Coming Soon to Android...

Reed Solomon Error Correction

The Educational App

Reed Solomon Educational App

- Why do you do it?
- Historical Background
- How do you do it?
- Why do **we** do it?
- How do **we** do it?
- What have we done so far?

Why do you do it?


Avoid This

Example Applications Include...


Delete
PgUp
Home
PgDn
End
9
PgUp
6
Enter


Microsoft Azure

Example Applications Include...

Historical Background

Historical Background

J Soc. Indust. Appl. Math.
Vol. 8, No. 2, June, 1960
Printed in U.S.A.

POLYNOMIAL CODES OVER CERTAIN FINITE FIELDS*†

I. S. REED and G. SOLOMON‡

Introduction. A code is a mapping from a vector space of dimension m over a finite field K (denoted by $V_m(K)$) into a vector space of higher dimension $n > m$ over the same field ($V_n(K)$). K is usually taken to be the field of two elements Z_2, in which case it is a mapping of m-tuples of binary digits (bits) into n-tuples of binary digits. If one transmits n bits, the additional $n - m$ bits are "redundant" and allow one to recover the original message in the event that noise corrupts the signal during transmission and causes some bits of the code to be in error. A multiple-error-correcting code of order s consists of a code which maps m-tuples of zeros and ones into n-tuples of zeros and ones, where m and n both depend on s, and a decoding procedure which recovers the message completely, assuming no more than s errors occur during transmission in the vector of n bits. The Hamming code [1] is an example of a systematic one bit error-correcting code. We present here a new class of redundant codes along with a decoding procedure.

Let K be a field of degree n over the field of two elements Z_2. K contains 2^n elements. Its multiplicative group is cyclic and is generated by powers of α where α is the root of a suitable irreducible polynomial over Z_2. We discuss here a code E which maps m-tuples of K into 2^n-tuples of K.

Consider the polynomial $P(x)$ of degree $m - 1$

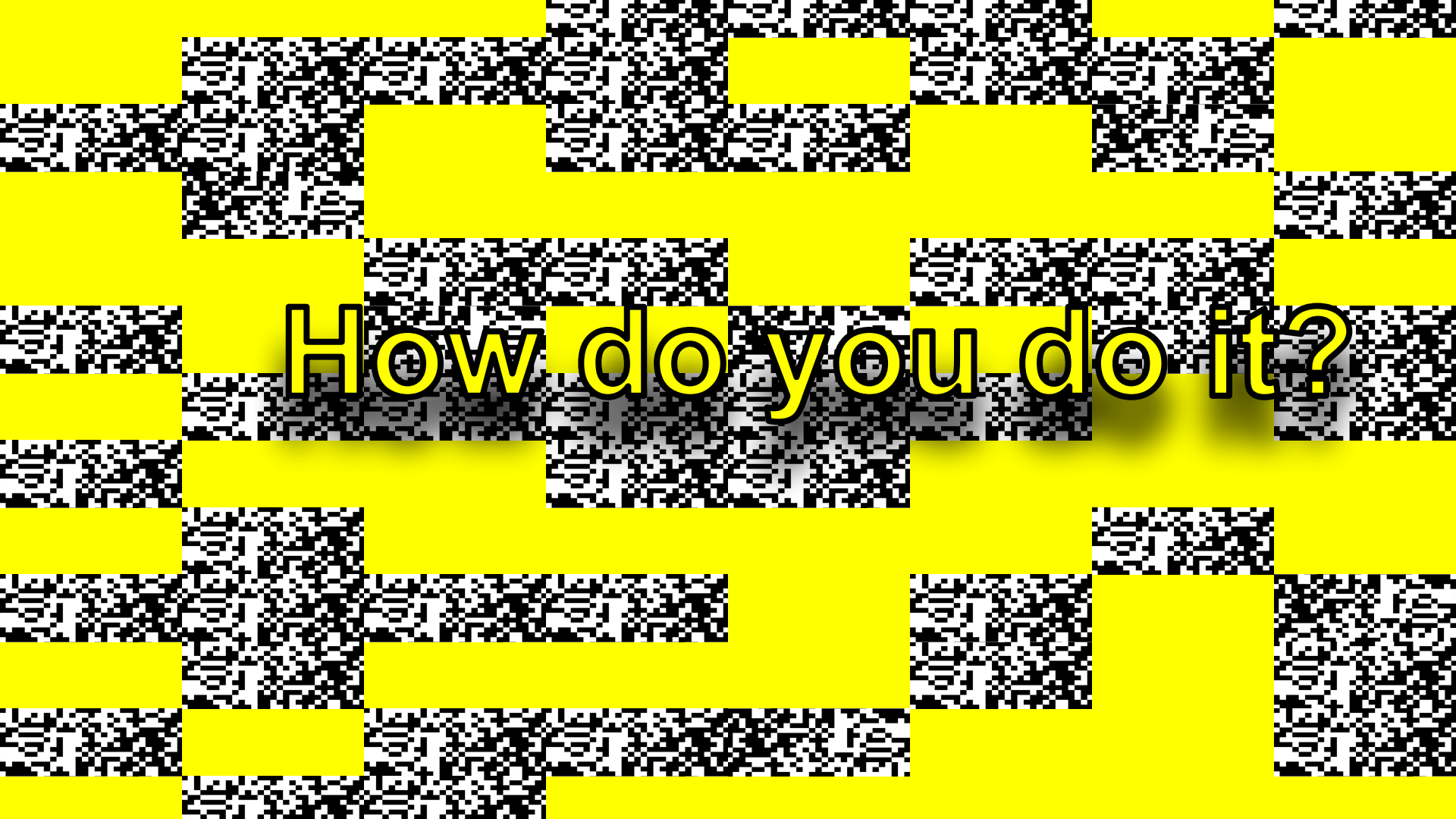
$$P(x) = a_0 + a_1 x + \cdots + a_{m-1} x^{m-1},$$

where $a_i \in K$ and $m < 2^n$. Code E is the mapping of the m-tuple $(a_0, a_1, \cdots, a_{m-1})$ into the 2^n-tuple $(P(0), P(\alpha), P(\alpha^2), \cdots, P(1))$; this m-tuple might be some encoded message and the corresponding 2^n-tuple is to be transmitted. This mapping of m symbols into 2^n symbols will be shown to be $(2^n - m)/2$ or $(2^n - m - 1)/2$ symbol correcting, depending on whether m is even or odd.

