# Blockchain at Michigan

## W6: Layer 2 Protocols

2022

# Who am I?

- Nisreen Bahrainwala (@nisreencodes)
- Senior - Data Science and Cognitive Science
- Lurking in Blockchain since 2018
- Forever searching for the perfect pen, please give me recommendations!!
    - Current favorite: Pilot Plumix

# Agenda

1) Scalability Problem
2) L1/L2 Scaling Solutions
3) Side Chains
4) Rollups
    a) How they work
    b) ZK vs Optimistic
5) Zero Knowledge Proofs
6) ZK-SNARK Implementation
7) Why it all matters

# The Scalability Problem

- Transactions per Second
  - Ethereum ~ 13 TPS
  - Centralized payment solution such as VISA ~ 1,700
- Two types of bottlenecks
  - Latency (time spent waiting for a transaction to be processed)
    - Mining difficulty (takes longer to get consensus)
  - Throughput (number of transactions in a given time period)
    - Practicality (hard to work with terabytes of data)
    - Node processing limits (potential to fall out of sync)
    - Block gas limit

# Potential Scaling Solutions

- Key Objectives
  - Maintain the core features of the blockchain solution
- Methods
  - Make the blockchain process more transactions
  - Change the way the blockchain is used
- Example Solutions
  - L1 blockchains (Solana)
  - Sidechains
  - Rollups
  - State channels
  - Ethereum Plasma

# Sidechains vs Rollups

Sidechains

- Term has evolved over time, generally means an independent blockchain protocol that can communicate with another blockchain
- Bridge contract **DOES NOT** validate transactions
  - Can receive information about each other
  - Validation is carried out by parties at either end of the contract
- Examples:
  - Polygon

Rollups

- Computes transactions “off-line” from the mainnet
- Bridge contract **DOES** validate transactions
  - Each party provides information
  - Bridge contract is responsible for verifying this information
- Examples:
  - Arbitrum
  - Optimism
  - zkSync

