Zero-Knowledge Proofs And ZK-SNARKs (2): Concrete Protocols

Foundations Seminar

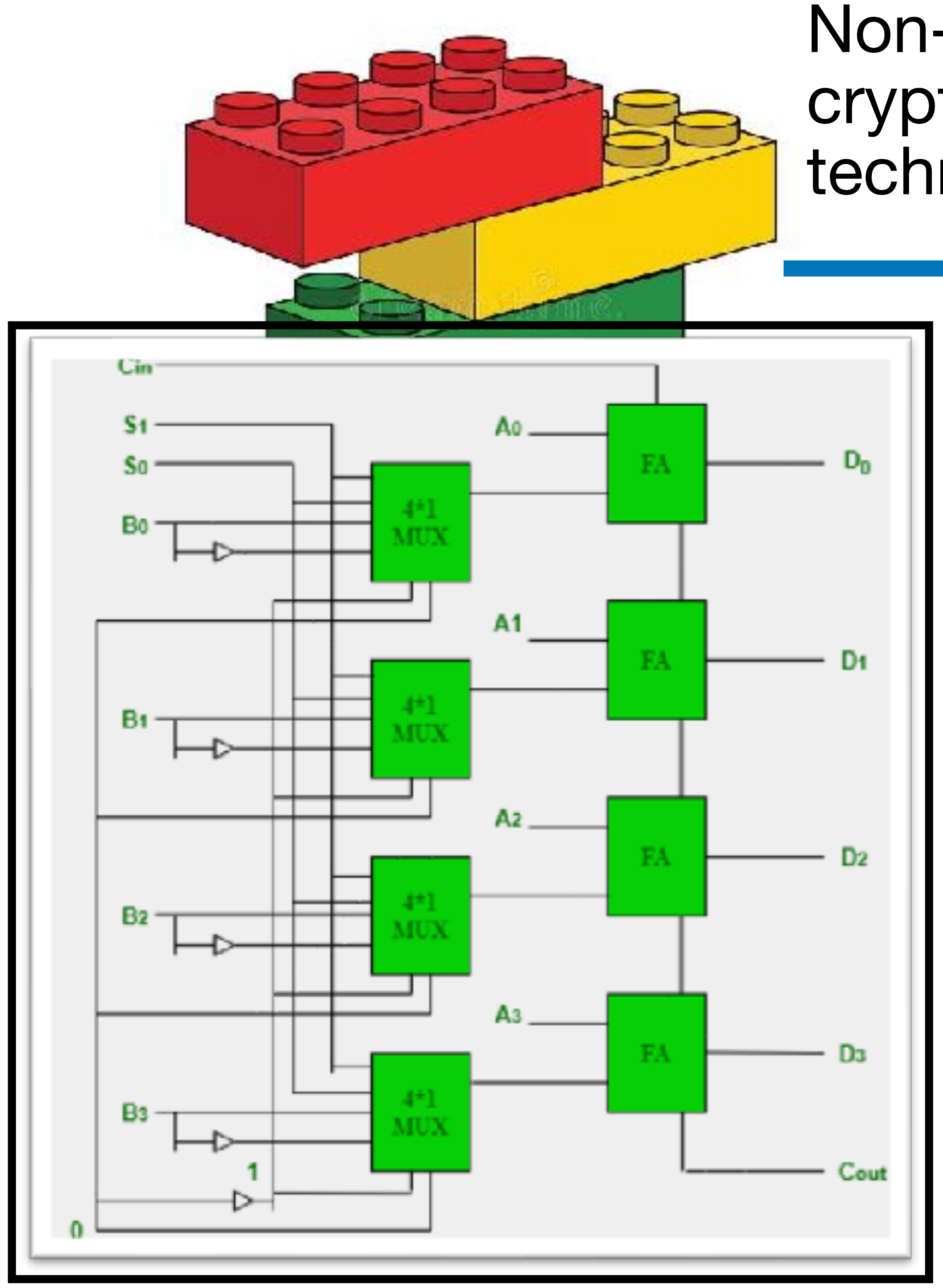
Helger Lipmaa, April 8, 2025

Up To Now

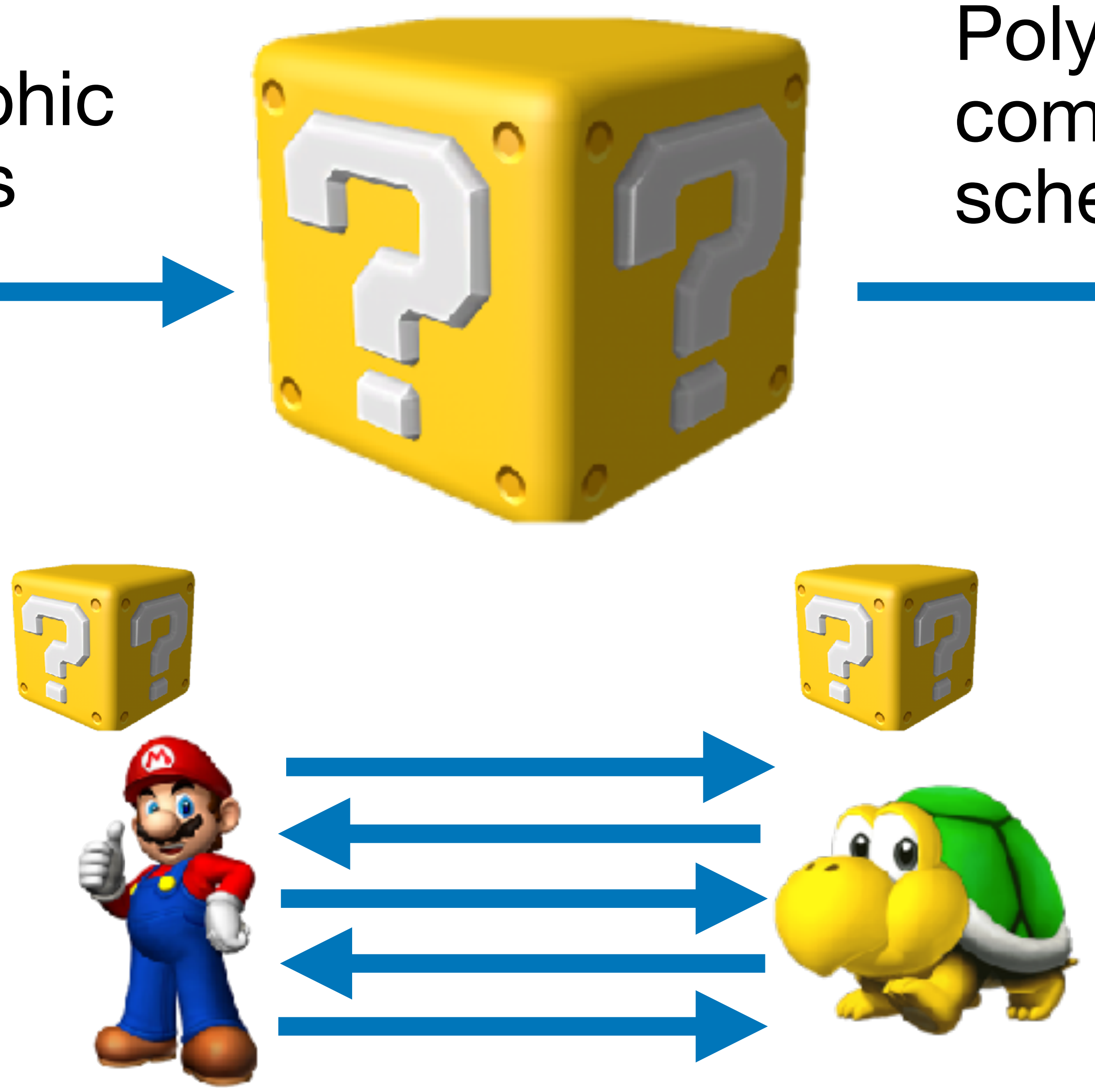
Intermediate Representation

(Polynomial) Interactive Oracle Proof

ZK-SNARKs



Non-cryptographic techniques

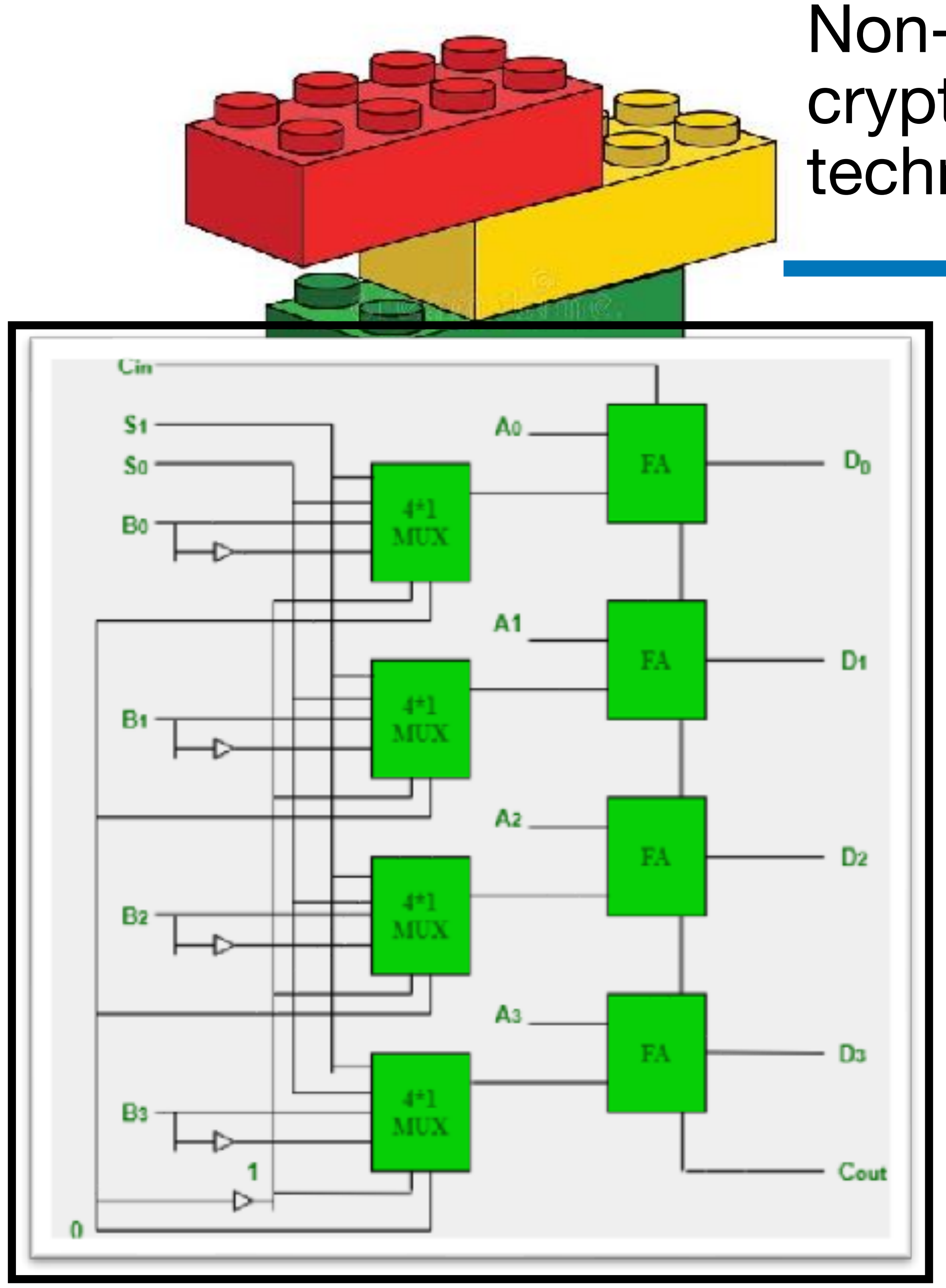


Polynomial commitment scheme

Up To Now

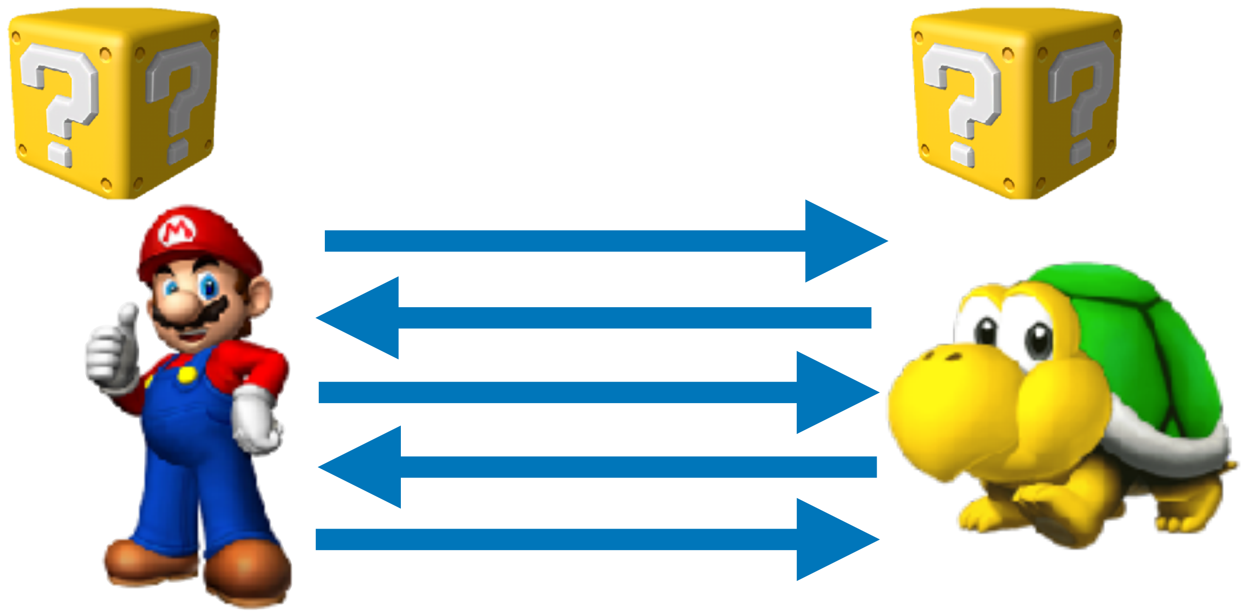
- We explored the current high-level landscape of zk-SNARKs

Intermediate Representation



Non-cryptographic techniques

(Polynomial) Interactive Oracle Proof



Polynomial commitment scheme

ZK-SNARKs

Today's Seminar

- Common IR: arithmetic circuits + **low-degree extensions**

Today's Seminar

- Common IR: arithmetic circuits + **low-degree extensions**
  - **Low-degree extensions = interpolating polynomials**

Today's Seminar

- Common IR: arithmetic circuits + **low-degree extensions**
  - **Low-degree extensions = interpolating polynomials**
  - We will explain interpolation, omit a.c. (not enough time)

Today's Seminar

- Common IR: arithmetic circuits + **low-degree extensions**
  - **Low-degree extensions = interpolating polynomials**
  - We will explain interpolation, omit a.c. (not enough time)
- Simplest possible PIOP: Zero Check

Today's Seminar

- Common IR: arithmetic circuits + **low-degree extensions**
  - **Low-degree extensions = interpolating polynomials**
  - We will explain interpolation, omit a.c. (not enough time)
- Simplest possible PIOP: Zero Check
- More complicated PIOP: Product Check

Today's Seminar

- Common IR: arithmetic circuits + **low-degree extensions**
  - **Low-degree extensions = interpolating polynomials**
  - We will explain interpolation, omit a.c. (not enough time)
- Simplest possible PIOP: Zero Check
- More complicated PIOP: Product Check
- Efficiency of product check

Mathematical Setting

- $\mathbb{F}$ is a **finite field** of **prime** order $|\mathbb{F}| \leq 2^{256}$