A natural correspondence is established between the field elements of K and certain binary sequences of length n. Under this correspondence, code E may be regarded as a mapping of binary sequences of mn bits into binary sequences of $n2^n$ bits. Thus code E can be interpreted to be a systematic multiple-error-correcting code of binary sequences.

* Received by the editors January 21, 1959 and in revised form August 26, 1959.
† The work reported here was performed at Lincoln Laboratory, a technical center operated by Massachusetts Institute of Technology with the joint support

Historical Background


How do you do it?

(A lot of math)

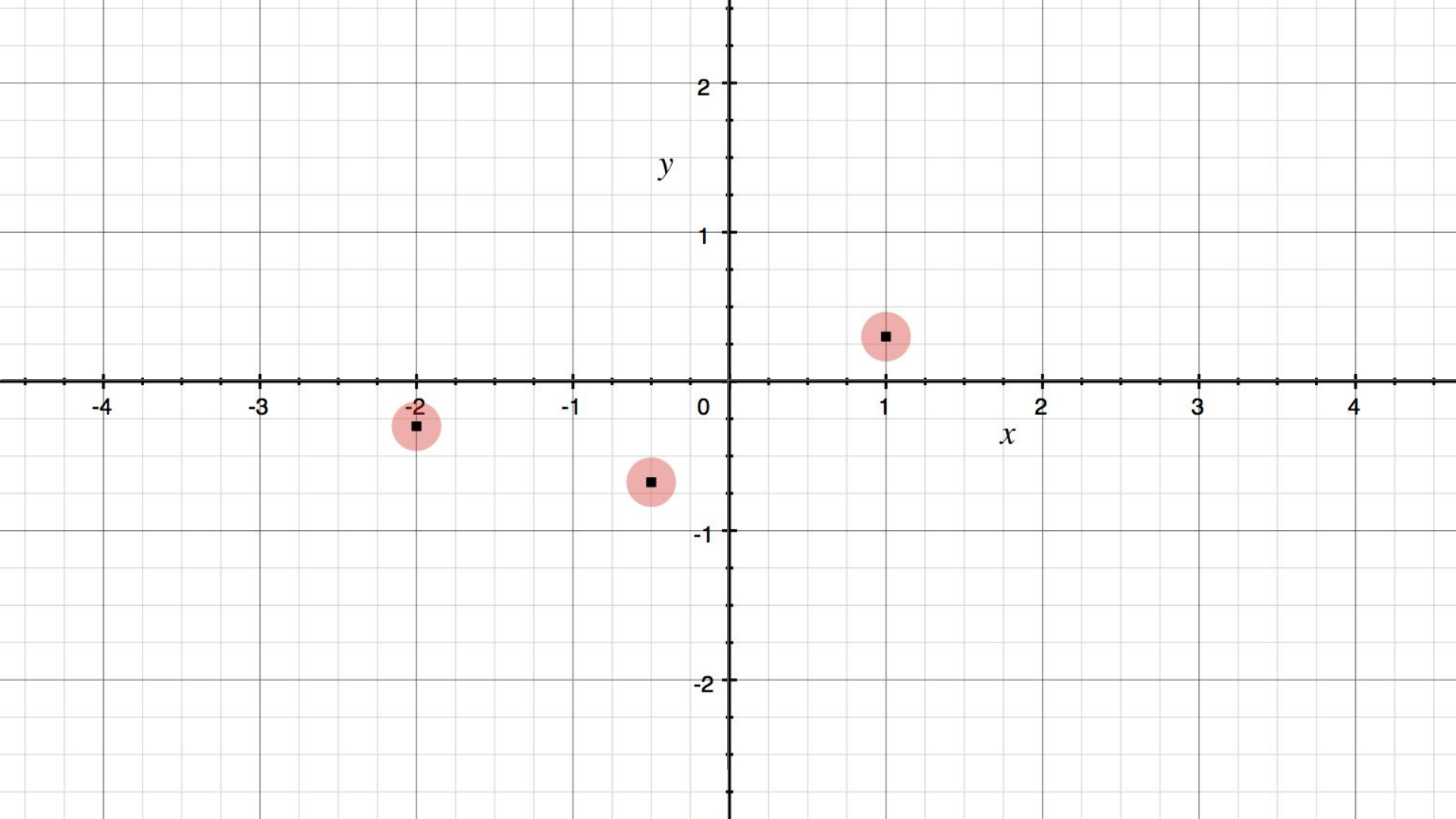
How do you do it?

