struct sockaddr_un saun;
    int sock = socket(AF_UNIX, SOCK_STREAM, 0);
    const char *address;

    if(sock < 0) {
      fprintf(stderr, "amphibian_app: unable to create unix domain socket: %s",
              strerror(errno));
      return -1;
    }

    memset(&saun, 0, sizeof(struct sockaddr_un));
    saun.sun_family = AF_UNIX;
    saun.sun_path[0]='\0';
    if((address = getenv("AMPHIBIAN")) != NULL) {
      sprintf(saun.sun_path+1, "fish-%s-%s", getenv("USER"), address);
      if( connect(sock, (const struct sockaddr *)&saun, sizeof(struct sockaddr_un)) <
        fprintf(stderr, "amphibian: unable to connect fishsocket: %s\n", strerror(err
        fprintf(stderr, "amphibian: is your solution running with address %s?\n", add
        close(sock);
        return -1;
      }
    } else {
      int i;
      int zok = -1;
      for(i=1; i<=100 && zok == -1; i++) {
        sprintf(saun.sun_path+1, "fish-%s-%d", getenv("USER"), i);
        zok = connect(sock, (const struct sockaddr *)&saun, sizeof(struct sockaddr_un
      }
      if(zok == -1) {
        fprintf(stderr, "amphibian_app: unable to find a working fishsocket: %s\n", s
      }
    }
    return sock;
  } else {
    return socket(domain, type, protocol);
  }
}

/* returns 0 on success, -1 on failure */
int fishsockets_connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrl
  if(serv_addr != NULL && addrlen != 0) {
    if( ((const struct sockaddr_fish *)serv_addr)-> sfish_family == AF_FISH ) {
      struct connection_request req;
      char status_buf[20];
      int readbytes;
```

```
--:--  amphibian_app.c      (C Abbrev)--L1--Top------------------------------------
Find file: ~/cse461/cse461-fishnet/
```

```c
/* routines internal to fishnet's amphibian module. */
/* students are expected to provide connect, close, write, and
   bind functions */

#ifndef _AMPHIBIAN_H
#define _AMPHIBIAN_H
#include <sys/socket.h>
#include "amphibian_app.h"

/* cpaque data type that acts as a handle for the transport
   protocol's communication.  It's roughly like a file
   descriptor or file pointer, but we can just pass pointers
   around instead of integers and it's ok. */
struct transport_state;

/* how the app tells the transport protocol to pass
   events, and stores a small amount of state */
/* represents the passing of information up from the
   transport protocol to the application */
struct transport_app_state {
  /* receive_data will take a buffer and deliver some part
     of it to the application.  receive_data returns the
    number of bytes read, possibly 0 */
  int (*receive_data)(struct transport_app_state *app,
                      const char *data,
                      int content_length);

  /* called *after* all data in the connection has been
     received, received_fin tells the application
     that no more data will be received. */
  void (*received_fin)(struct transport_app_state *ts);

  /* called whenever the other end has acknowledged data,
     indicates that perhaps more can be written into a
     buffer, or that a connection is now established */
  void (*ready_for_data)(struct transport_app_state *ts);

  /* called when the transport connection dies, which may be
     unexpectedly if the other side crashes or is
     unresponsive */
  void (*destroy)(struct transport_app_state *app);

  /* for the app-only use */
  int fd;  /* a file descriptor for the app to use */
  void *extra; /* a pointer for the app to use. */

  /* for the transport protocol-only use */
  struct transport_state *ts; /* may be a pointer to itself, or be unused */
};

/* for fishnet code... when initializing the amphibian
   module (and thus becoming ready to service fishapps),
   fill in this structure, and hand it to init_amphibian.
   DO NOT FREE THE STRUCTURE OR ALLOCATE IT ON THE STACK. */

/* represents the passing of information down from the
   application to the transport protocol. */
struct amphibian_protocol {
  /* a non-blocking connect - package up the SYN packet and send it.
     the application will get notified that the connect is complete
     because the 'ready_for_data' callback will be called */
  struct transport_app_state *(*connect)(int remoteaddr, int remoteport);
  /* a non-blocking close - no more data will be sent on this socket,
     no more data should be received on it either. */
```