Mathematical Setting

- $\mathbb{F}$ is a **finite field** of **prime** order $|\mathbb{F}| \leq 2^{256}$
  - Prime order: $\mathbb{F} = \mathbb{Z}_p = \{0,\ldots,p-1\}$ with modular arithmetic

Mathematical Setting

- $\mathbb{F}$ is a **finite field** of **prime** order $|\mathbb{F}| \leq 2^{256}$
  - Prime order: $\mathbb{F} = \mathbb{Z}_p = \{0,\ldots,p-1\}$ with modular arithmetic
  - Using smaller finite fields is possible

Mathematical Setting

- $\mathbb{F}$ is a **finite field** of **prime** order $|\mathbb{F}| \leq 2^{256}$
  - Prime order: $\mathbb{F} = \mathbb{Z}_p = \{0,\ldots,p-1\}$ with modular arithmetic
  - Using smaller finite fields is possible
  - Given setting is easiest to explain, and **needed** when using elliptic curves

Mathematical Setting

- $\mathbb{F}$ is a **finite field** of **prime** order $|\mathbb{F}| \leq 2^{256}$
  - Prime order: $\mathbb{F} = \mathbb{Z}_p = \{0,\ldots,p-1\}$ with modular arithmetic
  - Using smaller finite fields is possible
  - Given setting is easiest to explain, and **needed** when using elliptic curves
    - Small field **can** cause problems: special-soundness, knowledge error

Mathematical Setting

- $\mathbb{F}$ is a **finite field** of **prime** order $|\mathbb{F}| \leq 2^{256}$
  - Prime order: $\mathbb{F} = \mathbb{Z}_p = \{0,\ldots,p-1\}$ with modular arithmetic
  - Using smaller finite fields is possible
  - Given setting is easiest to explain, and **needed** when using elliptic curves
    - Small field **can** cause problems: special-soundness, knowledge error
- Currently, we use **univariate** polynomials

Mathematical Setting

- $\mathbb{F}$ is a **finite field** of **prime** order $|\mathbb{F}| \leq 2^{256}$
  - Prime order: $\mathbb{F} = \mathbb{Z}_p = \{0,\ldots,p-1\}$ with modular arithmetic
  - Using smaller finite fields is possible
  - Given setting is easiest to explain, and **needed** when using elliptic curves
    - Small field **can** cause problems: special-soundness, knowledge error
- Currently, we use **univariate** polynomials
  - Alternative: multilinear polynomials // not this time

Mathematical Setting

- $\mathbb{F}$ is a **finite field** of **prime** order $|\mathbb{F}| \leq 2^{256}$
  - Prime order: $\mathbb{F} = \mathbb{Z}_p = \{0,\ldots,p-1\}$ with modular arithmetic
  - Using smaller finite fields is possible
  - Given setting is easiest to explain, and **needed** when using elliptic curves
    - Small field **can** cause problems: special-soundness, knowledge error
- Currently, we use **univariate** polynomials
  - Alternative: multilinear polynomials // not this time
- **Notation** $\mathbb{F}_{\leq n}[X]$, $\mathbb{F}_{\leq n}[X]$: univariate polynomials over $\mathbb{F}$ of degree $\leq n$, $< n$

Mathematical Setting

- $\mathbb{F}$ is a **finite field** of **prime** order $|\mathbb{F}| \leq 2^{256}$
  - Prime order: $\mathbb{F} = \mathbb{Z}_p = \{0, \ldots, p-1\}$ with modular arithmetic
  - Using smaller finite fields is possible
  - Given setting is easiest to explain, and **needed** when using elliptic curves
    - Small field **can** cause problems: special-soundness, knowledge error
- Currently, we use **univariate** polynomials
  - Alternative: multilinear polynomials // not this time
- **Notation** $\mathbb{F}_{\leq n}[X]$, $\mathbb{F}_{\leq n}[X]$: univariate polynomials over $\mathbb{F}$ of degree $\leq n$, $< n$
- Input size $n \geq 2^{24}$, companies are pushing for $n \geq 2^{28}$

Reminder: FFT (NTT)

- **FFT** = multipoint evaluation: $f(X) = \sum_{i=0}^{n-1} f_i(X) \mapsto (f(\omega_0), \ldots, f(\omega_{n-1}))$

Reminder: FFT (NTT)

- **FFT** = multipoint evaluation: $f(X) = \sum_{i=0}^{n-1} f_i(X) \mapsto (f(\omega_0), \ldots, f(\omega_{n-1}))$
- **Inverse FFT** = interpolation: $(f(\omega_0), \ldots, f(\omega_{n-1})) \mapsto f(X) = \sum_{i=0}^{n-1} f_i(X)$

Reminder: FFT (NTT)

- **FFT** = multipoint evaluation: $f(X) = \sum_{i=0}^{n-1} f_i(X) \mapsto (f(\omega_0), \ldots, f(\omega_{n-1}))$
- **Inverse FFT** = interpolation: $(f(\omega_0), \ldots, f(\omega_{n-1})) \mapsto f(X) = \sum_{i=0}^{n-1} f_i(X)$
    - $f(X) \leftarrow \sum_{i=0}^{n} f(\omega^i)\ell_i(X)$ // $\ell_i(X)$ are Lagrange polynomials

Reminder: FFT (NTT)

- **FFT** = multipoint evaluation: $f(X) = \sum_{i=0}^{n-1} f_i(X) \mapsto (f(\omega_0), \ldots, f(\omega_{n-1}))$
- **Inverse FFT** = interpolation: $(f(\omega_0), \ldots, f(\omega_{n-1})) \mapsto f(X) = \sum_{i=0}^{n-1} f_i(X)$
  - $f(X) \leftarrow \sum_{i=0}^{n} f(\omega^i)\ell_i(X)$ // $\ell_i(X)$ are Lagrange polynomials
- $\mathbb{F}$ is “FFT-friendly”: $2^{32} \mid (|\mathbb{F}| - 1)$

Reminder: FFT (NTT)

- **FFT** = multipoint evaluation: $f(X) = \sum_{i=0}^{n-1} f_i(X) \mapsto (f(\omega_0), \ldots, f(\omega_{n-1}))$
- **Inverse FFT** = interpolation: $(f(\omega_0), \ldots, f(\omega_{n-1})) \mapsto f(X) = \sum_{i=0}^{n-1} f_i(X)$
  - $f(X) \leftarrow \sum_{i=0}^{n} f(\omega^i)\ell_i(X)$ // $\ell_i(X)$ are Lagrange polynomials
- $\mathbb{F}$ is "FFT-friendly": $2^{32} \mid (|\mathbb{F}| - 1)$
  - Exists $\mathbb{H} = \langle \omega \rangle = \{\omega^i : i \in [0, n-1]\}$: mult. subgroup of $\mathbb{F}^*$ of order n

Reminder: FFT (NTT)

- **FFT** = multipoint evaluation: $f(X) = \sum_{i=0}^{n-1} f_i(X) \mapsto (f(\omega_0), \ldots, f(\omega_{n-1}))$
- **Inverse FFT** = interpolation: $(f(\omega_0), \ldots, f(\omega_{n-1})) \mapsto f(X) = \sum_{i=0}^{n-1} f_i(X)$
  - $f(X) \leftarrow \sum_{i=0}^{n} f(\omega^i)\ell_i(X)$ // $\ell_i(X)$ are Lagrange polynomials
- $\mathbb{F}$ is "FFT-friendly": $2^{32} \mid (|\mathbb{F}| - 1)$
  - Exists $\mathbb{H} = \langle \omega \rangle = \{\omega^i : i \in [0, n-1]\}$: mult. subgroup of $\mathbb{F}^*$ of order n
  - FFT $f(X) = \sum_{i=0}^{n-1} f_i(X) \mapsto (f(\omega^0), \ldots, f(\omega^{n-1}))$ in $O(n \log n)$ field ops

Reminder: FFT (NTT)

- **FFT** = multipoint evaluation: $f(X) = \sum_{i=0}^{n-1} f_i(X) \mapsto (f(\omega_0), \ldots, f(\omega_{n-1}))$
- **Inverse FFT** = interpolation: $(f(\omega_0), \ldots, f(\omega_{n-1})) \mapsto f(X) = \sum_{i=0}^{n-1} f_i(X)$
    - $f(X) \leftarrow \sum_{i=0}^{n} f(\omega^i)\ell_i(X)$ // $\ell_i(X)$ are Lagrange polynomials
- $\mathbb{F}$ is "FFT-friendly": $2^{32} \mid (|\mathbb{F}| - 1)$
    - Exists $\mathbb{H} = \langle \omega \rangle = \{\omega^i : i \in [0,n-1]\}$: mult. subgroup of $\mathbb{F}^*$ of order n
    - FFT $f(X) = \sum_{i=0}^{n-1} f_i(X) \mapsto (f(\omega^0), \ldots, f(\omega^{n-1}))$ in $O(n \log n)$ field ops
    - Interpolation $(f(\omega^0), \ldots, f(\omega^{n-1})) \mapsto f(X) = \sum_{i=0}^{n-1} f_i(X)$ in $O(n \log n)$ f.o.

Reminder: FFT (NTT)

- **FFT** = multipoint evaluation: $f(X) = \sum_{i=0}^{n-1} f_i(X) \mapsto (f(\omega_0), \ldots, f(\omega_{n-1}))$
- **Inverse FFT** = interpolation: $(f(\omega_0), \ldots, f(\omega_{n-1})) \mapsto f(X) = \sum_{i=0}^{n-1} f_i(X)$
  - $f(X) \leftarrow \sum_{i=0}^{n} f(\omega^i)\ell_i(X)$ // $\ell_i(X)$ are Lagrange polynomials
- $\mathbb{F}$ is "FFT-friendly": $2^{32} \mid (|\mathbb{F}| - 1)$
  - Exists $\mathbb{H} = \langle\omega\rangle = \{\omega^i : i \in [0,n-1]\}$: mult. subgroup of $\mathbb{F}^*$ of order n
  - FFT $f(X) = \sum_{i=0}^{n-1} f_i(X) \mapsto (f(\omega^0), \ldots, f(\omega^{n-1}))$ in $O(n \log n)$ field ops
  - Interpolation $(f(\omega^0), \ldots, f(\omega^{n-1})) \mapsto f(X) = \sum_{i=0}^{n-1} f_i(X)$ in $O(n \log n)$ f.o.
  - => almost all univariate PIOP based SNARKs use such fields

Reminder: Polynomial IOP

Reminder: Polynomial IOP

Zero Check

First PIOP: Zero Check

- **Witness:** vector $a \in \mathbb{F}^n$

First PIOP: Zero Check

- **Witness:** vector $\boldsymbol{a} \in \mathbb{F}^n$
  - Any vector somewhere in the middle of calculations…

First PIOP: Zero Check

- **Witness:** vector $\boldsymbol{a} \in \mathbb{F}^n$
    - Any vector somewhere in the middle of calculations…
    - Vector of wire values of a circuit

First PIOP: Zero Check

- **Witness:** vector $\boldsymbol{a} \in \mathbb{F}^n$
  - Any vector somewhere in the middle of calculations…
  - Vector of wire values of a circuit
- "**Public input":** oracle to $\boldsymbol{a} \in \mathbb{F}^n$

First PIOP: Zero Check

- **Witness:** vector $\boldsymbol{a} \in \mathbb{F}^n$
  - Any vector somewhere in the middle of calculations…
  - Vector of wire values of a circuit
- "**Public input":** oracle to $\boldsymbol{a} \in \mathbb{F}^n$
- **Goal:** The prover aims to convice the verifier $\boldsymbol{a} = \mathbf{0}_n$ is **zero vector**

First PIOP: Zero Check

- **Witness:** vector $\boldsymbol{a} \in \mathbb{F}^n$
  - Any vector somewhere in the middle of calculations…
  - Vector of wire values of a circuit
- "**Public input**": oracle to $\boldsymbol{a} \in \mathbb{F}^n$
- **Goal:** The prover aims to convice the verifier $\boldsymbol{a} = \boldsymbol{0}_n$ is **zero vector**
- Formally: prove $(\mathrm{x}, \mathrm{w}) \in \mathscr{R}_0$ for $\mathscr{R}_0 := \{(\blacksquare, \boldsymbol{a}) : \mathsf{Enc}(\boldsymbol{a}) \in \blacksquare \wedge \boldsymbol{a} = \boldsymbol{0}_n\}$

First PIOP: Zero Check

- **Witness:** vector $\boldsymbol{a} \in \mathbb{F}^n$
  - Any vector somewhere in the middle of calculations…
  - Vector of wire values of a circuit
- "**Public input":** oracle to $\boldsymbol{a} \in \mathbb{F}^n$
- **Goal:** The prover aims to convice the verifier $\boldsymbol{a} = \mathbf{0}_n$ is **zero vector**
- Formally: prove $(\mathsf{x}, \mathsf{w}) \in \mathscr{R}_0$ for $\mathscr{R}_0 := \{(\blacksquare, \boldsymbol{a}) : \mathsf{Enc}(\boldsymbol{a}) \in \blacksquare \wedge \boldsymbol{a} = \mathbf{0}_n\}$
  - However, we have PIOP, so oracle contains a polynomial

First PIOP: Zero Check

- **Witness:** vector $\boldsymbol{a} \in \mathbb{F}^n$
  - Any vector somewhere in the middle of calculations…
  - Vector of wire values of a circuit
- "**Public input":** oracle to $\boldsymbol{a} \in \mathbb{F}^n$
- **Goal:** The prover aims to convice the verifier $\boldsymbol{a} = \mathbf{0}_n$ is **zero vector**
- Formally: prove $(\mathtt{x}, \mathtt{w}) \in \mathscr{R}_0$ for $\mathscr{R}_0 := \{(\blacksquare, \boldsymbol{a}) : \mathsf{Enc}(\boldsymbol{a}) \in \blacksquare \wedge \boldsymbol{a} = \mathbf{0}_n\}$
  - However, we have PIOP, so oracle contains a polynomial
  - We will explain that next…

Zero Check

Motivation

- Zero Check is a very basic check

Zero Check
Motivation

- Zero Check is a very basic check
  - Underlies essentially anything else

Zero Check

Motivation

- Zero Check is a very basic check
  - Underlies essentially anything else
- **Example:** $a = b \Leftrightarrow a - b = 0$

Zero Check

Motivation

- Zero Check is a very basic check
  - Underlies essentially anything else
- **Example:** $a = b \Leftrightarrow a - b = 0$
  - $a + b = c \Leftrightarrow a + b - c = 0$

Polynomial View of Zero Check

- **Part of Intermediate Representatn:** interpret $a = 0$ as polynomial constraint

Polynomial View of Zero Check

- **Part of Intermediate Representatn:** interpret $\boldsymbol{a} = \mathbf{0}$ as polynomial constraint
  - Enc: map $\boldsymbol{a} \in \mathbb{F}^n$ to $\hat{a}(X) \in \mathbb{F}_{\leq n-1}[X]$, its **interpolating polynomial**

Polynomial View of Zero Check

- **Part of Intermediate Representatn:** interpret $\boldsymbol{a} = \mathbf{0}$ as polynomial constraint
  - Enc: map $\boldsymbol{a} \in \mathbb{F}^n$ to $\hat{a}(X) \in \mathbb{F}_{\leq n-1}[X]$, its **interpolating polynomial**
    - $\forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1}) = a_i$

Polynomial View of Zero Check

- **Part of Intermediate Representatn:** interpret $\boldsymbol{a} = \mathbf{0}$ as polynomial constraint
  - Enc: map $\boldsymbol{a} \in \mathbb{F}^n$ to $\hat{a}(X) \in \mathbb{F}_{\leq n-1}[X]$, its **interpolating polynomial**
    - $\forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1}) = a_i$
    - Enc and its inverse ($\mathsf{Enc}^{-1} = \mathsf{FFT}$) are bijective, efficiently computable