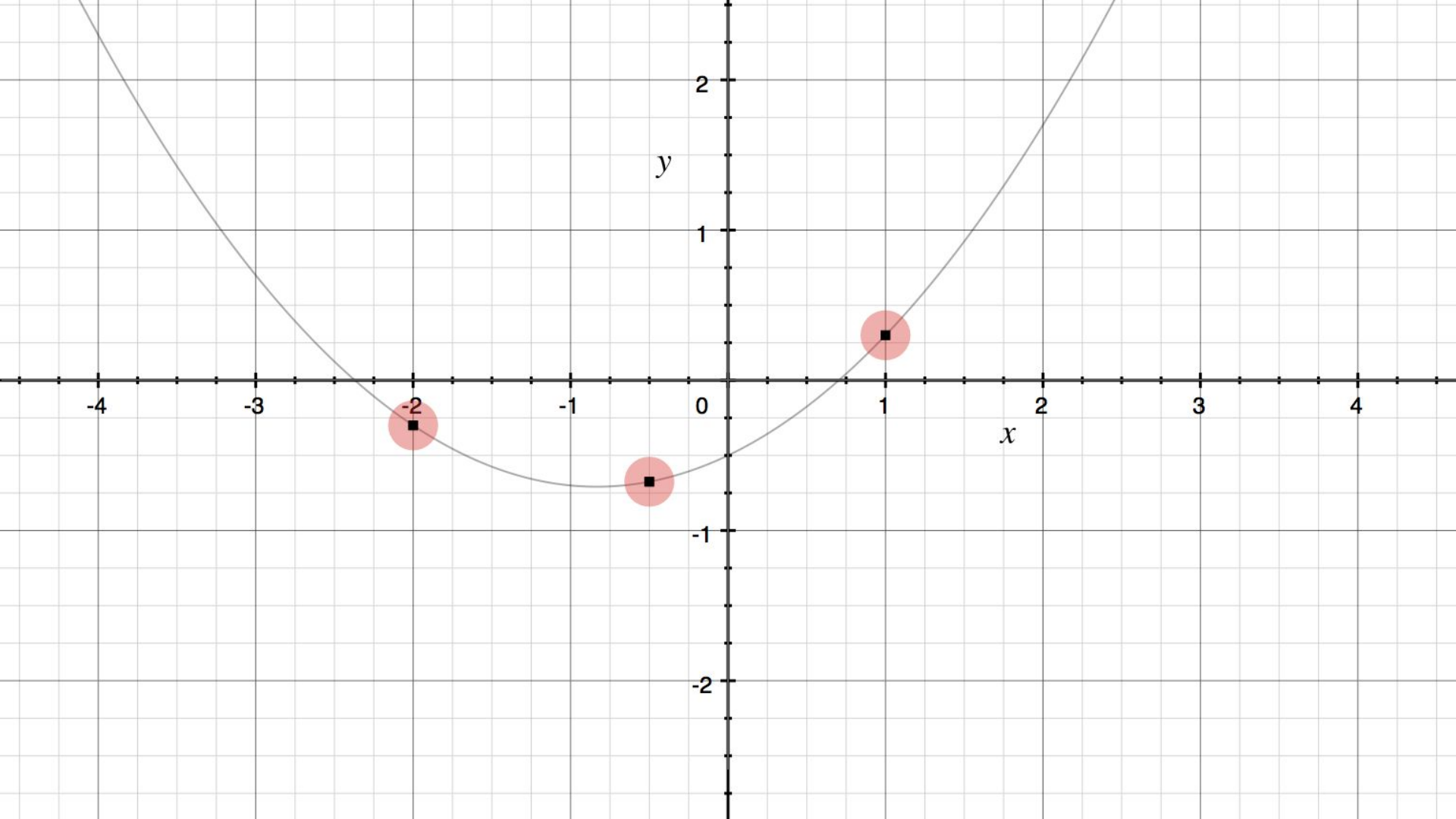
- *Sending evaluations of message polynomial*

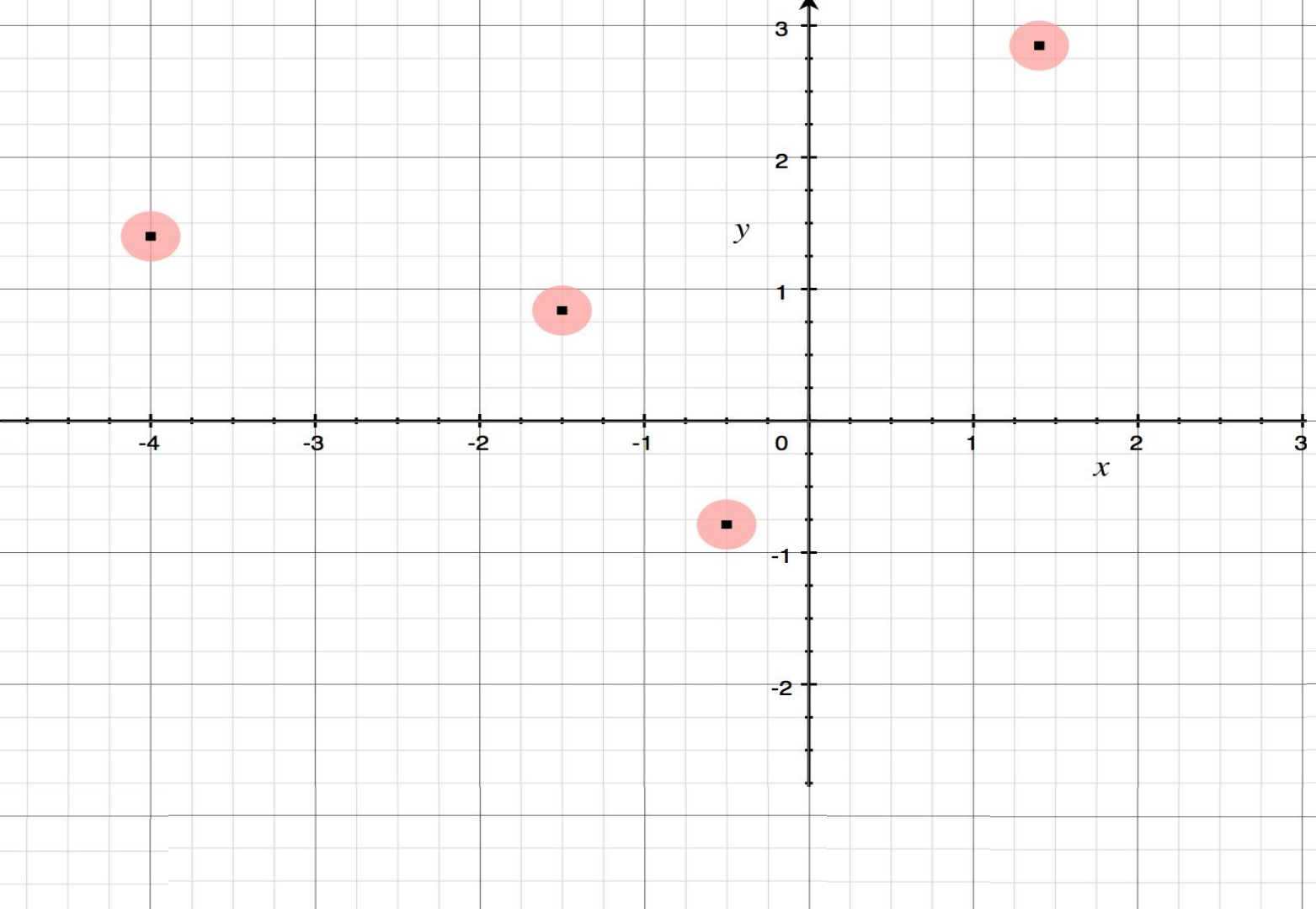
- *You can reconstruct a polynomial from the points you receive*