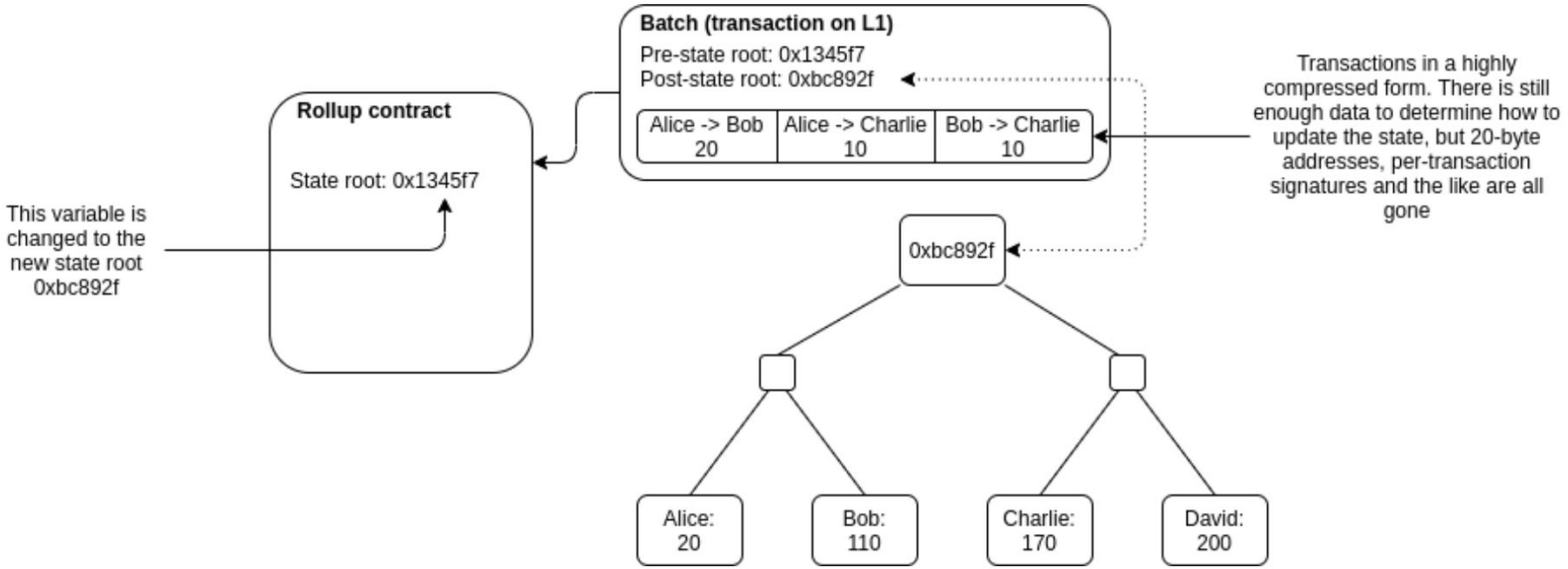
# Rollups: ZK and Optimistic

- Main goal
  - Process transactions outside of the L1 blockchain by moving computation and state storage off chain, keep minimal data on chain
- Two main types
  - Optimistic rollups
    - Assumes transactions are valid by default
    - Runs fraud proof when there is a challenge
  - Zero-knowledge rollups
    - Submits validity proof to chain (SNARK or STARK)

# Anatomy of a Rollup

- Main chain has a state root, this is the merkle root of the rollup
- Transactors (users of the system) can make transactions on this protocol
- Relayers (or aggregators) will compress these transactions, forming a Merkle Tree
- The compressed transaction’s previous Merkle Tree root is compared to the state root
- If there is a match, the state root is updated to the transaction’s root

# Anatomy of a Rollup



# Tradeoffs Between Proof Methods

Optimistic

- Lightweight
- Withdrawal period ~ 1 week (need to wait for someone to publish a fraud proof/challenge)
- Not very complex/easier to generalize
- Lower off chain computation costs

ZK

- Higher gas cost per batch
- Very fast
- Highly complex
- Not as easily generalizable
- Higher on chain computation costs

# Zero Knowledge Proofs

# The Basics: What is a Zero Knowledge Proof?

- A has a secret. B knows the same secret
- A needs to prove to B that they know the secret without revealing what the secret is
- Pen Example (I need a volunteer)