Polynomial View of Zero Check

- **Part of Intermediate Representatn:** interpret $\boldsymbol{a} = \mathbf{0}$ as polynomial constraint
  - Enc: map $\boldsymbol{a} \in \mathbb{F}^n$ to $\hat{a}(X) \in \mathbb{F}_{\leq n-1}[X]$, its **interpolating polynomial**
    - $\forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1}) = a_i$
    - Enc and its inverse ($\mathsf{Enc}^{-1} = \mathsf{FFT}$) are bijective, efficiently computable
- **Zero check:** $\boldsymbol{a} = \mathbf{0}$ iff $\hat{a}(\omega^{i-1}) = 0$ for all $i \in [1,n]$

Polynomial View of Zero Check

- **Part of Intermediate Representatn:** interpret $\boldsymbol{a} = \mathbf{0}$ as polynomial constraint
  - Enc: map $\boldsymbol{a} \in \mathbb{F}^n$ to $\hat{a}(X) \in \mathbb{F}_{\leq n-1}[X]$, its **interpolating polynomial**
    - $\forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1}) = a_i$
    - Enc and its inverse ($\mathsf{Enc}^{-1} = \mathsf{FFT}$) are bijective, efficiently computable
- **Zero check:** $\boldsymbol{a} = \mathbf{0}$ iff $\hat{a}(\omega^{i-1}) = 0$ for all $i \in [1,n]$
  - If $\hat{a}(X) \in \mathbb{F}_{\leq n-1}[X]$, the latter holds iff $\hat{a}(X) = 0$

Polynomial View of Zero Check

- **Part of Intermediate Representatn:** interpret $\boldsymbol{a} = \boldsymbol{0}$ as polynomial constraint
  - Enc: map $\boldsymbol{a} \in \mathbb{F}^n$ to $\hat{a}(X) \in \mathbb{F}_{\leq n-1}[X]$, its **interpolating polynomial**
    - $\forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1}) = a_i$
    - Enc and its inverse ($\mathsf{Enc}^{-1} = \mathsf{FFT}$) are bijective, efficiently computable
- **Zero check:** $\boldsymbol{a} = \boldsymbol{0}$ iff $\hat{a}(\omega^{i-1}) = 0$ for all $i \in [1,n]$
  - If $\hat{a}(X) \in \mathbb{F}_{\leq n-1}[X]$, the latter holds iff $\hat{a}(X) = 0$
- Zero Check with oracles for $\mathbb{F}_{\leq n-1}[X]$ is really trivial

Polynomial View of Zero Check

- **Part of Intermediate Representatn:** interpret $\boldsymbol{a} = \mathbf{0}$ as polynomial constraint
  - Enc: map $\boldsymbol{a} \in \mathbb{F}^n$ to $\hat{a}(X) \in \mathbb{F}_{\leq n-1}[X]$, its **interpolating polynomial**
    - $\forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1}) = a_i$
    - Enc and its inverse ($\text{Enc}^{-1} = \text{FFT}$) are bijective, efficiently computable
- **Zero check:** $\boldsymbol{a} = \mathbf{0}$ iff $\hat{a}(\omega^{i-1}) = 0$ for all $i \in [1,n]$
  - If $\hat{a}(X) \in \mathbb{F}_{\leq n-1}[X]$, the latter holds iff $\hat{a}(X) = 0$
- Zero Check with oracles for $\mathbb{F}_{\leq n-1}[X]$ is really trivial
  - Assuming the oracle guarantees the polynomial has "low degree" $\leq n - 1$

Making Polynomial Tests Succinct

- Recall: verifier needs to test $\hat{a}(X) = 0$

Making Polynomial Tests Succinct

- Recall: verifier needs to test $\hat{a}(X) = 0$
- How to do it efficiently?

Making Polynomial Tests Succinct

- Recall: verifier needs to test $\hat{a}(X) = 0$
- How to do it efficiently?
- What do we mean by "efficiently"?

Making Polynomial Tests Succinct

- Recall: verifier needs to test $\hat{a}(X) = 0$
- How to do it efficiently?
- What do we mean by "efficiently"?
    - **Short argument:** Prover sends less information than the whole polynomial

Making Polynomial Tests Succinct

- Recall: verifier needs to test $\hat{a}(X) = 0$
- How to do it efficiently?
- What do we mean by "efficiently"?
  - **Short argument:** Prover sends less information than the whole polynomial
  - **Efficient verifier:** V does less work than checking each coefficient is 0

Making Polynomial Tests Succinct

- Recall: verifier needs to test $\hat{a}(X) = 0$
- How to do it efficiently?
- What do we mean by "efficiently"?
  - **Short argument:** Prover sends less information than the whole polynomial
  - **Efficient verifier:** V does less work than checking each coefficient is 0
- We need to come up with some efficient test of the fact $\hat{a}(X) = 0$ 🤔

Making Polynomial Tests Succinct

- Recall: verifier needs to test $\hat{a}(X) = 0$
- How to do it efficiently?
- What do we mean by "efficiently"?
  - **Short argument:** Prover sends less information than the whole polynomial
  - **Efficient verifier:** V does less work than checking each coefficient is 0
- We need to come up with some efficient test of the fact $\hat{a}(X) = 0$ 🤔
- Sending $\hat{a}(X)$ to verifier is not efficient

Making Polynomial Tests Succinct

- Recall: verifier needs to test $\hat{a}(X) = 0$
- How to do it efficiently?
- What do we mean by "efficiently"?
  - **Short argument:** Prover sends less information than the whole polynomial
  - **Efficient verifier:** V does less work than checking each coefficient is 0
- We need to come up with some efficient test of the fact $\hat{a}(X) = 0$ 🤔
- Sending $\hat{a}(X)$ to verifier is not efficient
- **Hint 1:** we can query the values of $\hat{a}(X)$ at any location

Making Polynomial Tests Succinct

- Recall: verifier needs to test $\hat{a}(X) = 0$
- How to do it efficiently?
- What do we mean by "efficiently"?
  - **Short argument:** Prover sends less information than the whole polynomial
  - **Efficient verifier:** V does less work than checking each coefficient is 0
- We need to come up with some efficient test of the fact $\hat{a}(X) = 0$ 🤔
- Sending $\hat{a}(X)$ to verifier is not efficient
- **Hint 1:** we can query the values of $\hat{a}(X)$ at any location
- **Hint 2:** the verifier can toss random coins

Making Polynomial Tests Succinct

- Recall: verifier needs to test $\hat{a}(X) = 0$
- How to do it efficiently?
- What do we mean by "efficiently"?
  - **Short argument:** Prover sends less information than the whole polynomial
  - **Efficient verifier:** V does less work than checking each coefficient is 0
- We need to come up with some efficient test of the fact $\hat{a}(X) = 0$ 🤔
- Sending $\hat{a}(X)$ to verifier is not efficient
- **Hint 1:** we can query the values of $\hat{a}(X)$ at any location
- **Hint 2:** the verifier can toss random coins
- **Idea** 💡**:** test that $\hat{a}(r) = 0$ for random r **sampled by the verifier**

Making Polynomial Tests Succinct

- Recall: verifier needs to test $\hat{a}(X) = 0$
- How to do it efficiently?
- What do we mean by “efficiently”?
  - **Short argument:** Prover sends less information than the whole polynomial
  - **Efficient verifier:** V does less work than checking each coefficient is 0
- We need to come up with some efficient test of the fact $\hat{a}(X) = 0$ 🤔
- Sending $\hat{a}(X)$ to verifier is not efficient
- **Hint 1:** we can query the values of $\hat{a}(X)$ at any location
- **Hint 2:** the verifier can toss random coins
- **Idea** 💡**:** test that $\hat{a}(r) = 0$ for random r **sampled by the verifier**

Why does this idea work?

Schwartz-Zippel Lemma

- **Lemma.** Let $f(X) \in \mathbb{F}_{\leq n}[X]$ be a **non-zero** polynomial of degree $n \geq 0$. Let $r \leftarrow_{\$} \mathbb{F}$ be sampled uniformly at random. The probability that $f(r) = 0$ is at most $n/|\mathbb{F}|$

Schwartz-Zippel Lemma

Low-degree polynomials don't have too many roots

- **Lemma.** Let $f(X) \in \mathbb{F}_{\leq n}[X]$ be a **non-zero** polynomial of degree $n \geq 0$. Let $r \leftarrow_\$ \mathbb{F}$ be sampled uniformly at random. The probability that $f(r) = 0$ is at most $n/|\mathbb{F}|$
- **Proof:** Straightforward since $f(X)$ has at most n roots

Schwartz-Zippel Lemma

Low-degree polynomials don't have too many roots

- **Lemma.** Let $f(X) \in \mathbb{F}_{\leq n}[X]$ be a **non-zero** polynomial of degree $n \geq 0$. Let $r \leftarrow_\$ \mathbb{F}$ be sampled uniformly at random. The probability that $f(r) = 0$ is at most $n/|\mathbb{F}|$
- **Proof:** Straightforward since $f(X)$ has at most n roots
- Schwartz-Zippel lemma generalises this to multivariate poly-s and subsets of $\mathbb{F}$

Schwartz-Zippel Lemma

Low-degree polynomials don't have too many roots

- **Lemma.** Let $f(X) \in \mathbb{F}_{\leq n}[X]$ be a **non-zero** polynomial of degree $n \geq 0$. Let $r \leftarrow_{\$} \mathbb{F}$ be sampled uniformly at random. The probability that $f(r) = 0$ is at most $n/|\mathbb{F}|$
- **Proof:** Straightforward since $f(X)$ has at most n roots
- Schwartz-Zippel lemma generalises this to multivariate poly-s and subsets of $\mathbb{F}$
- **Lemma (Schwartz-Zippel).** Let $f(X) \in \mathbb{F}[X_1, \ldots, X_m]$ be a **non-zero** polynomial of total degree $n \geq 0$. Let S be a finite subset of $\mathbb{F}$. Let $r_1, \ldots, r_m \leftarrow_{\$} S$ be sampled uniformly at random. Then the probability that $f(r_1, \ldots, r_m) = 0$ is at most $n/|S|$.

Schwartz-Zippel Lemma

Low-degree polynomials don't have too many roots

- **Lemma.** Let $f(X) \in \mathbb{F}_{\leq n}[X]$ be a **non-zero** polynomial of degree $n \geq 0$. Let $r \leftarrow_{\$} \mathbb{F}$ be sampled uniformly at random. The probability that $f(r) = 0$ is at most $n/|\mathbb{F}|$
- **Proof:** Straightforward since $f(X)$ has at most n roots
- Schwartz-Zippel lemma generalises this to multivariate poly-s and subsets of $\mathbb{F}$
- **Lemma (Schwartz-Zippel).** Let $f(X) \in \mathbb{F}[X_1, \ldots, X_m]$ be a **non-zero** polynomial of total degree $n \geq 0$. Let S be a finite subset of $\mathbb{F}$. Let $r_1, \ldots, r_m \leftarrow_{\$} S$ be sampled uniformly at random. Then the probability that $f(r_1, \ldots, r_m) = 0$ is at most $n/|S|$.
- See https://en.wikipedia.org/wiki/Schwartz-Zippel_lemma for a proof

Schwartz-Zippel Lemma

Low-degree polynomials don't have too many roots

- **Lemma.** Let $f(X) \in \mathbb{F}_{\leq n}[X]$ be a **non-zero** polynomial of degree $n \geq 0$. Let $r \leftarrow_\$ \mathbb{F}$ be sampled uniformly at random. The probability that $f(r) = 0$ is at most $n/|\mathbb{F}|$
- **Proof:** Straightforward since $f(X)$ has at most n roots
- Schwartz-Zippel lemma generalises this to multivariate poly-s and subsets of $\mathbb{F}$
- **Lemma (Schwartz-Zippel).** Let $f(X) \in \mathbb{F}[X_1, \ldots, X_m]$ be a **non-zero** polynomial of total degree $n \geq 0$. Let S be a finite subset of $\mathbb{F}$. Let $r_1, \ldots, r_m \leftarrow_\$ S$ be sampled uniformly at random. Then the probability that $f(r_1, \ldots, r_m) = 0$ is at most $n/|S|$.
- See https://en.wikipedia.org/wiki/Schwartz-Zippel_lemma for a proof

- Schwartz-Zippel is hugely important in constructing efficient zk-SNARKs
- We mostly just use the first lemma (but still call it Schwartz-Zippel)

On Schwartz-Zippel

- **Degree mantra:** if $f(X) \neq 0$ then $f(r) \neq 0$ with "high" probability

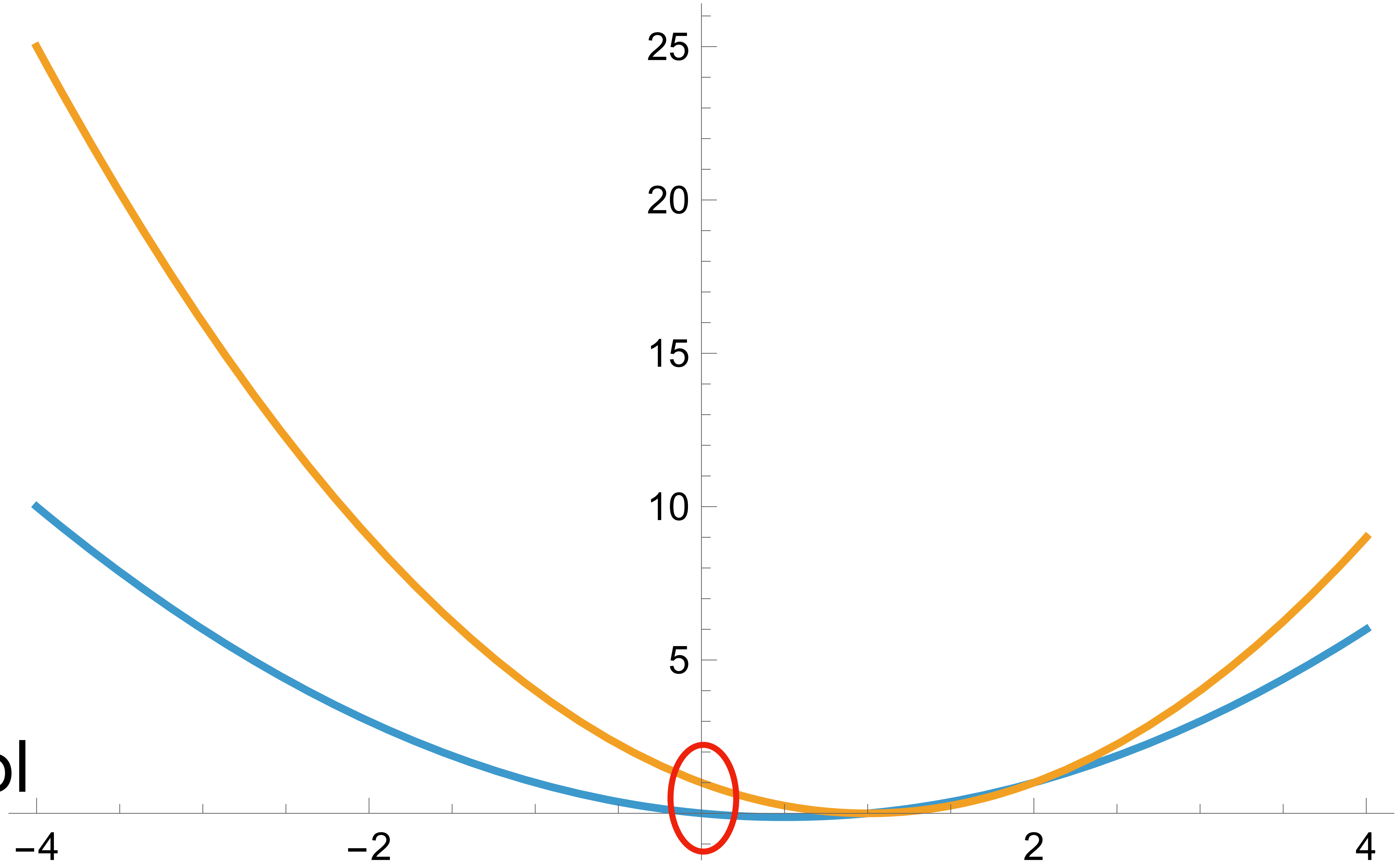
On Schwartz-Zippel

- **Degree mantra:** if $f(X) \neq 0$ then $f(r) \neq 0$ with “high” probability
- Schwartz-Zippel is extremely useful tool