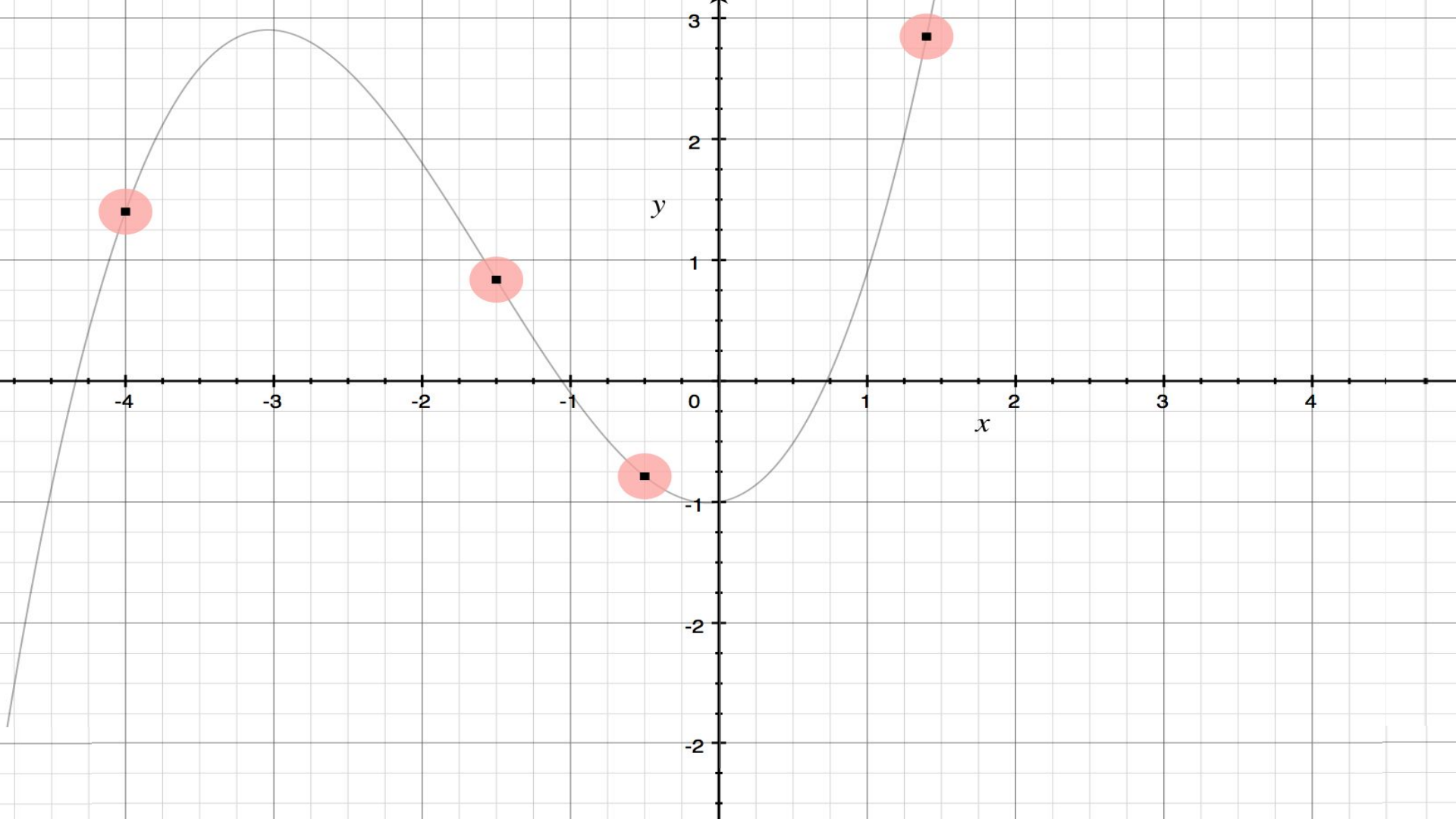
How do you do it?