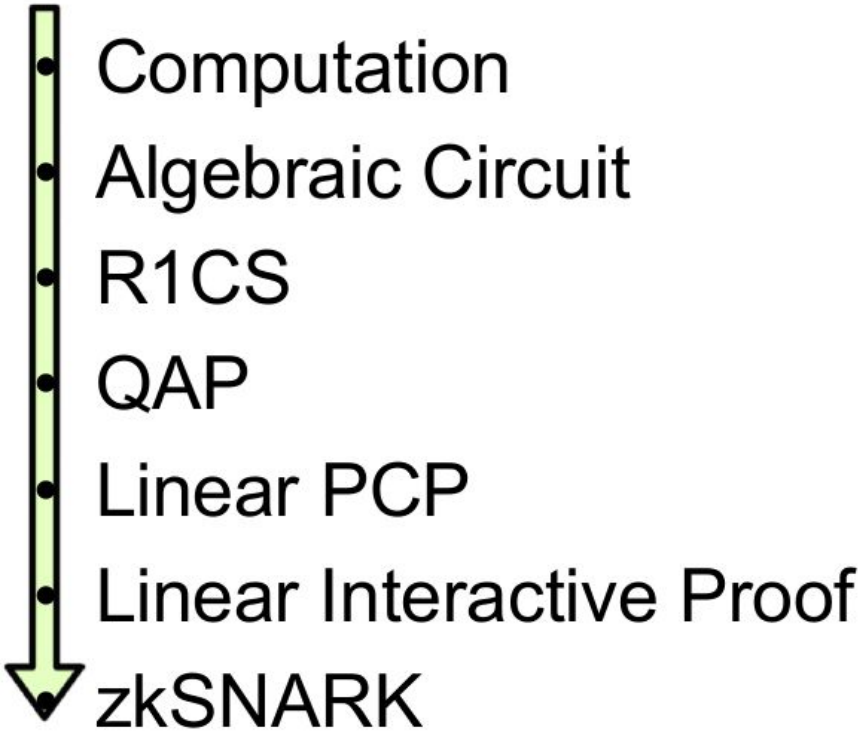
# Attributes

- 3 Main Characteristics
  - Correctness
    - Both people have to know the secret
  - Soundness
    - If one person doesn’t know the secret, then they should not be able to prove the statement and should not be able to prove that I know the statement
  - Zero-Knowledge
    - Nothing other than the statement you want to prove should be revealed during the course of the proof

# What is a ZK-SNARK?

- Zero-Knowledge Succinct Non-Interactive Argument of Knowledge
  - Succinct - size of message is small compared to computation
  - Non-interactive - no/little interaction (single message from prover to verifier after setup)
  - ARguments - only protected against computationally limited provers
  - Knowledge - not possible to construct argument without knowing a certain witness (s vector)
- Use cases:
  - Zcash
- Introduced in 1980s, many interactions/variations since then
- Another important proof is ZK-STARK
  - Zero-Knowledge Scalable Transparent ARguments of Knowledge

# The Math Process



# No more hand waving: Quadratics for the win

Without any reduction, this problem is NP

For any NP problem, there is a reduction function that makes it computable in polynomial time

Convert problem into the “right form” - Quadratic Arithmetic Program (QAP)

Example : x**3 + x + 5 == 35

Cubic equation, we know the answer is 3

# 1) Flattening

- Want to convert the original code into a sequence of statements with two forms
  - x = y
  - x = y (op) z
- Express x**3 + x + 5 == 35 in code

```
sym_1 = x * x
y = sym_1 * x
sym_2 = y + x
~out = sym_2 + 5
```

# 2) Gates to R1CS

- Rank-1 constraint system
  - Sequence of groups of three vectors (Linear Combination)
  - The solution to vectors (a, b, c) is a vector s
    - `s . a * s . b - s . c = 0`
  - “Zip together numbers”
- Vector length = num variables in the system (including dummy var ~one and ~out)
- Why?
  - “Given an assignment describing the full state of a computation, which lists the value of every variable used in the execution, the R1CS can be used to check that all the steps were correctly computed, until one last step(s) confirm(s) the output is as expected.”

# 2) Gates to R1CS cont.

- Variable mapping
  - `'~one', 'x', '~out', 'sym_1', 'y', 'sym_2'`
- Example of a gate
  - a,b,c for sym_1 * x = y

```
a = [0, 0, 0, 1, 0, 0]

b = [0, 1, 0, 0, 0, 0]

c = [0, 0, 0, 0, 1, 0]
```

# Complete R1CS

- Solution Vector (witness)
  - Assignment to all the variables, including input, output and internal variables
  - Start with x = 3
  - `[1, 3, 35, 9, 27, 30]`

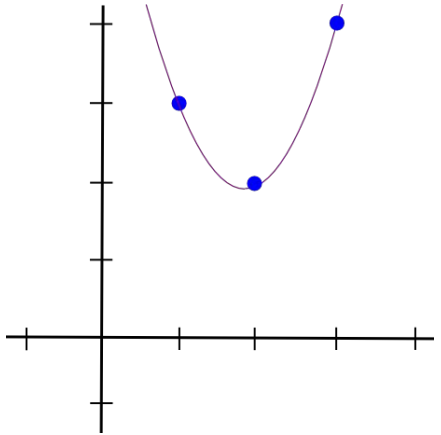
```
A
[0, 1, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 1, 0]
[5, 0, 0, 0, 0, 1]

B
[0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0]

C
[0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0]
```