On Schwartz-Zippel

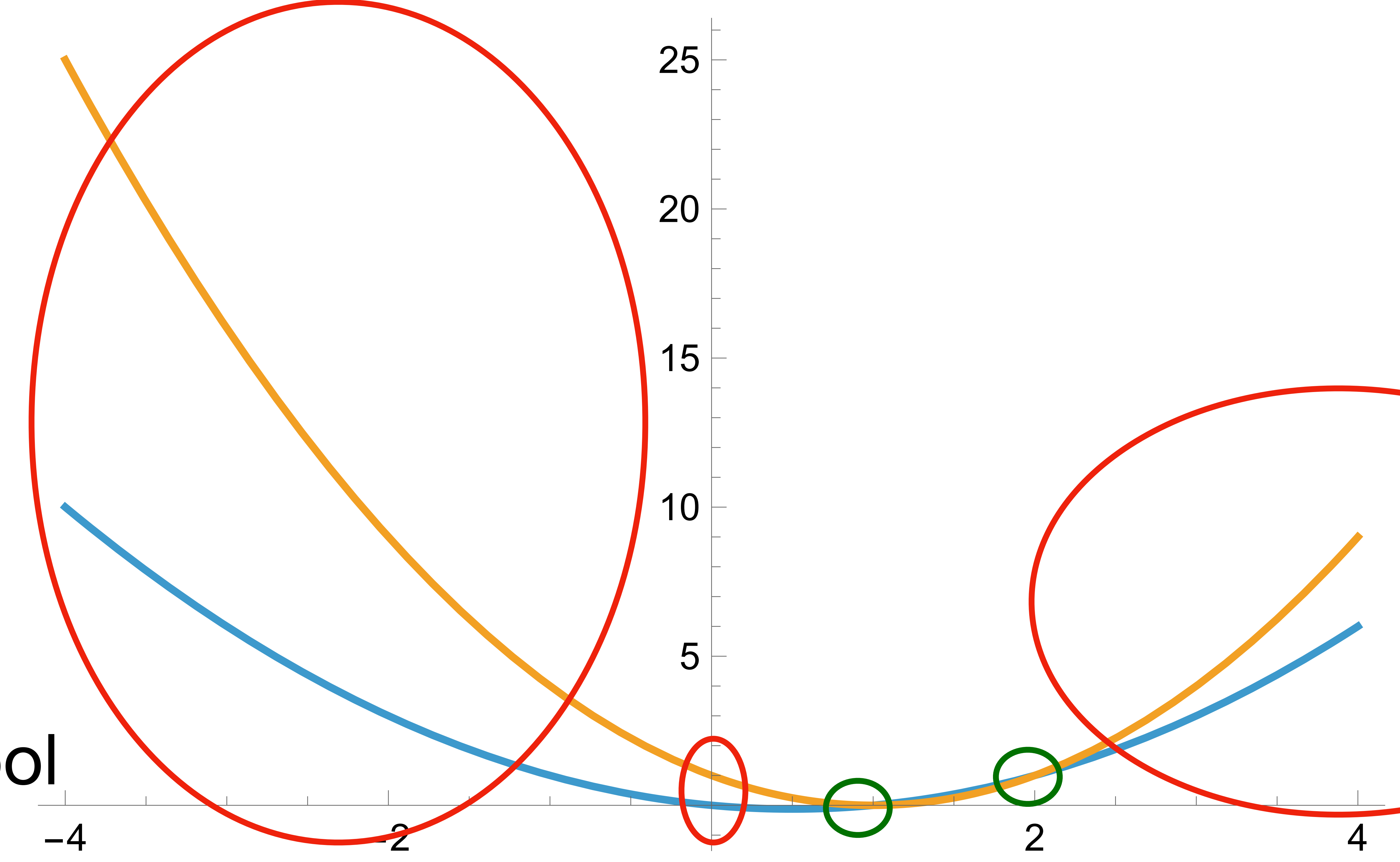
- **Degree mantra:** if $f(X) \neq 0$ then $f(r) \neq 0$ with “high” probability
- Schwartz-Zippel is extremely useful tool
- Intuition why so useful:

On Schwartz-Zippel



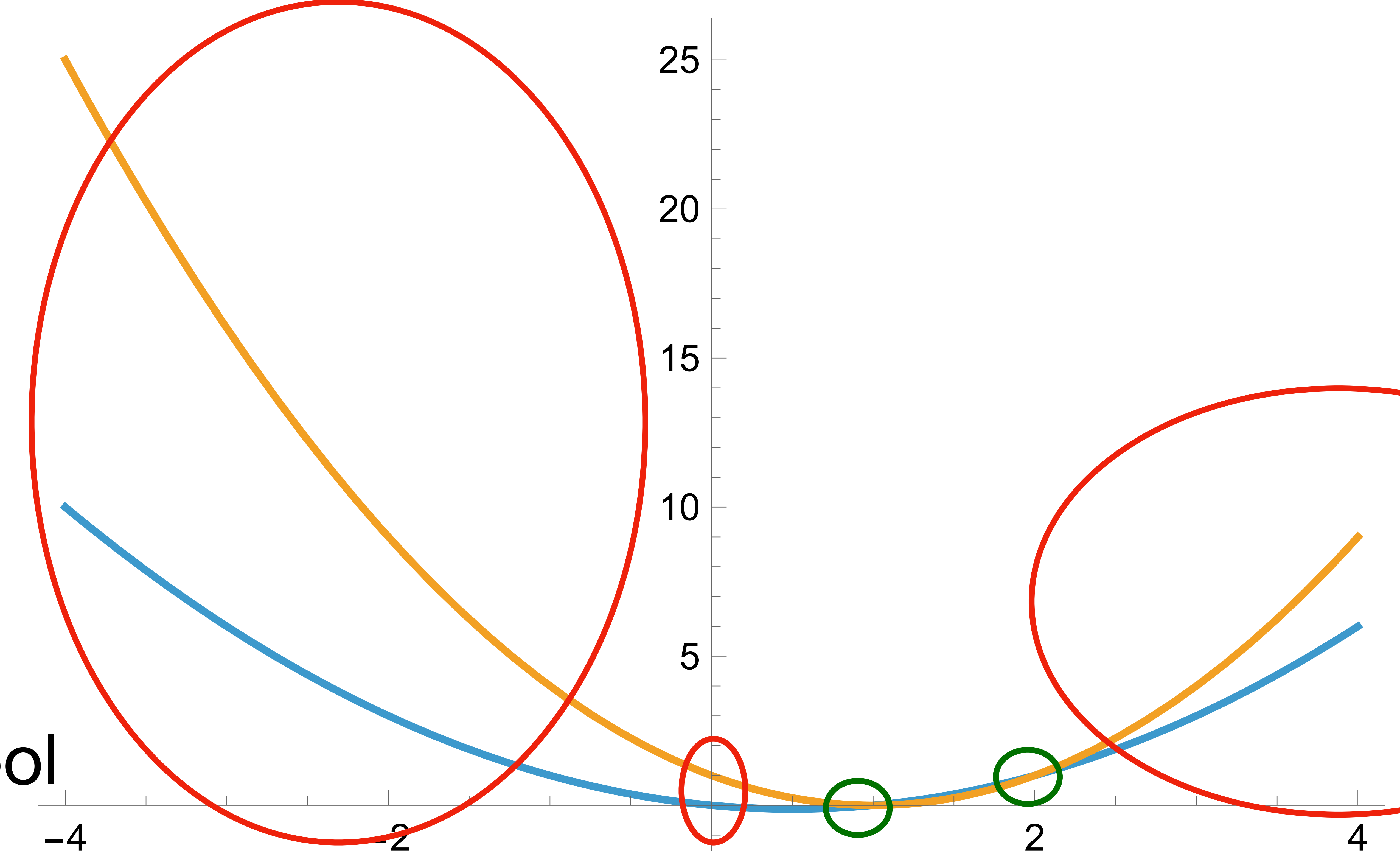
- **Degree mantra:** if $f(X) \neq 0$ then $f(r) \neq 0$ with "high" probability
- Schwartz-Zippel is extremely useful tool
- Intuition why so useful:
  - If $f(X) \in \mathbb{F}_{\leq n}[X]$ and $g(X) \in \mathbb{F}_{\leq n}[X]$ differ at a **single** point, they differ on an **overwhelming** faction of points of $\mathbb{F}$

On Schwartz-Zippel



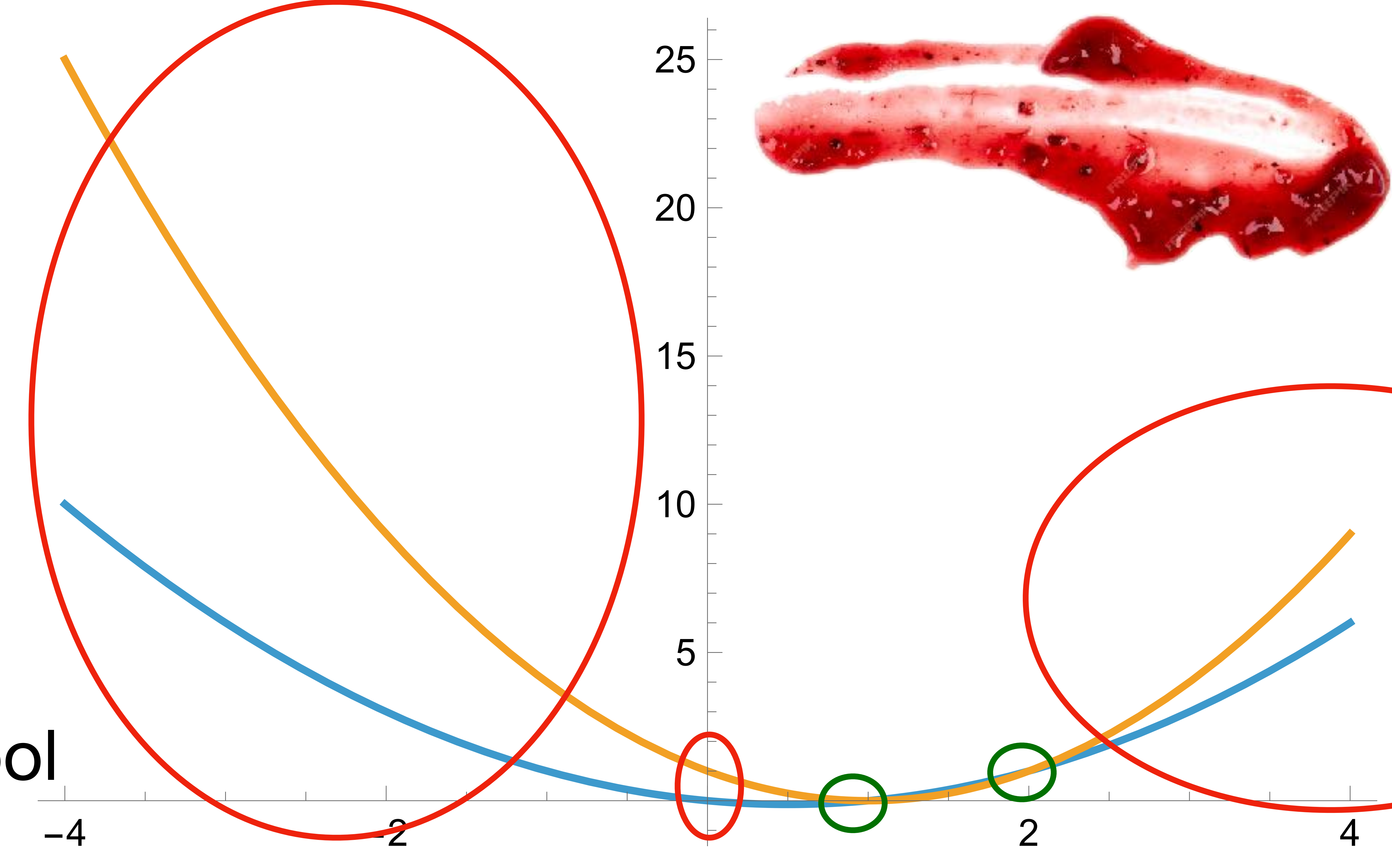
- **Degree mantra:** if $f(X) \neq 0$ then $f(r) \neq 0$ with "high" probability
- Schwartz-Zippel is extremely useful tool
- Intuition why so useful:
  - If $f(X) \in \mathbb{F}_{\leq n}[X]$ and $g(X) \in \mathbb{F}_{\leq n}[X]$ differ at a **single** point, they differ on an **overwhelming** faction of points of $\mathbb{F}$

On Schwartz-Zippel



- **Degree mantra:** if $f(X) \neq 0$ then $f(r) \neq 0$ with "high" probability
- Schwartz-Zippel is extremely useful tool
- Intuition why so useful:
  - If $f(X) \in \mathbb{F}_{\leq n}[X]$ and $g(X) \in \mathbb{F}_{\leq n}[X]$ differ at a **single** point, they differ on an **overwhelming** faction of points of $\mathbb{F}$
  - Thus, if prover cheats even at one point, the verifier can discover the cheating (w.h.p.), querying a random point of the polynomial

On Schwartz-Zippel



- **Degree mantra:** if $f(X) \neq 0$ then $f(r) \neq 0$ with “high” probability
- Schwartz-Zippel is extremely useful tool
- Intuition why so useful:
  - If $f(X) \in \mathbb{F}_{\leq n}[X]$ and $g(X) \in \mathbb{F}_{\leq n}[X]$ differ at a **single** point, they differ on an **overwhelming** faction of points of $\mathbb{F}$
  - Thus, if prover cheats even at one point, the verifier can discover the cheating (w.h.p.), querying a random point of the polynomial
  - “Smears" around the error — akin to error-correcting codes