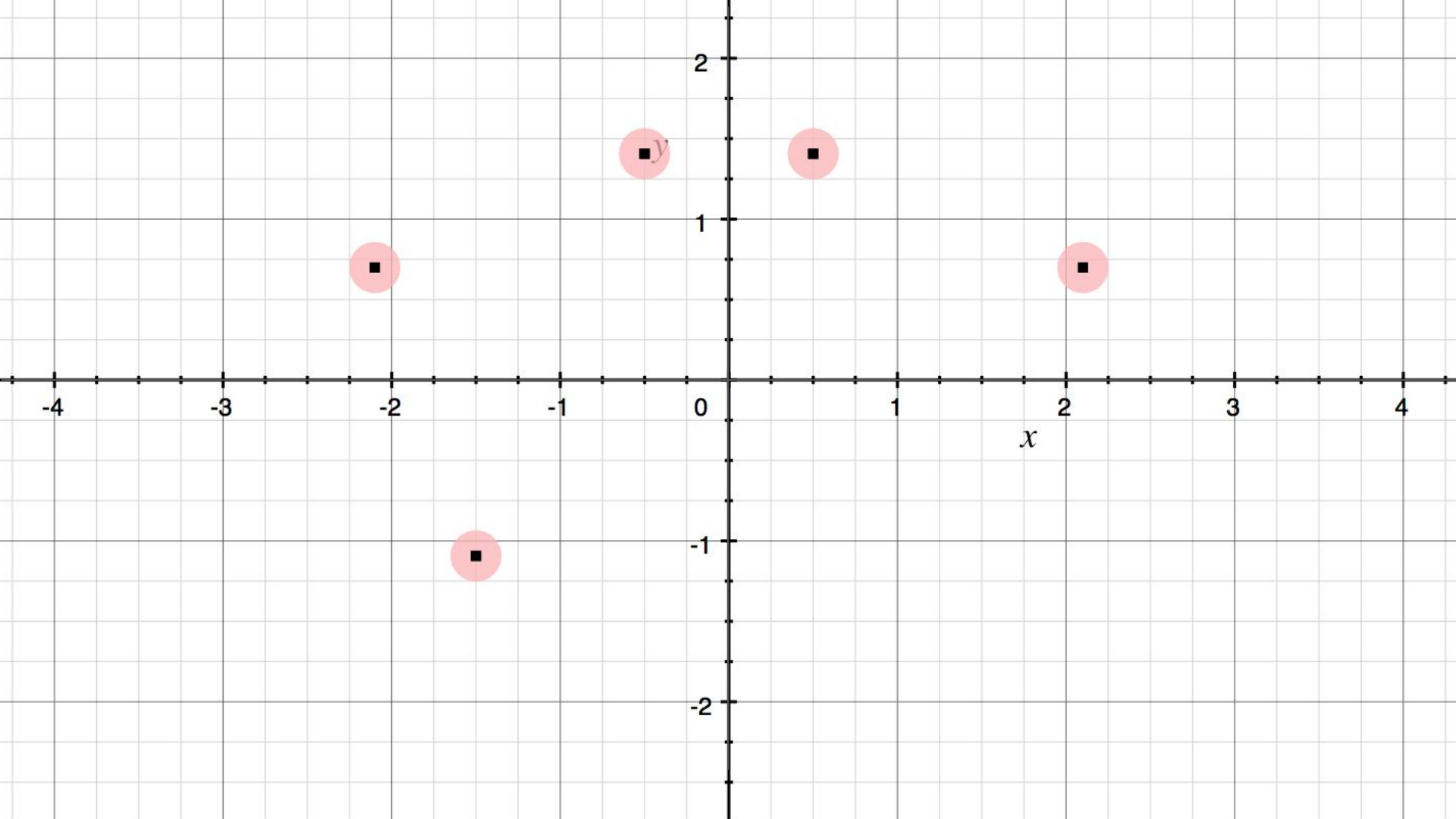
"Recall that there is exactly one polynomial of degree $k - 1$ that passes through k points"