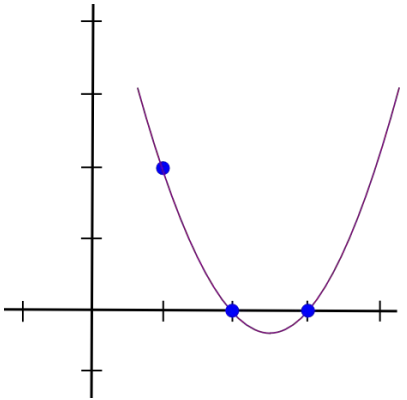
# 3) R1CS to QAP

- Convert from groups of three vectors to six groups of three degree polynomials
- Evaluating at each x coordinate represents one of the constraints
  - x=1, x=2 etc.
- Lagrange Interpolation
  - If you have a set of points (x,y), do lagrange interpolation, you get a polynomial that pases through all points

# Example

- Example:
  - (1,3), (2,2), (3,4)
  - Start by making a polynomial that passes through (1,3), (2,0), (3,0)
    - (x-2) * (x-3)
  - Scale so height at 1 is correct
    - (x-2) * (x-3) * 3 / ((1-2) * (1-3))
    - 1.5 * x**2 - 7.5 * x + 9
  - Repeat this process for the other points)
  - Add all polynomials together
  - 1.5 * x**2 - 5.5 * x + 7





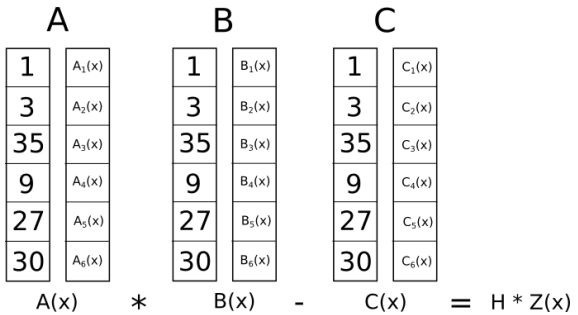
# 4) Checking the QAP

What was the point of all of that? -> we can now use dot products, check all constraints at once

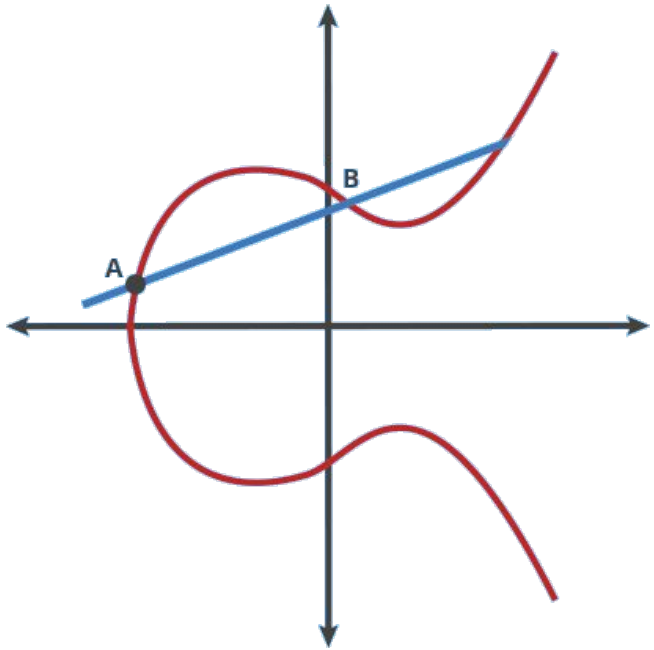
How does this work?

We don't check the polynomial at every point, rather divide by another polynomial (z) and make sure there is no remainder (polynomial equivalence)

If any of the variables in the R1CS solution is falsified, a polynomial will fail the check above



# Elliptic Curve Pairings




Billiards Ball Example:

Hit a ball from point A to B, it then goes from point B to C, and then back to A.