“Trivial” Zero Check PIOP

$\mathbb{x} = \emptyset, \mathbb{w} = \boldsymbol{a} \in \mathbb{F}^n$

“Trivial” Zero Check PIOP

$\mathbb{x} = \emptyset, \mathbb{w} = \boldsymbol{a} \in \mathbb{F}^n$

$\hat{a}(X) \leftarrow \mathrm{IFFT}(\mathbf{0}) = 0$

$\hat{a}(X)$

$\leq n-1$

“Trivial” Zero Check PIOP

$\mathbb{x} = \emptyset, \mathbb{w} = \boldsymbol{a} \in \mathbb{F}^n$

$\hat{a}(X) \leftarrow \mathrm{IFFT}(\mathbf{0}) = 0$

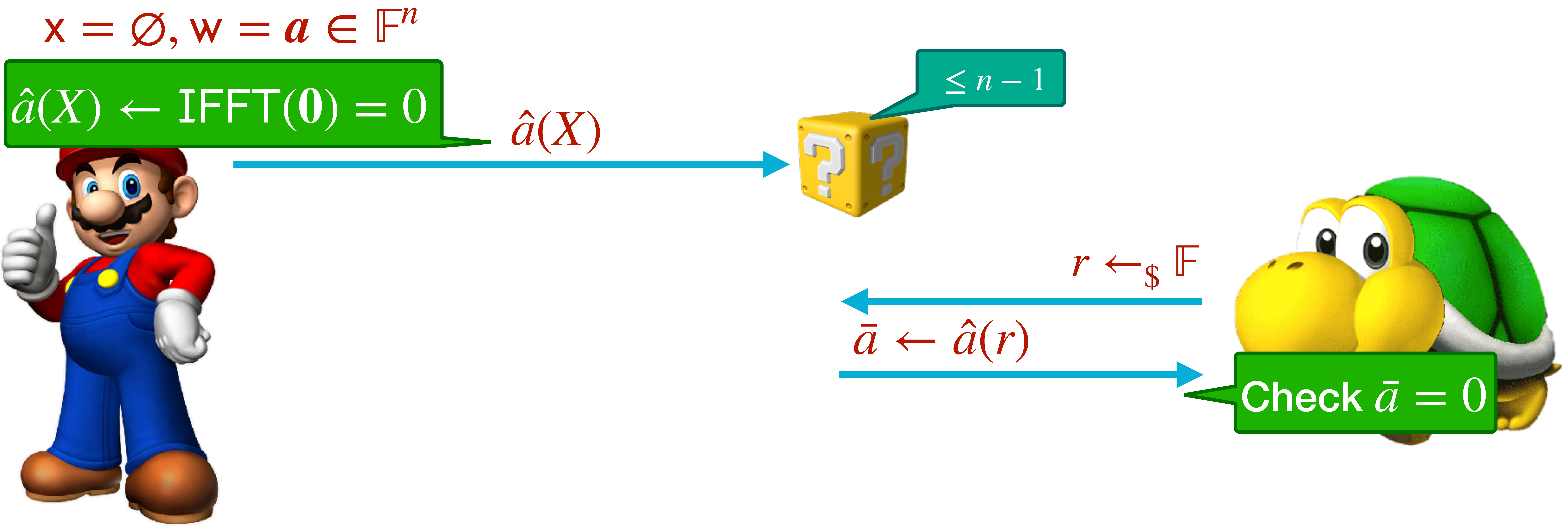
$\hat{a}(X)$

$\leq n - 1$

$r \leftarrow_{\$} \mathbb{F}$

$\bar{a} \leftarrow \hat{a}(r)$

“Trivial” Zero Check PIOP



“Trivial” Zero Check PIOP

$\mathrm{x} = \emptyset, \mathrm{w} = \boldsymbol{a} \in \mathbb{F}^n$

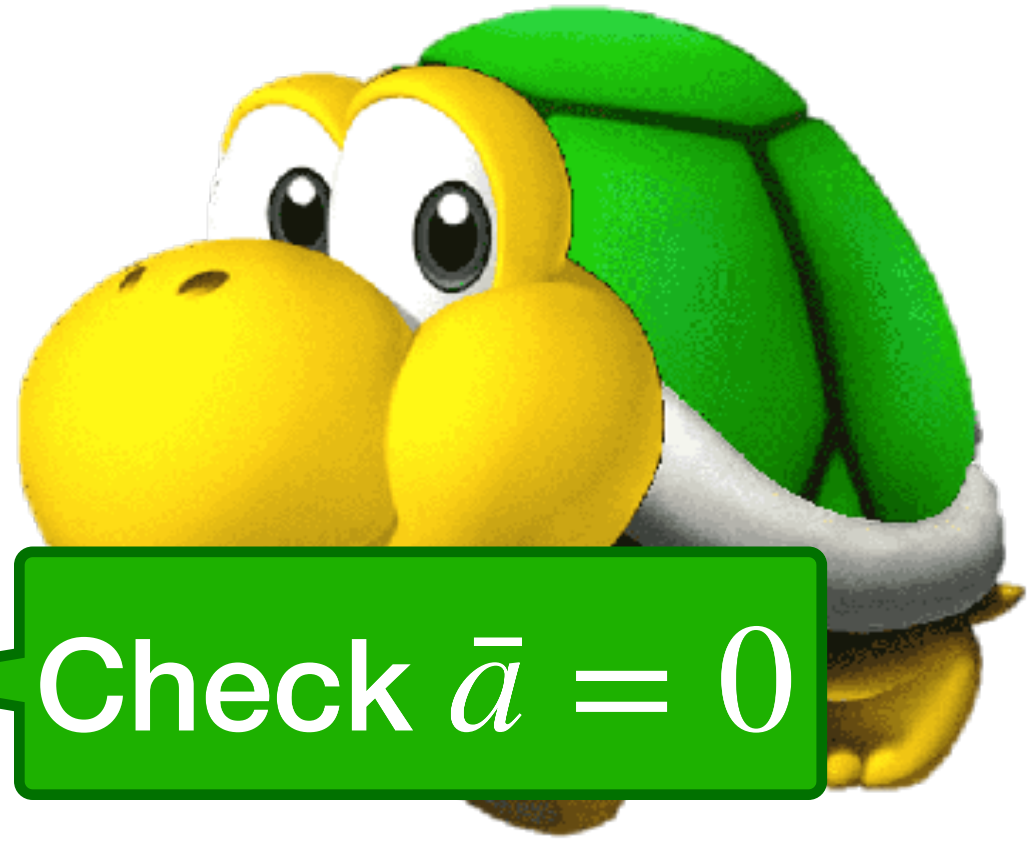
$\hat{a}(X) \leftarrow \mathrm{IFFT}(\mathbf{0}) = 0$

$\hat{a}(X)$

$\leq n-1$

$r \leftarrow_{\$} \mathbb{F}$

$\bar{a} \leftarrow \hat{a}(r)$



Check $\bar{a} = 0$

- Note: the goal of the protocol is to check $\boldsymbol{a} = \mathbf{0}$
- This protocol makes little sense if nothing about $\boldsymbol{a}$ is given as an input to the protocol // what exactly is $= \mathbf{0}$?
- Solution: an oracle $[\![\hat{a}(X)]\!]$ is a part of the input
- $\mathscr{R} = \{(\mathrm{x}, \mathrm{w}) : \mathrm{x} = [\![\hat{a}(X)]\!] \wedge \mathrm{w} = \mathrm{FFT}(\hat{a}(X)) \wedge \mathrm{w} = \mathbf{0}\}$

"Trivial" Zero Check PIOP

$\mathbb{x} = [\![\hat{a}(X)]\!], \mathbb{w} = \boldsymbol{a} = \mathbf{0} \in \mathbb{F}^n$

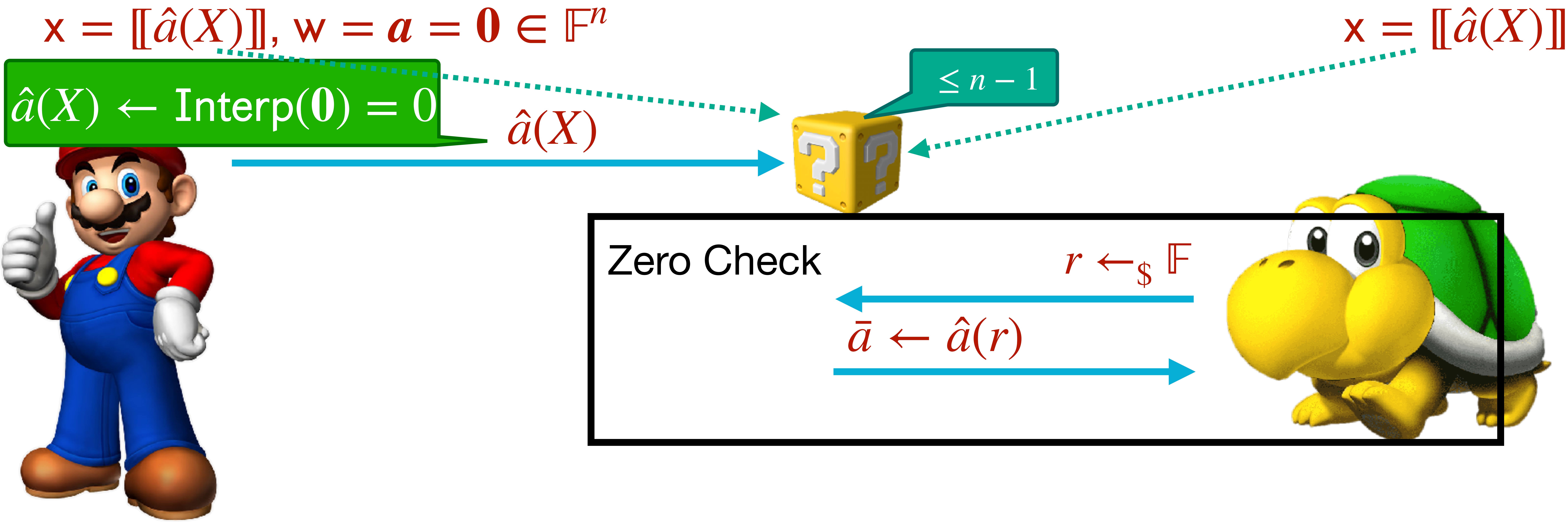
$\hat{a}(X) \leftarrow \mathrm{Interp}(\mathbf{0}) = 0$

$\hat{a}(X)$

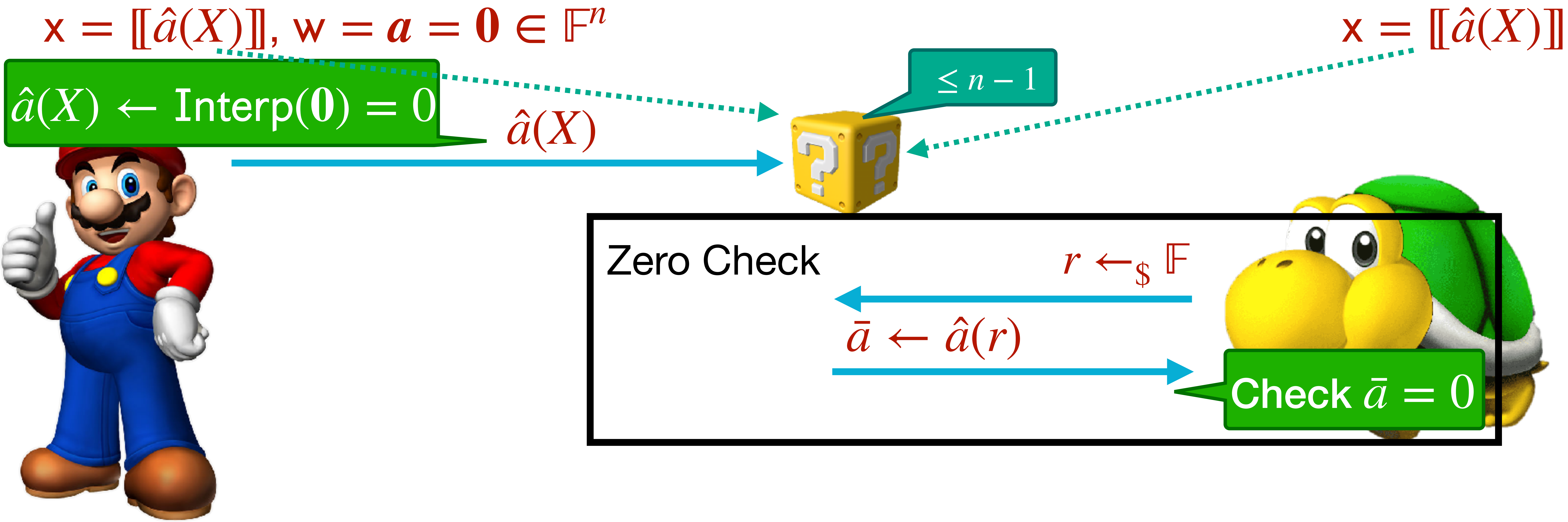
$\leq n-1$

$\mathbb{x} = [\![\hat{a}(X)]\!]$

"Trivial" Zero Check PIOP



"Trivial" Zero Check PIOP



"Trivial" Zero Check PIOP

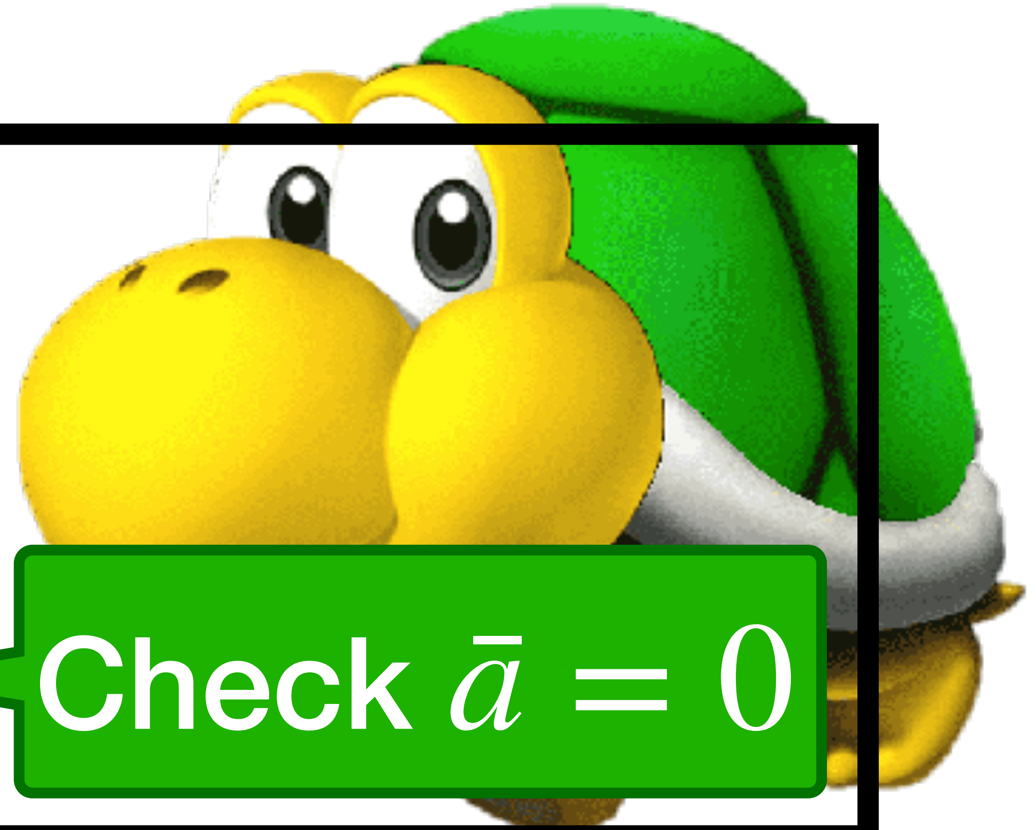
$\mathsf{x} = [\![\hat{a}(X)]\!], \mathsf{w} = \boldsymbol{a} = \mathbf{0} \in \mathbb{F}^n$

$\mathsf{x} = [\![\hat{a}(X)]\!]$

$\hat{a}(X) \leftarrow \mathsf{Interp}(\mathbf{0}) = 0$

$\hat{a}(X)$

$\leq n-1$



Zero Check

$r \leftarrow_{\$} \mathbb{F}$

$\bar{a} \leftarrow \hat{a}(r)$

Check $\bar{a} = 0$

- $\mathscr{R} = \{(\mathsf{x}, \mathsf{w}) : \mathsf{x} = [\![\hat{a}(X)]\!] \wedge \mathsf{w} = \mathsf{FFT}(\hat{a}(X)) \wedge \mathsf{w} = \mathbf{0}\}$
- In general, PIOP is a **proof of knowledge** of **knowing** the **contents of the oracles** that satisfy some relation
- In zk-SNARKs, when replacing oracles with commitments, we get a proof of knowledge of knowing the contents of the **commitments** that satisfy some relation

Product Check

“Virtual” Zero Check

- Zero check is always used as a subroutine (not as a separate goal)

“Virtual” Zero Check

- Zero check is always used as a subroutine (not as a separate goal)
- Simple example use cases:

“Virtual” Zero Check

- Zero check is always used as a subroutine (not as a separate goal)
- Simple example use cases:
    - **Boolean check:** given input vector $\boldsymbol{a}$, check $\forall i \,.\, a_i(a_i - 1) = 0$