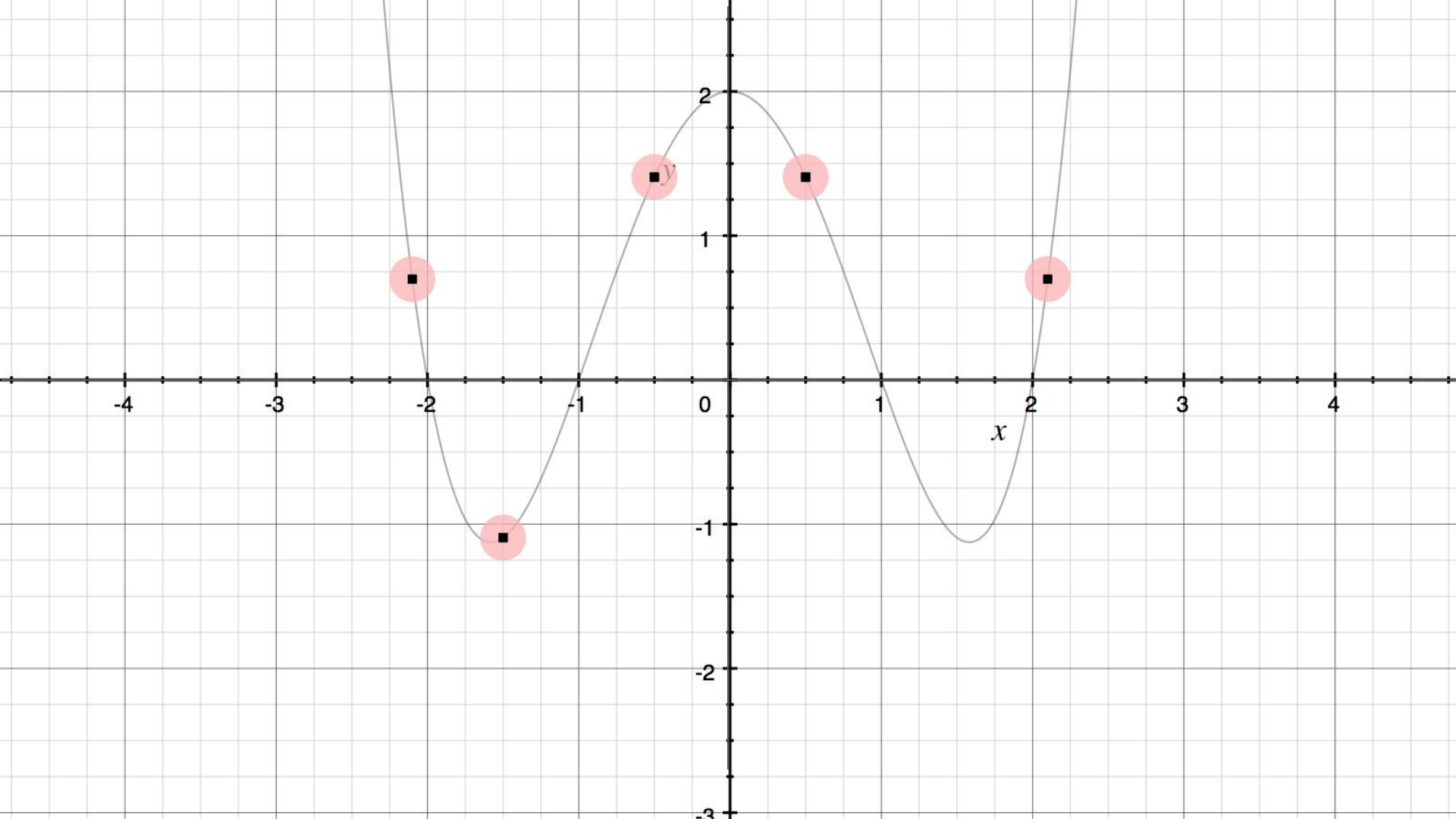

2
y
1
-4
-3
-2
-1
0
1
2
3
4
x
-1
-2


2
y
1
-4
-3
-2
-1
0
1
2
3
4
x
-1
-2


3
2
y
1
-4
-3
-2
-1
0
1
2
3
x
-1
-2


3
2
y
1
-4
-3
-2
-1
0
1
2
3
4
x
-1
-2
-2


2
1
-1
-2
-4
-3
-2
-1
0
1
2
3
4
y
x


y
x
-4
-3
-2
-1
0
1
2
3
4
2
1
-1
-2

How do you do it?

How do you send evaluations of a message polynomial?

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$$

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ 1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1} & x_{n-1}^n \\ 1 & x_n & x_n^2 & \ldots & x_n^{n-1} & x_n^n \end{bmatrix}}_{\mathbf{V}} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix}}_{\vec{a}} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}}_{\vec{b}}$$

How do you do it?

Vandermonde Matrix

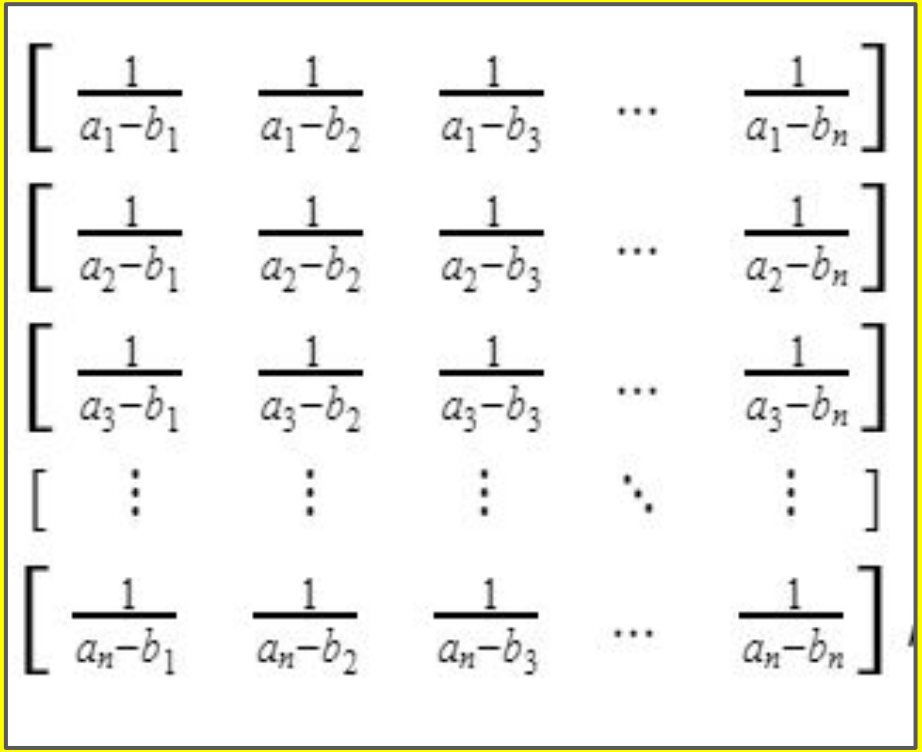
evaluates a polynomial at a set of points