A person would not be able to tell how many times this happened

# Putting it all Together

- Key Cryptographic Assumption (knowledge of exponent):
  - Pair of points P and Q
  - P ** k = Q
  - You get point C. It is not possible to come up with C ** k unless you know the connection between C and P
- In practice
  - Points (P, Q) fall out of the sky
  - P ** k = Q, no one knows k
  - You know (R, S) where R ** k = S
    - Under the assumption above, you had to have taken P and Q, and multiplied by a factor of z to get R ** k = S
    - $e(R, Q) = e(P, S)$

# Putting it All Together cont.

- We have more points fall from the sky (P_1, Q_1), (P_2, Q_2)… (P_10, Q_10)
- You are provided points (R, S) where R * k = S
- Conclusion? There must be a linear combination of P_1 * i_1…, where you know the coefficients i_1..
- Concept of a trusted setup: whoever initialized k must destroy it
    - Why? Because if you know k, there is no need for a linear combination (you can do whatever)

# Back to the QAP

- Reminder: We had a set of polynomials (A, B, C)
- The solution to this was in the form: $A(x) * B(x) - C(x) = H(x) * Z(x)$
- In practice A,B,C are extremely large
  - Prover does not provide the linear combinations directly, rather we prove that it is a linear combination without revealing anything
- t = secret point to evaluate polynomial
- G = “generator” (elliptic curve point)
- T,k_a,k_b,k_c (toxic waste)

- G * A_1(t), G * A_1(t) * k_a
- G * A_2(t), G * A_2(t) * k_a
- …
- G * B_1(t), G * B_1(t) * k_b
- G * B_2(t), G * B_2(t) * k_b
- …
- G * C_1(t), G * C_1(t) * k_c
- G * C_2(t), G * C_2(t) * k_c
- …

- $\pi\_a = G * A(t), \pi'\_a = G * A(t) * k\_a$
- $\pi\_b = G * B(t), \pi'\_b = G * B(t) * k\_b$
- $\pi\_c = G * C(t), \pi'\_c = G * C(t) * k\_c$

# Prover Pt.2

- Just proved that QAP is a linear combination
- Now need to prove that they all have the same coefficients
  - Add another set of values to the trusted setup
  - $G * (A\_i(t) + B\_i(t) + C\_i(t)) * b$
  - b is also toxic waste
- Prover now creates a linear combination with these given values and coefficients, use the pairing trick from before to make sure it matches up with the provided A + B + C

# Prover Part 3

- We have proven that it is a linear combination with the same coefficients
- Now: prove $A * B - C = H * Z$
- Pairing Check!
  - $e(\pi\_a, \pi\_b) / e(\pi\_c, G) ?= e(\pi\_h, G * Z(t))$
  - $\pi\_h = G * H(t)$
- One more thing to consider
  - H is a polynomial, hard to predict its coefficients ahead of time
  - Zcash has generated this sequence up to 2 million, you will always be able to compute H(t)
    - $G, G * t, G * t^2, G * t^3, G * t^4$ ....