```
--:--  amphibian.h      (C Abbrev)--L30--Top------------------------------------------
```

http://catalyst.washington.edu/

What does this say?

W*lcome to the cl*ss. I h*pe you w*ll h*ve as mu*h f*n as I wi*l hav* t*ach*ng it!

Welcome to the class. I hope you will have as much fun as I will have teaching it!

Why did the example work?

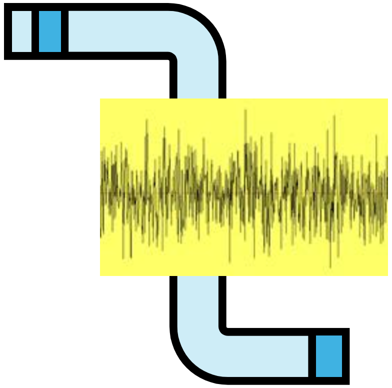
English has in built redundancy

Can tolerate “errors”

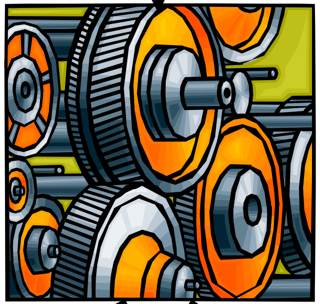
The setup

x

C(x)



y = C(x)+error



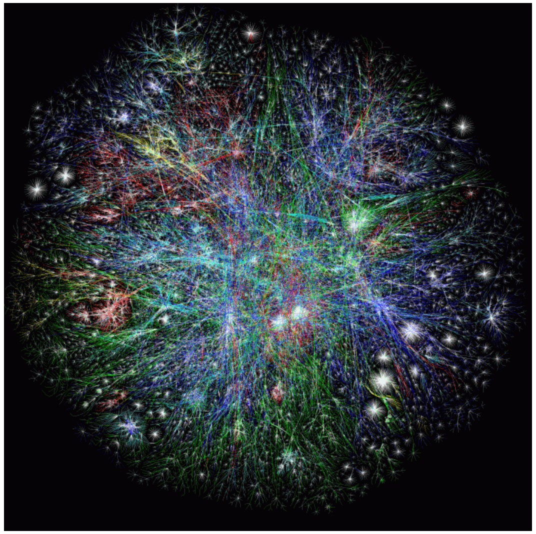
x

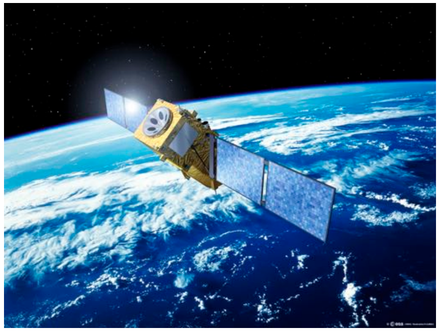
Give up

- Mapping **C**
  - Error-correcting code or just code
  - Encoding: x → C(x)
  - Decoding: y → x
  - C(x) is a codeword

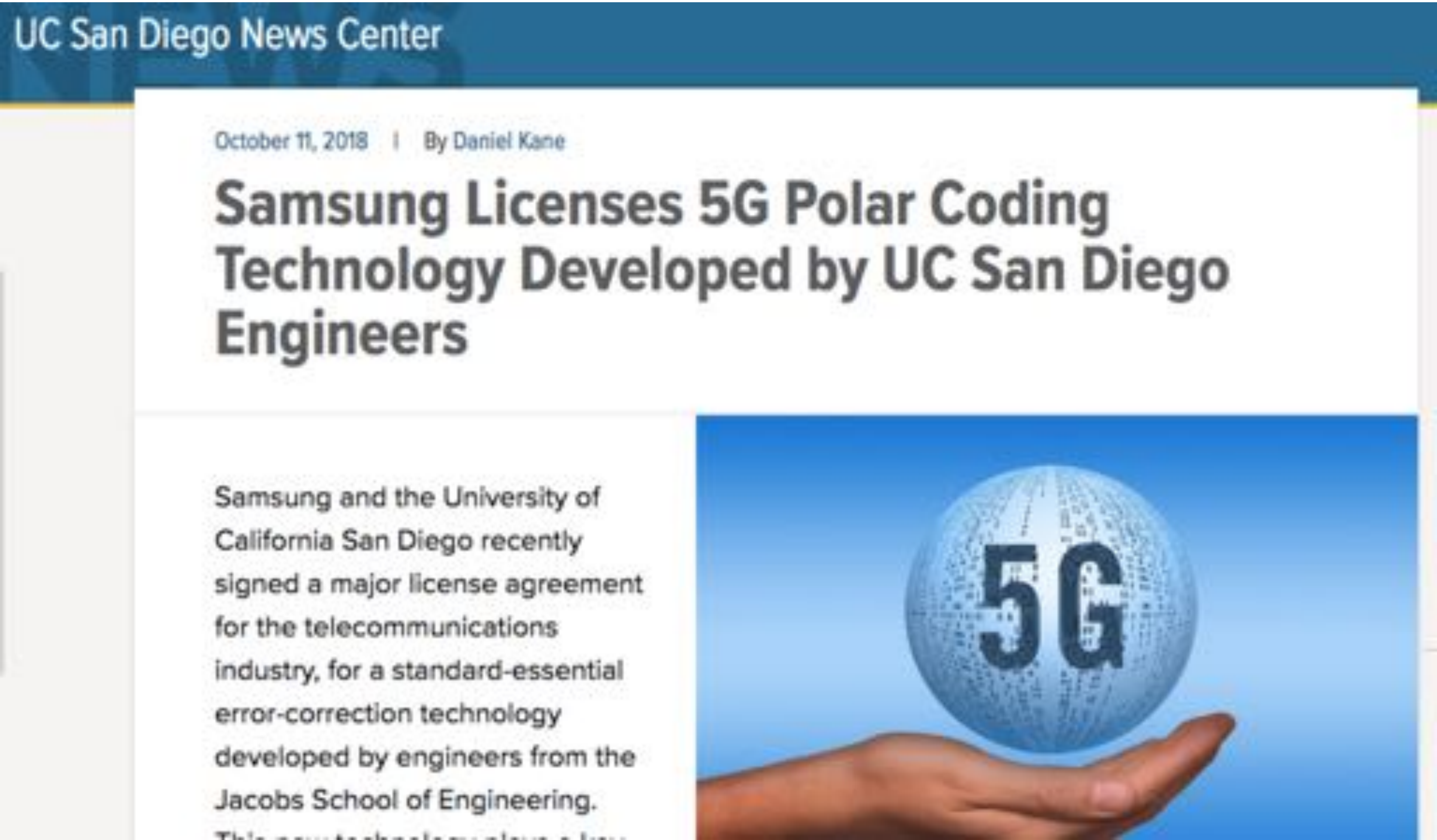
Communication

- Internet
  - Checksum used in multiple layers of TCP/IP stack
- Cell phones
- Satellite broadcast
  - TV
- Deep space telecommunications
  - Mars Rover





Codes and 5G



UC San Diego News Center

October 11, 2018 | By Daniel Kane

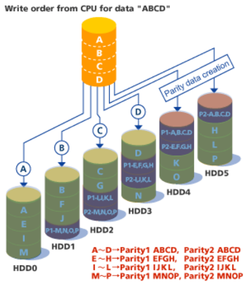
Samsung Licenses 5G Polar Coding Technology Developed by UC San Diego Engineers

Samsung and the University of California San Diego recently signed a major license agreement for the telecommunications industry, for a standard-essential error-correction technology developed by engineers from the Jacobs School of Engineering.

“Unusual” applications

Data Storage

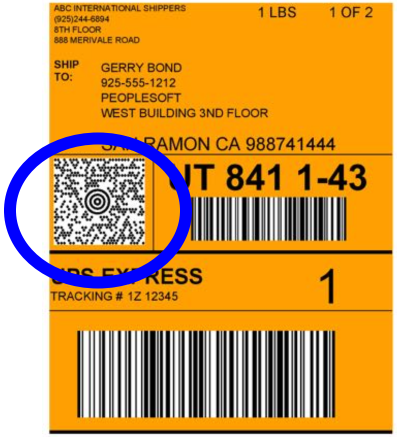
- CDs and DVDs
- RAID
- ECC memory



Write order from CPU for data "ABCD"
A
B
C
D
Parity data creation
A
B
C
D
A
E
I
M
HDD0
B
F
J
P1-M,N,O,P
HDD1
C
G
P1-I,J,K,L
P2-M,N,O,P
HDD2
D
P1-E,F,G,H
P2-I,J,K,L
N
HDD3
P1-A,B,C,D
P2-E,F,G,H
K
O
HDD4
P2-A,B,C,D
H
L
P
HDD5
A～D→Parity1 ABCD, Parity2 ABCD
E～H→Parity1 EFGH, Parity2 EFGH
I～L→Parity1 IJKL, Parity2 IJKL