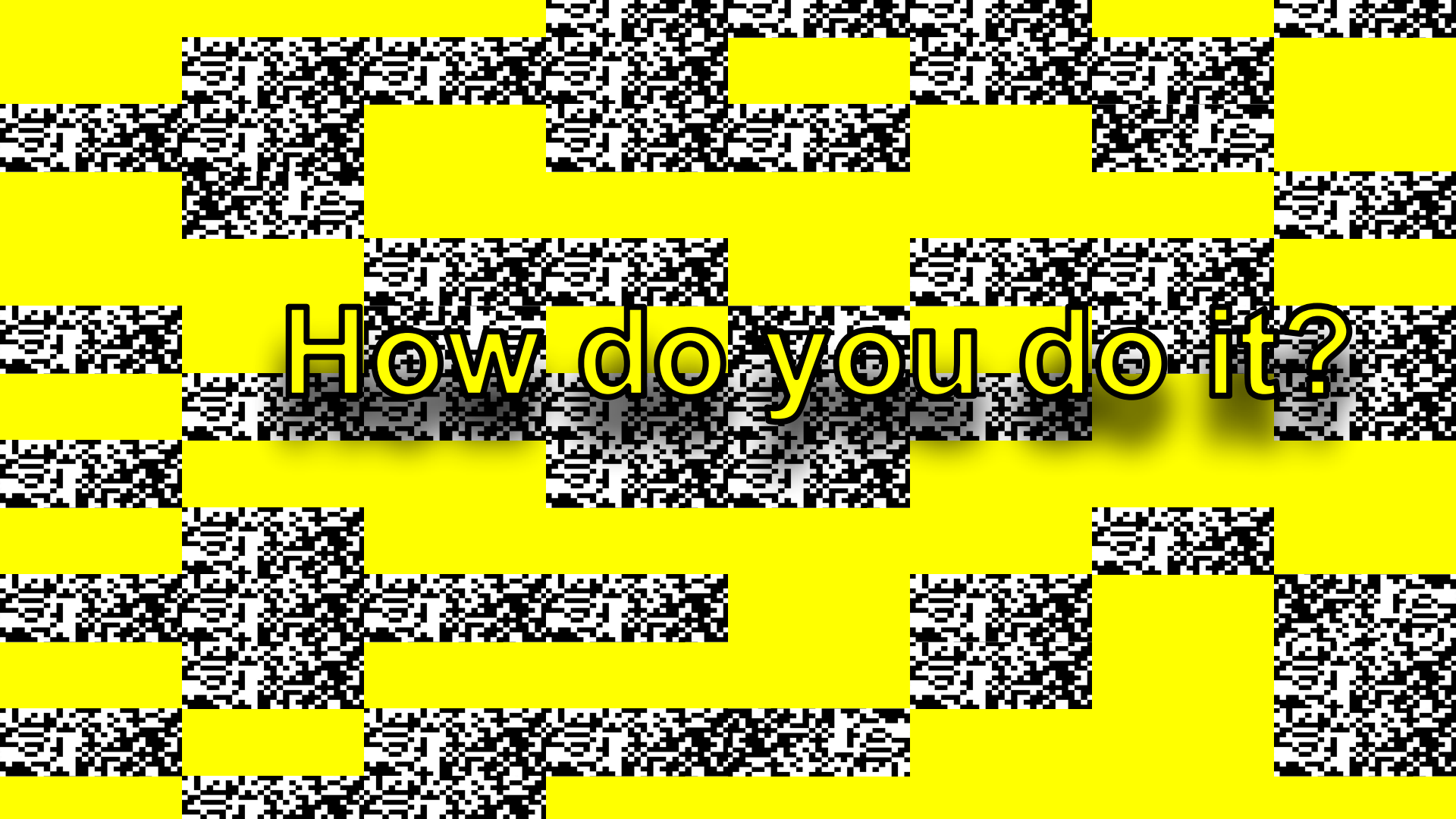
$$\begin{bmatrix} 1 & a_1 & a_1^2 & \dots & a_1^{n-1} \\ 1 & a_2 & a_2^2 & \dots & a_2^{n-1} \\ 1 & a_3 & a_3^2 & \dots & a_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_m & a_m^2 & \dots & a_m^{n-1} \end{bmatrix}$$

How do you do it?

Cauchy Matrix

set of linearly independent equations




How do you do it?


Why do we do this?

Why do this?


Why do we do this?

How do we do this?

Back end API

How do we do it?

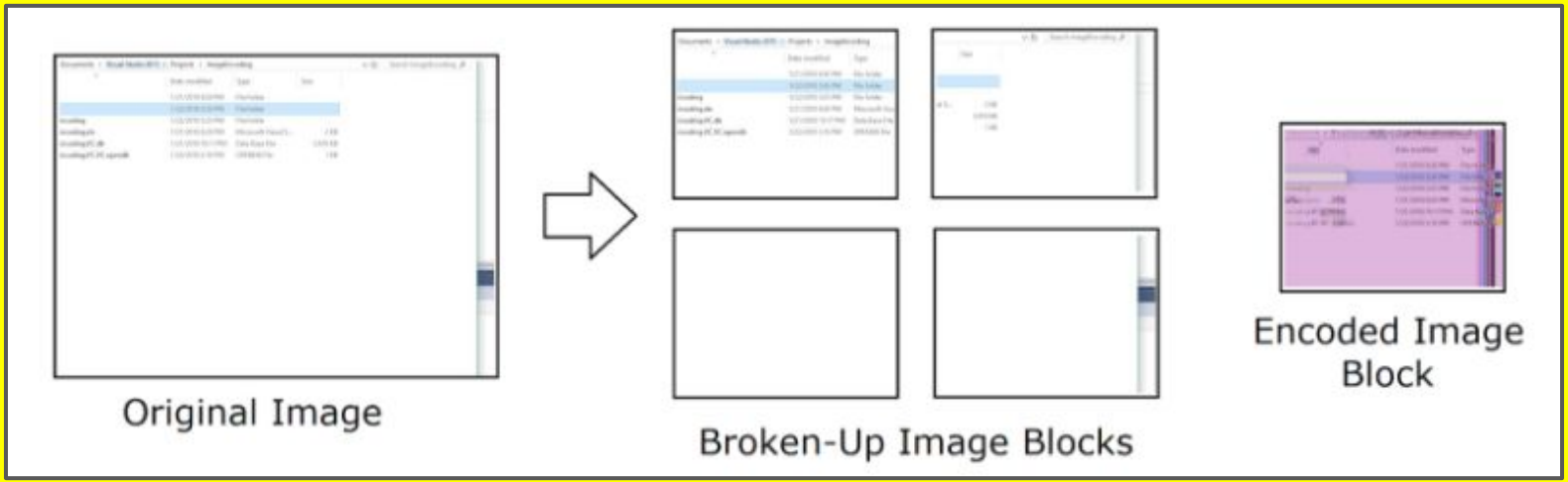
Plan of action for Back-end

Four main features:

- Image encoder using specified parameters
- Step wise algebraic solver
- Image decoder (following steps of algebraic solver)
- Noise and filter insertion