“Virtual” Zero Check

- Zero check is always used as a subroutine (not as a separate goal)
- Simple example use cases:
    - **Boolean check:** given input vector $\boldsymbol{a}$, check $\forall i \,.\, a_i(a_i - 1) = 0$
    - **Addition check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i + b_i - c_i = 0$

“Virtual” Zero Check

- Zero check is always used as a subroutine (not as a separate goal)
- Simple example use cases:
  - **Boolean check:** given input vector $\boldsymbol{a}$, check $\forall i \,.\, a_i(a_i - 1) = 0$
  - **Addition check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i + b_i - c_i = 0$
  - **Product check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i b_i - c_i = 0$

“Virtual” Zero Check

- Zero check is always used as a subroutine (not as a separate goal)
- Simple example use cases:
    - **Boolean check:** given input vector $\boldsymbol{a}$, check $\forall i \, . \, a_i(a_i - 1) = 0$
    - **Addition check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \, . \, a_i + b_i - c_i = 0$
    - **Product check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \, . \, a_i b_i - c_i = 0$
- Above described zero check works with degree- $\leq (n-1)$ polynomials

“Virtual” Zero Check

- Zero check is always used as a subroutine (not as a separate goal)
- Simple example use cases:
    - **Boolean check:** given input vector $\boldsymbol{a}$, check $\forall i \,.\, a_i(a_i - 1) = 0$
    - **Addition check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i + b_i - c_i = 0$
    - **Product check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i b_i - c_i = 0$
- Above described zero check works with degree- $\leq (n-1)$ polynomials
- Intermediate polyn’s in other checks can have higher degree than $|\mathbb{H}| = n$

“Virtual” Zero Check

- Zero check is always used as a subroutine (not as a separate goal)
- Simple example use cases:
    - **Boolean check:** given input vector $\boldsymbol{a}$, check $\forall i \,.\, a_i(a_i - 1) = 0$
    - **Addition check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i + b_i - c_i = 0$
    - **Product check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i b_i - c_i = 0$
- Above described zero check works with degree- $\leq (n-1)$ polynomials
- Intermediate polyn’s in other checks can have higher degree than $|\mathbb{H}| = n$
    - Product check has **virtual** oracle $\hat{a}(X)\hat{b}(X) - \hat{c}(X)$ of degree $\leq 2n - 1$

“Virtual” Zero Check

- Zero check is always used as a subroutine (not as a separate goal)
- Simple example use cases:
  - **Boolean check:** given input vector $\boldsymbol{a}$, check $\forall i \,.\, a_i(a_i - 1) = 0$
  - **Addition check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i + b_i - c_i = 0$
  - **Product check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i b_i - c_i = 0$
- Above described zero check works with degree- $\leq (n-1)$ polynomials
- Intermediate polyn’s in other checks can have higher degree than $|\mathbb{H}| = n$
  - Product check has **virtual** oracle $\hat{a}(X)\hat{b}(X) - \hat{c}(X)$ of degree $\leq 2n - 1$
  - Does not fit into n-degree polynomial oracle!

"Virtual" Zero Check

- Zero check is always used as a subroutine (not as a separate goal)
- Simple example use cases:
  - **Boolean check:** given input vector $\boldsymbol{a}$, check $\forall i \,.\, a_i(a_i - 1) = 0$
  - **Addition check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i + b_i - c_i = 0$
  - **Product check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i b_i - c_i = 0$
- Above described zero check works with degree- $\leq (n-1)$ polynomials
- Intermediate polyn's in other checks can have higher degree than $|\mathbb{H}| = n$
  - Product check has **virtual** oracle $\hat{a}(X)\hat{b}(X) - \hat{c}(X)$ of degree $\leq 2n - 1$
  - Does not fit into n-degree polynomial oracle!
- Need to modify zero check to work with high-degree "virtual" oracles

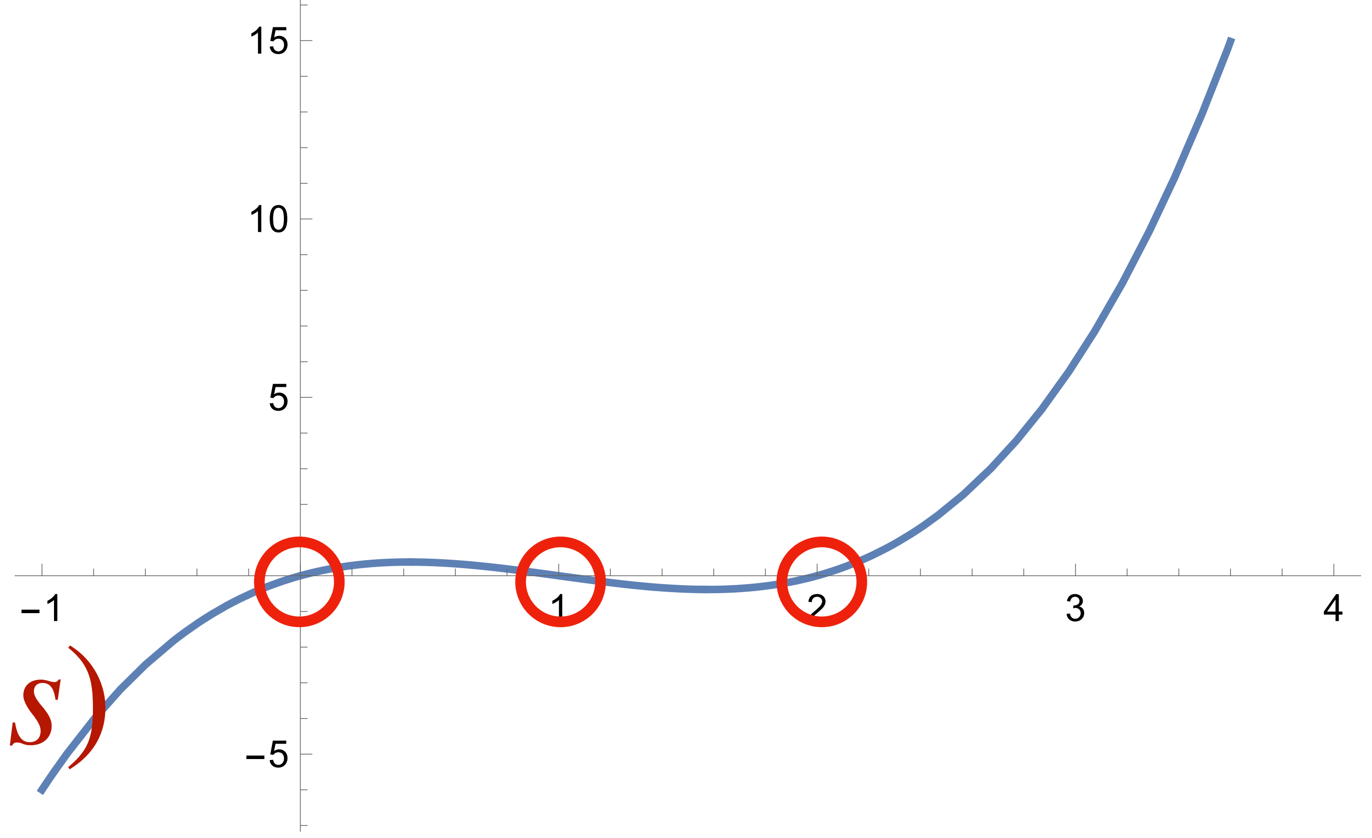
"Virtual" Zero Check

- Zero check is always used as a subroutine (not as a separate goal)
- Simple example use cases:
    - **Boolean check:** given input vector $\boldsymbol{a}$, check $\forall i \,.\, a_i(a_i - 1) = 0$
    - **Addition check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i + b_i - c_i = 0$
    - **Product check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i \,.\, a_i b_i - c_i = 0$
- Above described zero check works with degree- $\leq (n-1)$ polynomials
- Intermediate polyn's in other checks can have higher degree than $|\mathbb{H}| = n$
    - Product check has **virtual** oracle $\hat{a}(X)\hat{b}(X) - \hat{c}(X)$ of degree $\leq 2n - 1$
    - Does not fit into n-degree polynomial oracle!
- Need to modify zero check to work with high-degree "virtual" oracles
- We will give a concrete example for "product check"

“Virtual” Zero Check

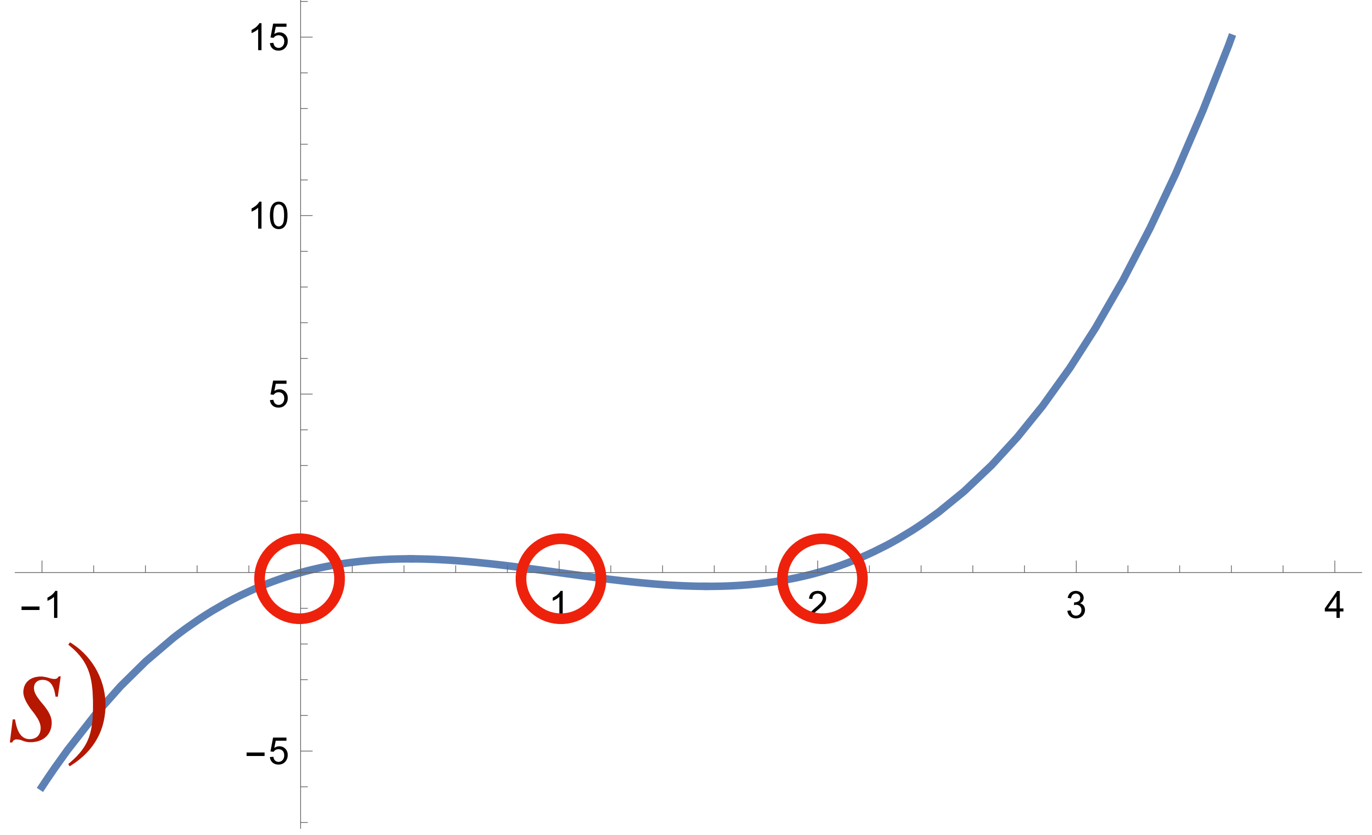
- Zero check is always used as a subroutine (not as a separate goal)
- Simple example use cases:
  - **Boolean check:** given input vector $\boldsymbol{a}$, check $\forall i\,.\,a_i(a_i - 1) = 0$
  - **Addition check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i\,.\,a_i + b_i - c_i = 0$
  - **Product check:** given input vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, check $\forall i\,.\,a_i b_i - c_i = 0$
- Above described zero check works with degree- $\leq (n-1)$ polynomials
- Intermediate polyn’s in other checks can have higher degree than $|\mathbb{H}| = n$
  - Product check has **virtual** oracle $\hat{a}(X)\hat{b}(X) - \hat{c}(X)$ of degree $\leq 2n - 1$
  - Does not fit into n-degree polynomial oracle!
- Need to modify zero check to work with high-degree “virtual" oracles
- We will give a concrete example for “product check”
- In addition, adding ZK will increase the degree of “virtual" oracles

Vanishing Polynomials



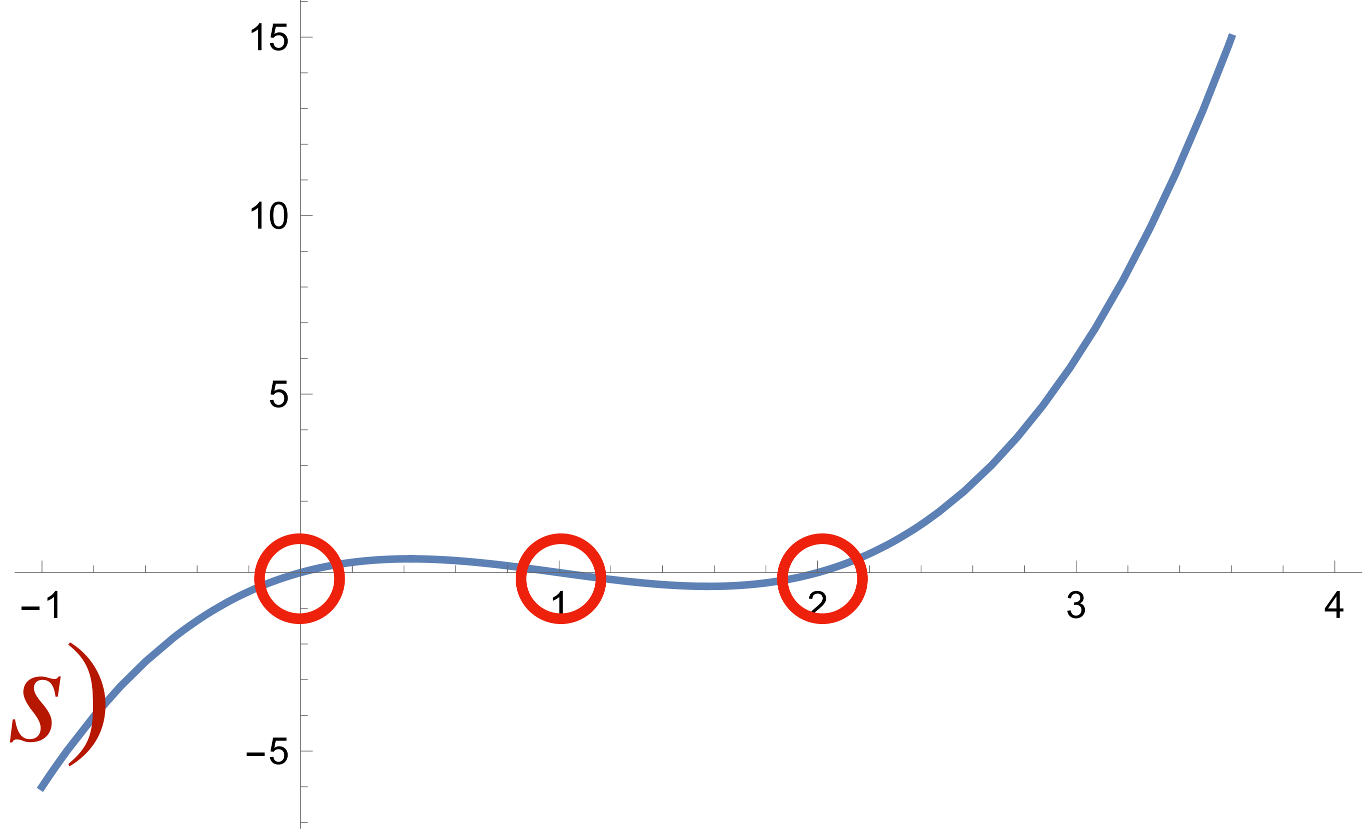
- **Defn. Vanishing polynomial of set** $\mathcal{S}$: $\mathbf{Z}_{\mathcal{S}}(X) := \prod_{s \in \mathcal{S}} (X - s)$