# Verification Algorithm

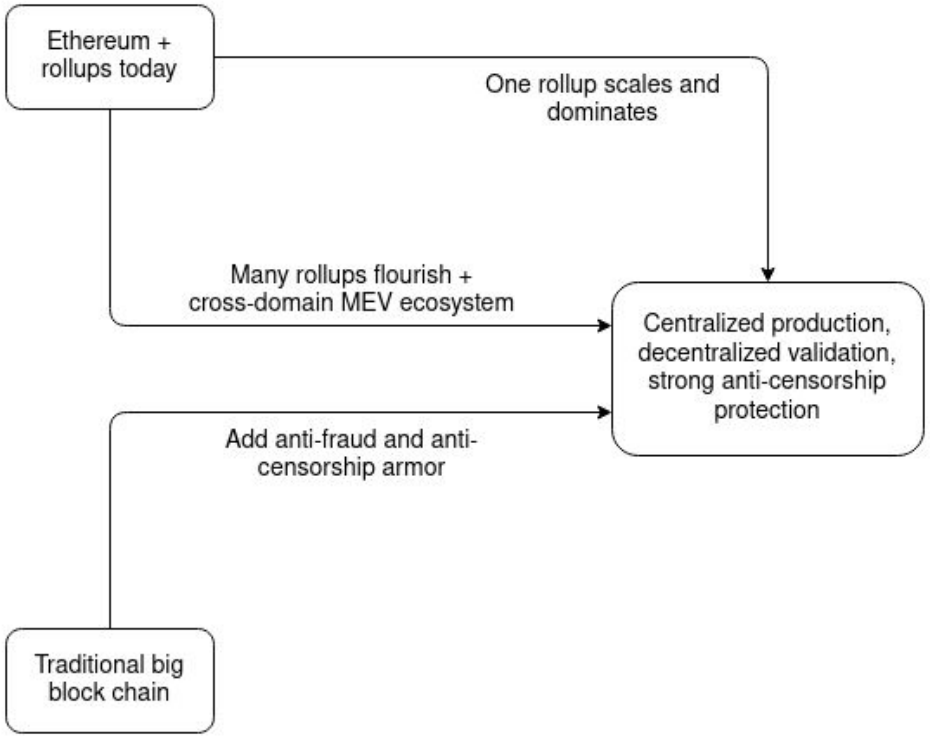
**(c) Verifier $V$**

- INPUTS: verification key vk, input $\vec{x} \in \mathbb{F}_r^n$, and proof $\pi$
- OUTPUTS: decision bit

1. Compute $\mathsf{vk}_{\vec{x}} := \mathsf{vk}_{\mathsf{IC},0} + \sum_{i=1}^{n} x_i \mathsf{vk}_{\mathsf{IC},i} \in \mathbb{G}_1$.
2. Check validity of knowledge commitments for $A, B, C$:
$$e(\pi_{\mathsf{A}}, \mathsf{vk}_{\mathsf{A}}) = e(\pi'_{\mathsf{A}}, \mathcal{P}_2)\,, e(\mathsf{vk}_{\mathsf{B}}, \pi_{\mathsf{B}}) = e(\pi'_{\mathsf{B}}, \mathcal{P}_2)\,,$$
$$e(\pi_{\mathsf{C}}, \mathsf{vk}_{\mathsf{C}}) = e(\pi'_{\mathsf{C}}, \mathcal{P}_2)\,.$$
3. Check same coefficients were used:
$$e(\pi_{\mathsf{K}}, \mathsf{vk}_{\gamma}) = e(\mathsf{vk}_{\vec{x}} + \pi_{\mathsf{A}} + \pi_{\mathsf{C}}, \mathsf{vk}^2_{\beta\gamma}) \cdot e(\mathsf{vk}^1_{\beta\gamma}, \pi_{\mathsf{B}})\ .$$
4. Check QAP divisibility:
$$e(\mathsf{vk}_{\vec{x}} + \pi_{\mathsf{A}}, \pi_{\mathsf{B}}) = e(\pi_{\mathsf{H}}, \mathsf{vk}_{\mathsf{Z}}) \cdot e(\pi_{\mathsf{C}}, \mathcal{P}_2)\ .$$
5. Accept if and only if all the above checks succeeded.

# Questions?

# Endgame


Ethereum + rollups today
One rollup scales and dominates
Many rollups flourish + cross-domain MEV ecosystem
Centralized production, decentralized validation, strong anti-censorship protection
Add anti-fraud and anti-censorship armor
Traditional big block chain

# Thank you!

Please feel free to reach out:

bahrainw@umich.edu

# 5 min break

# Discussion

1. question
2. question
3. question

# Questions?

# Text

More text

# Readings:

1. Reading, by Author
2. Reading, by Author

*There will be a discussion next session