Progress we've made so far

- We have programed an encoder
  - Capable of breaking up images and encoding them using one or more Vandermonde equations
- User will specify number of encoded image blocks to be produced



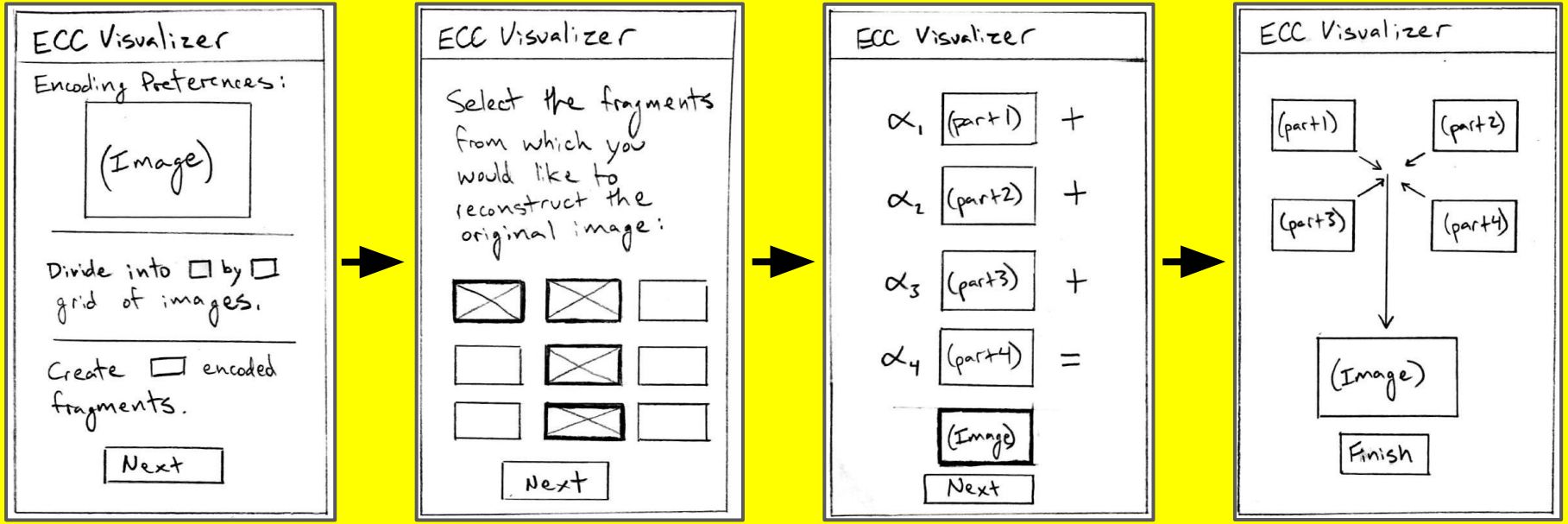
Front End App

Plan of Action for Front-end

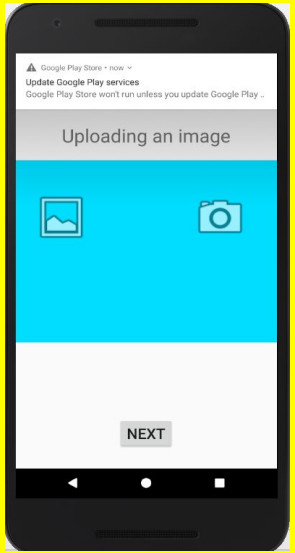
- Four main stages of the app:
  - Image upload
    - Display the image
  - Display the encoding images
    - Selected dimension to split the image and encoding preferences
  - Decoding visualization
    - Select some encoding image for animated reconstruction
  - Reconstructed image displayed

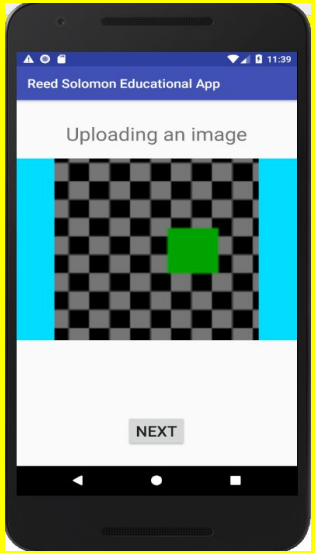
Plan of action

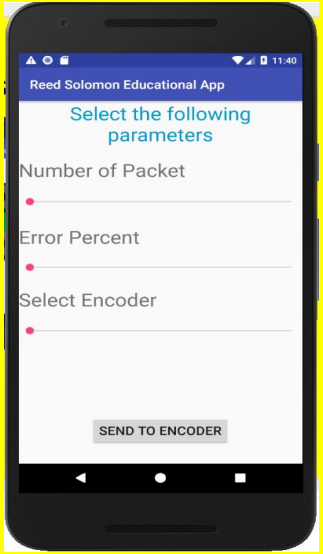
Front end: App Development



Front-end Progress







Currently working on displaying encoding images as function of packets receive from encoder

Timeline

Task	Deadline
Assignment	
Written Design Proposal	3/5/2018
Project Report	3/29/2018
Final Report	5/8/2018
API Development	
Encoder Method I (Vandermonde)	3/15/2018
Encoder Method II (Cauchy)	3/15/2018
Decoder Method I (Vandermonde)	3/23/2018
Decoder Method II (Cauchy)	3/23/2018
Java Wrapper	4/6/2018
Animation	4/9/2018
Application Development	
Application Layout in Android Studio	4/16/2018
Basic Application to Load/Save Image	4/16/2018
API Interface with GUI	4/16/2018
Extra Functionality	
Special-Effects Filter	4/23/2018
Add Noise to Symbols	4/23/2018

Coming Soon to Android....

Reed Solomon Error Correction