Vanishing Polynomials



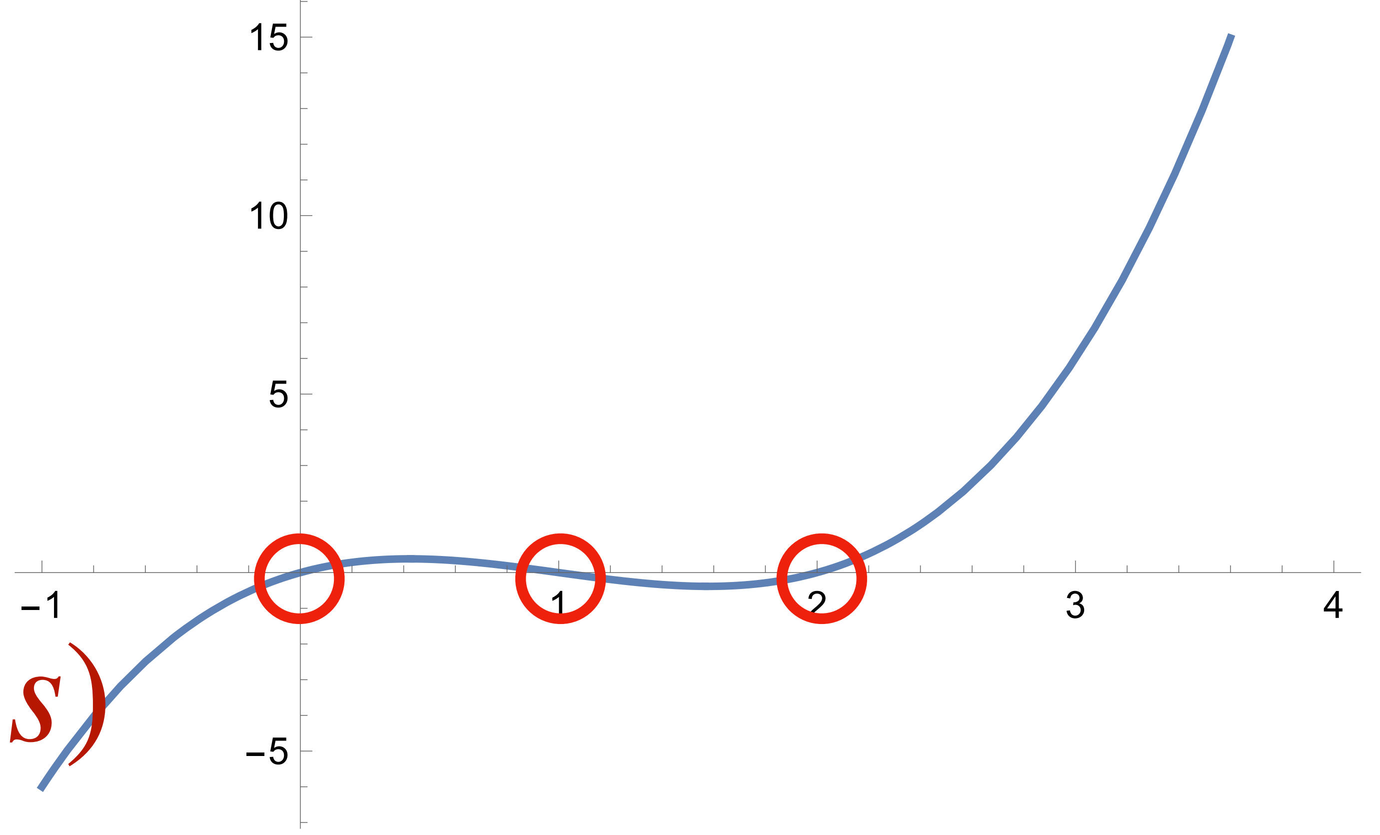
- **Defn. Vanishing polynomial of set** $\mathcal{S}$: $\mathbf{Z}_{\mathcal{S}}(X) := \prod_{s\in\mathcal{S}}(X-s)$
  - $\mathbf{Z}_{\mathbb{H}}(s) = 0$ for $s \in S$, $\mathbf{Z}_{\mathbb{H}}(s) \neq 0$ otherwise; $\deg(\mathbf{Z}_{\mathcal{S}}) = n$

Vanishing Polynomials



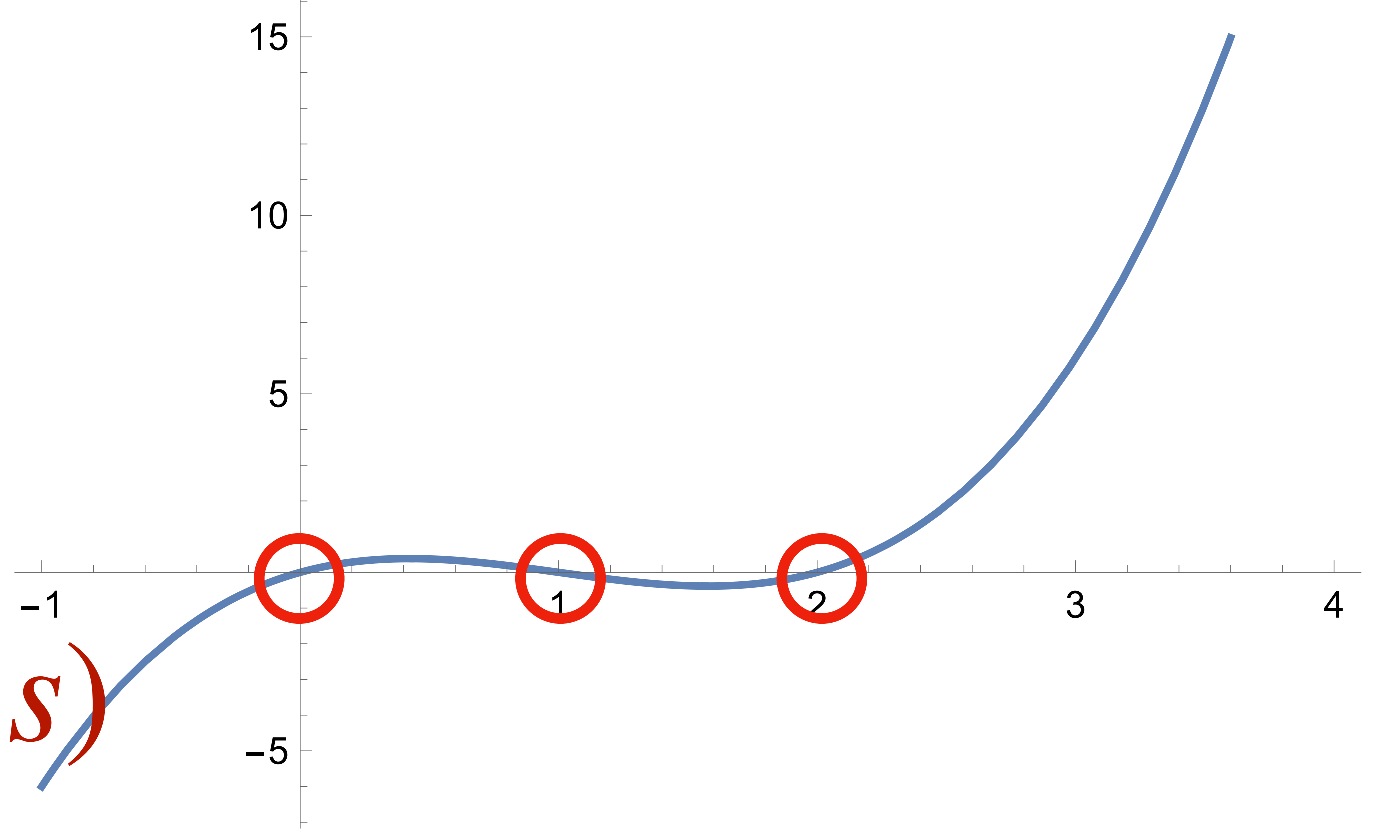
- **Defn. Vanishing polynomial of set** $\mathcal{S}$: $\mathbf{Z}_{\mathcal{S}}(X) := \prod_{s \in \mathcal{S}} (X - s)$
  - $\mathbf{Z}_{\mathbb{H}}(s) = 0$ for $s \in S$, $\mathbf{Z}_{\mathbb{H}}(s) \neq 0$ otherwise; $\deg(\mathbf{Z}_{\mathcal{S}}) = n$
- Any polynomial $f(X)$ that vanishes on $\mathcal{S}$ has all $s_i \in \mathcal{S}$ as roots

Vanishing Polynomials



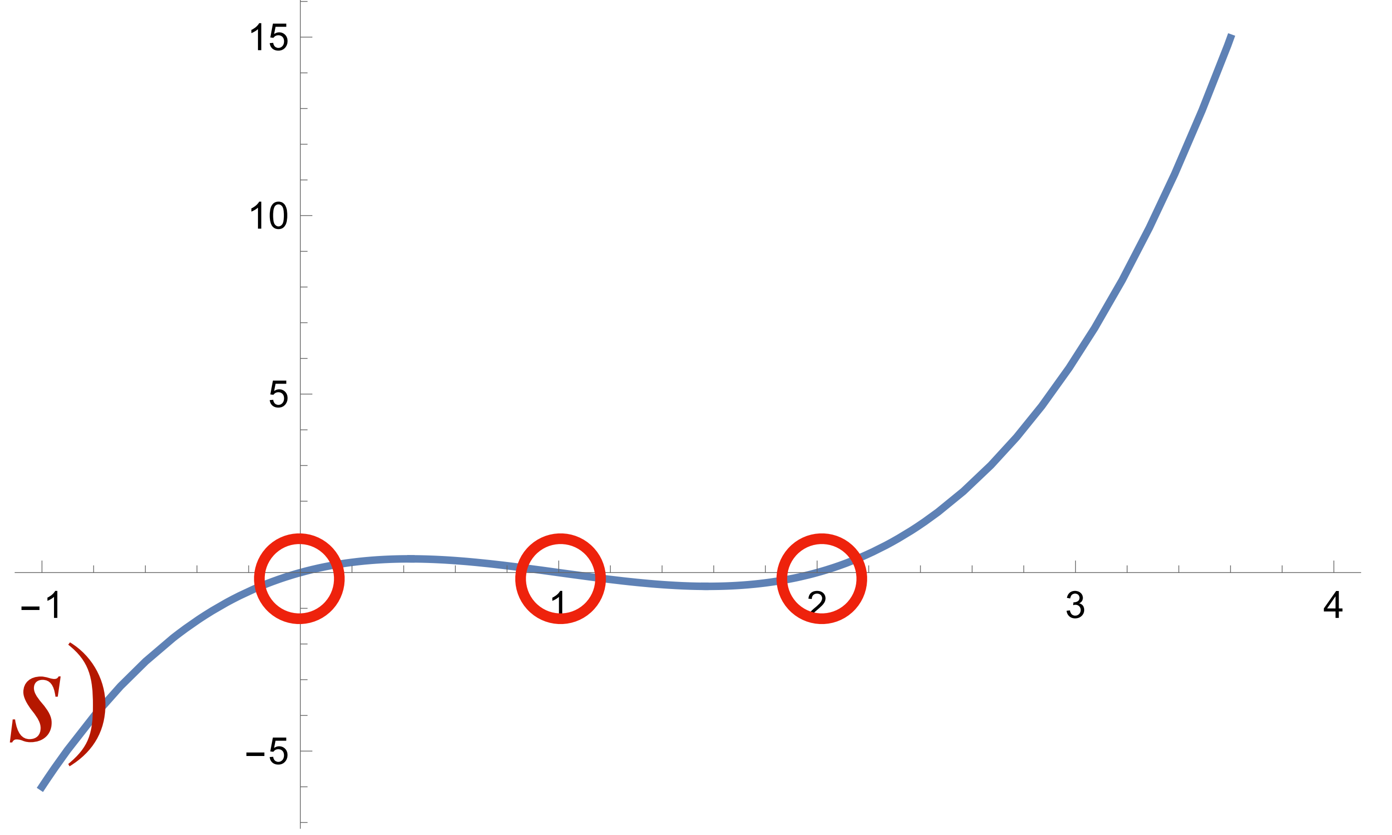
- **Defn. Vanishing polynomial of set** $\mathcal{S}$: $\mathbf{Z}_{\mathcal{S}}(X) := \prod_{s \in \mathcal{S}} (X - s)$
  - $\mathbf{Z}_{\mathbb{H}}(s) = 0$ for $s \in S$, $\mathbf{Z}_{\mathbb{H}}(s) \neq 0$ otherwise; $\deg(\mathbf{Z}_{\mathcal{S}}) = n$
- Any polynomial $f(X)$ that vanishes on $\mathcal{S}$ has all $s_i \in \mathcal{S}$ as roots
- Since $(X - s) \mid f(X)$ for all $s \in \mathcal{S}$ => $\mathbf{Z}_{\mathcal{S}}(X) \mid f(X)$