M～P→Parity1 MNOP, Parity2 MNOP

Paper bar codes

- UPS (MaxiCode)



ABC INTERNATIONAL SHIPPERS
(925)244-6894
8TH FLOOR
888 MERIVALE ROAD

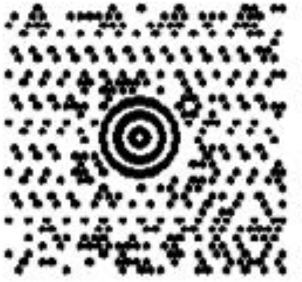
1 LBS 1 OF 2

SHIP TO:
GERRY BOND
925-555-1212
PEOPLESOFT
WEST BUILDING 3ND FLOOR
SAN RAMON CA 988741444

UT 841 1-43

UPS EXPRESS
TRACKING # 1Z 12345

1



Codes are all around us

Other applications of codes

- Outside communication/storage domain
- Tons of applications in theory
  - Complexity Theory
  - Cryptography
  - Algorithms

Coding theory is a good tool to have in your arsenal

The birth of coding theory

Claude E. Shannon

“A Mathematical Theory of Communication”

1948

Gave birth to Information theory

EE 634
(this semester!)

Richard W. Hamming

“Error Detecting and Error Correcting Codes”

1950

Structure of the course

Part I: Combinatorics

What can and cannot be done with codes

Part II: Algorithms

How to use codes efficiently

Part III: Applications

Applications in (theoretical) Computer Science

Redundancy vs. Error-correction

Repetition code: Repeat every bit say 100 times

- Good error correcting properties
- Too much redundancy

Parity code: Add a parity bit

- Minimum amount of redundancy
- Bad error correcting properties
  - Two errors go completely undetected

1 1 1 0 0	1

1 0 0 0 0	1

Neither of these codes are satisfactory

Two main challenges in coding theory

Problem with parity example

- Messages mapped to codewords which do not differ in many places

Need to pick a lot of codewords that differ a lot from each other

Efficient decoding

- Naive algorithm: check received word with all codewords

The fundamental tradeoff

Correct as many errors as possible with as little redundancy as possible

Can one achieve the “optimal” tradeoff with *efficient* encoding and decoding ?