Vanishing Polynomials



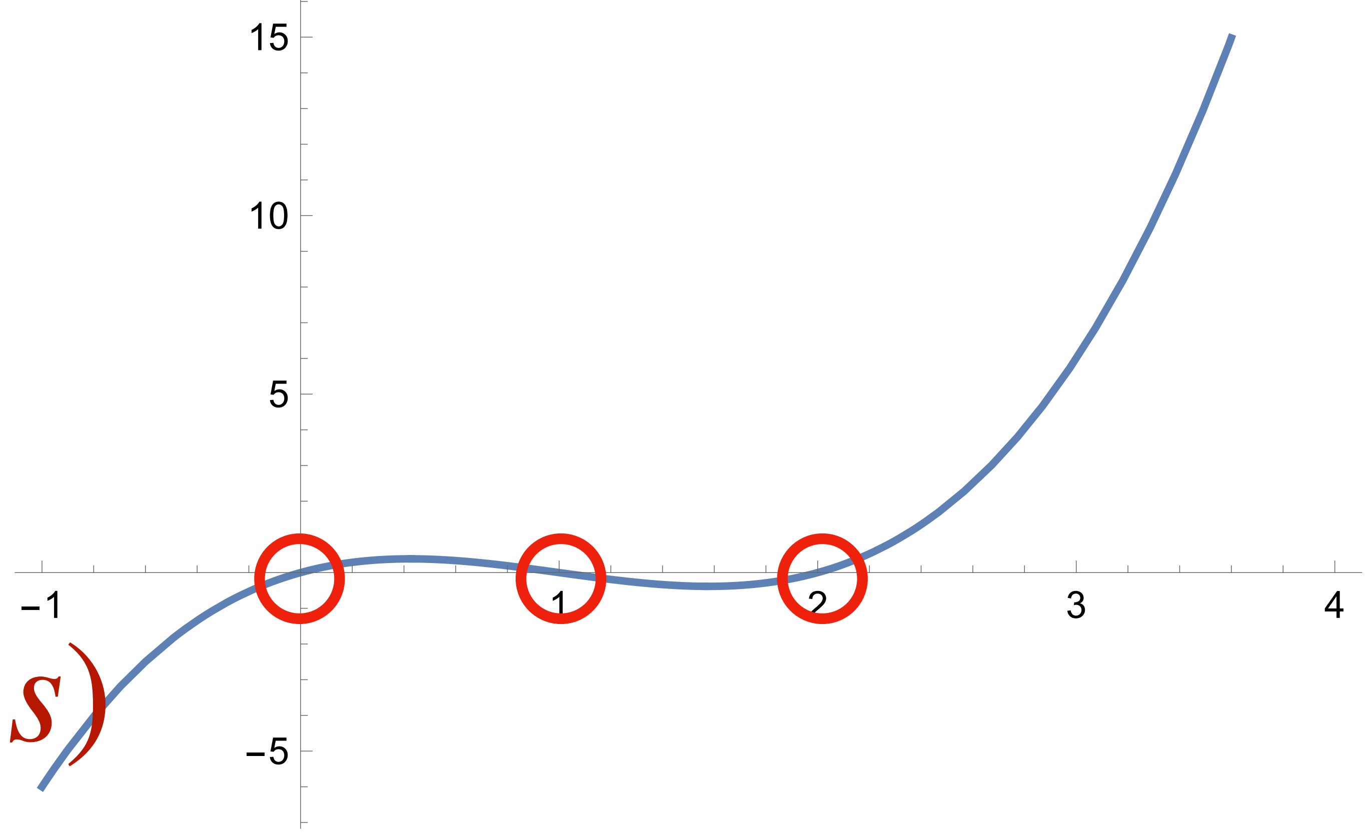
- **Defn. Vanishing polynomial of set** $\mathcal{S}$: $\mathbf{Z}_{\mathcal{S}}(X) := \prod_{s \in \mathcal{S}} (X - s)$
  - $\mathbf{Z}_{\mathbb{H}}(s) = 0$ for $s \in S$, $\mathbf{Z}_{\mathbb{H}}(s) \neq 0$ otherwise; $\deg(\mathbf{Z}_{\mathcal{S}}) = n$
- Any polynomial $f(X)$ that vanishes on $\mathcal{S}$ has all $s_i \in \mathcal{S}$ as roots
- Since $(X - s) \mid f(X)$ for all $s \in \mathcal{S}$ => $\mathbf{Z}_{\mathcal{S}}(X) \mid f(X)$
  - Assume $f(s) = 0$. Use polynomial long division (Extended Euclidean) to write $f(X) = q(X)(X - s) + r$ for a polynomial $q(X)$ and remainder $r \in \mathbb{F}$.

Vanishing Polynomials



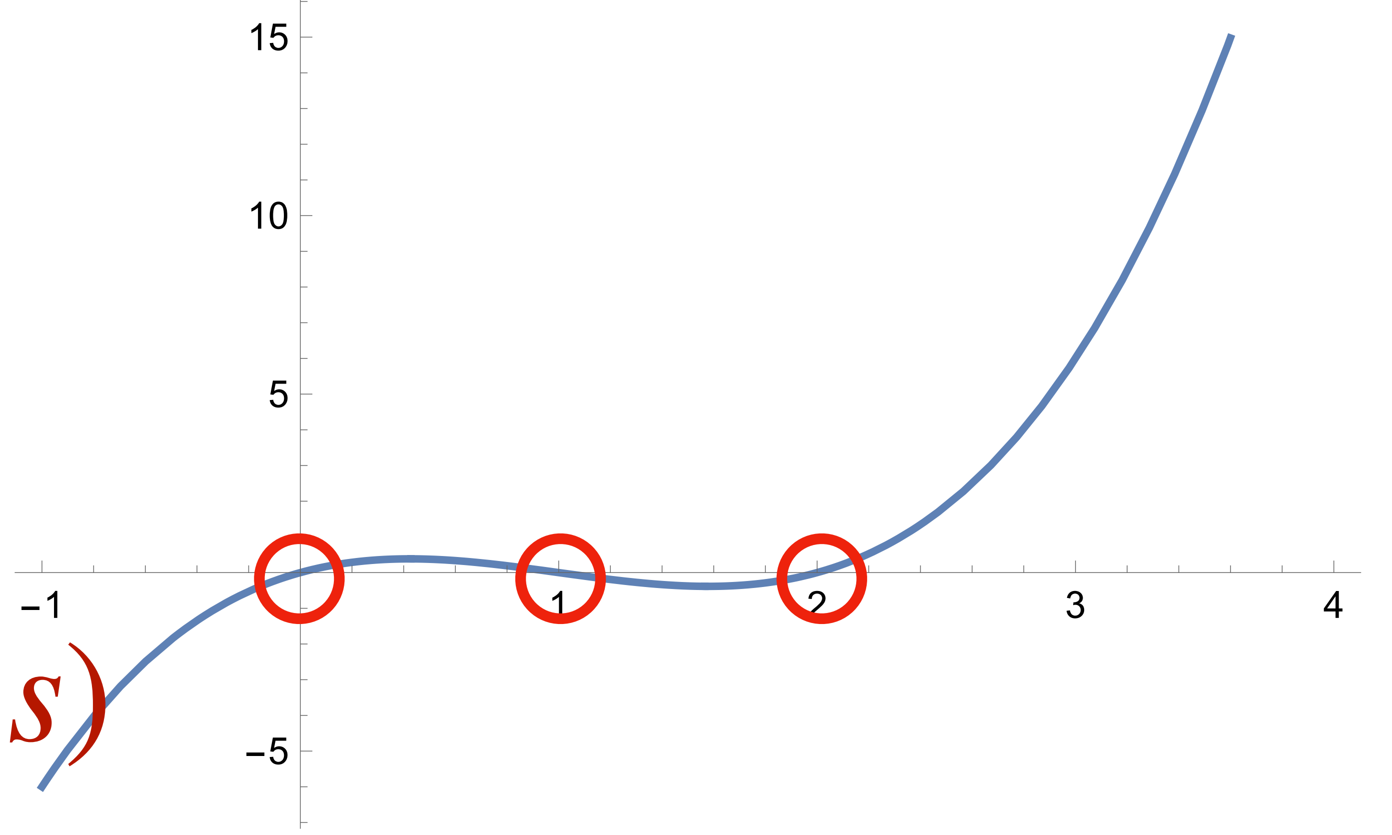
- **Defn. Vanishing polynomial of set** $\mathcal{S}$: $\mathbf{Z}_{\mathcal{S}}(X) := \prod_{s \in \mathcal{S}} (X - s)$
  - $\mathbf{Z}_{\mathbb{H}}(s) = 0$ for $s \in S$, $\mathbf{Z}_{\mathbb{H}}(s) \neq 0$ otherwise; $\deg(\mathbf{Z}_{\mathcal{S}}) = n$
- Any polynomial $f(X)$ that vanishes on $\mathcal{S}$ has all $s_i \in \mathcal{S}$ as roots
- Since $(X - s) \mid f(X)$ for all $s \in \mathcal{S}$ => $\mathbf{Z}_{\mathcal{S}}(X) \mid f(X)$
  - Assume $f(s) = 0$. Use polynomial long division (Extended Euclidean) to write $f(X) = q(X)(X - s) + r$ for a polynomial $q(X)$ and remainder $r \in \mathbb{F}$.
  - Evaluating LHS and RHS at $X = s$, we get $f(s) = r$, thus $r = 0$

Vanishing Polynomials



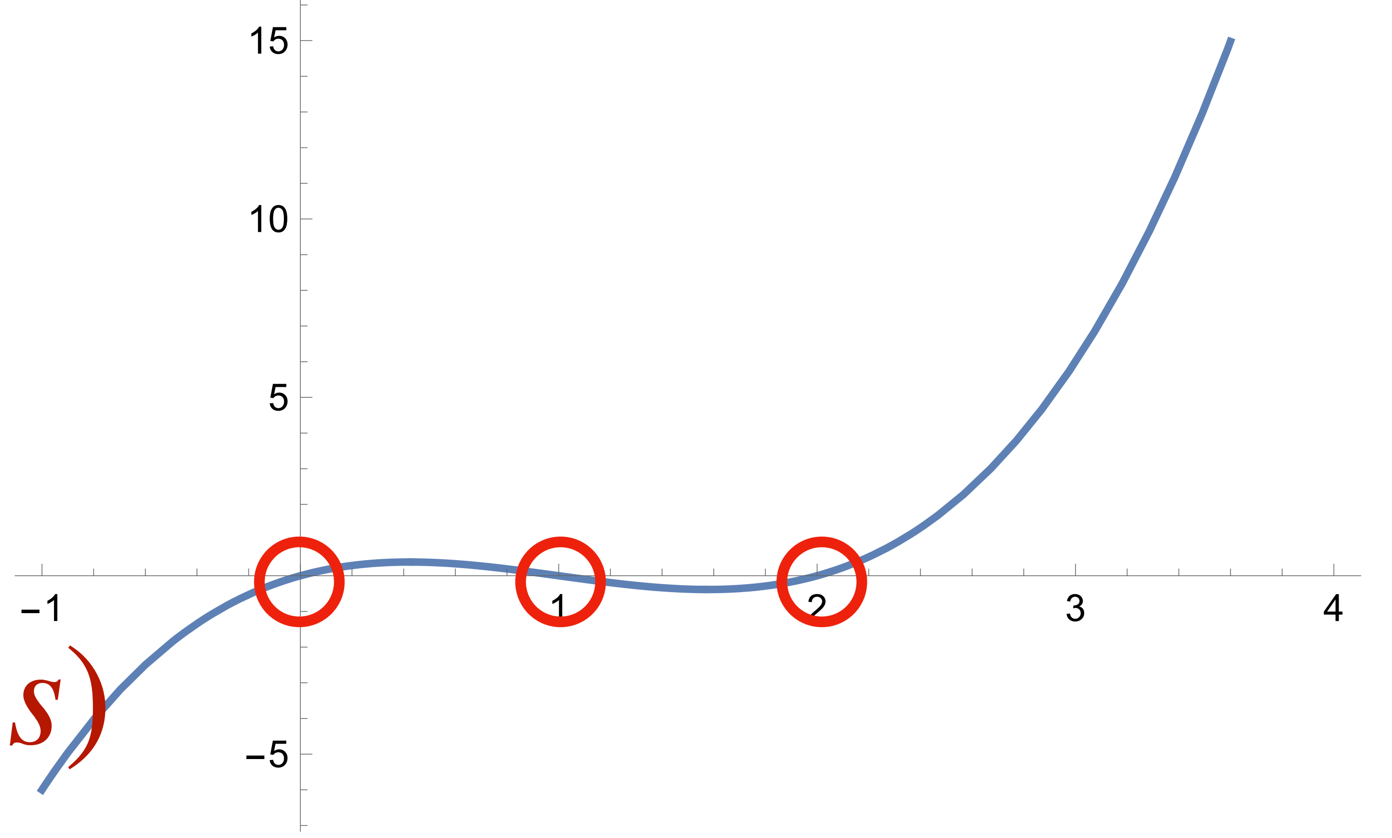
- **Defn. Vanishing polynomial of set $\mathcal{S}$:** $\mathbf{Z}_{\mathcal{S}}(X) := \prod_{s \in \mathcal{S}} (X - s)$
  - $\mathbf{Z}_{\mathbb{H}}(s) = 0$ for $s \in S$, $\mathbf{Z}_{\mathbb{H}}(s) \neq 0$ otherwise; $\deg(\mathbf{Z}_{\mathcal{S}}) = n$
- Any polynomial $f(X)$ that vanishes on $\mathcal{S}$ has all $s_i \in \mathcal{S}$ as roots
- Since $(X - s) \mid f(X)$ for all $s \in \mathcal{S}$ => $\mathbf{Z}_{\mathcal{S}}(X) \mid f(X)$
  - Assume $f(s) = 0$. Use polynomial long division (Extended Euclidean) to write $f(X) = q(X)(X - s) + r$ for a polynomial $q(X)$ and remainder $r \in \mathbb{F}$.
  - Evaluating LHS and RHS at $X = s$, we get $f(s) = r$, thus $r = 0$
  - Thus, $f(X) = q(X)(X - s)$ and $(X - s) \mid f(X)$

Vanishing Polynomials



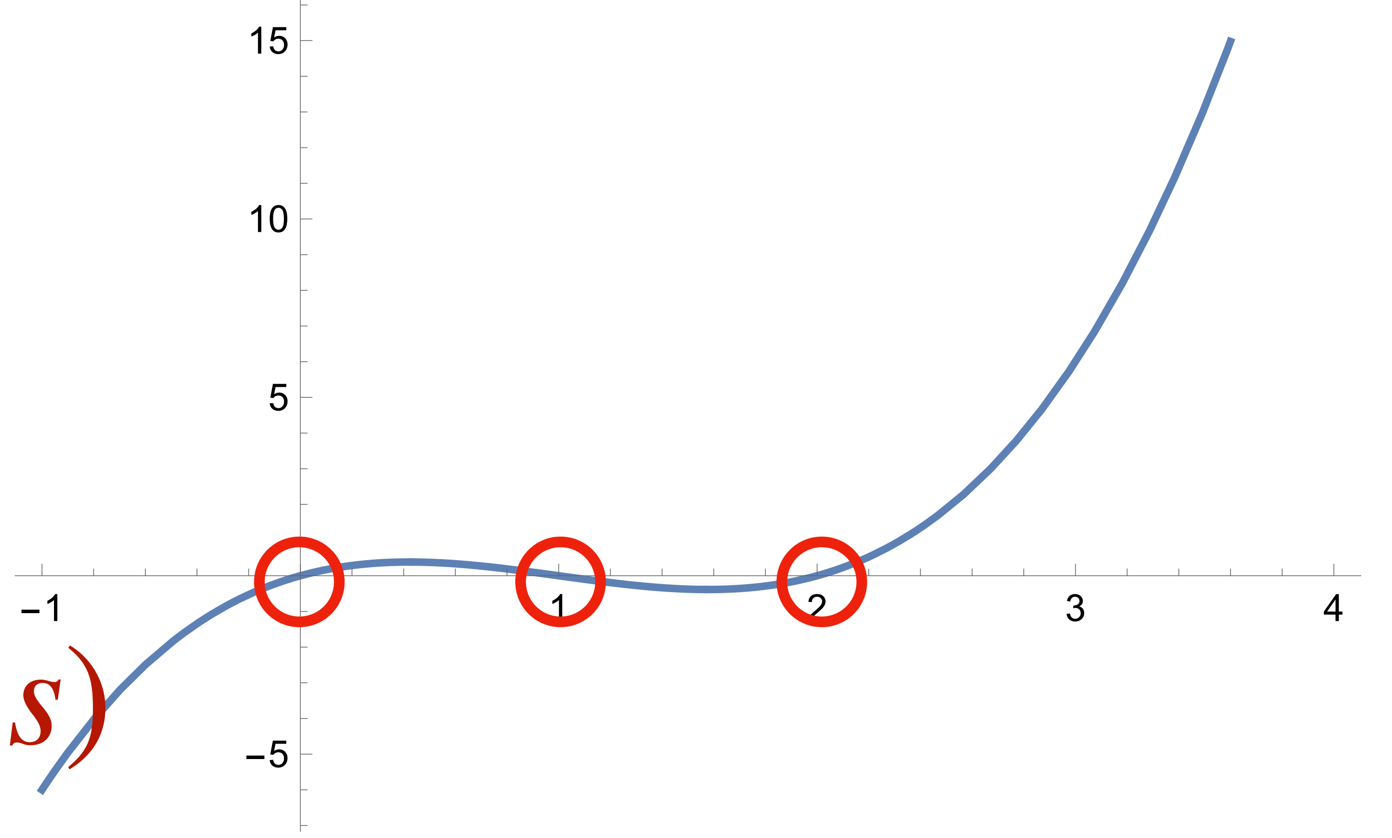
- **Defn. Vanishing polynomial of set** $\mathcal{S}$: $\mathbf{Z}_{\mathcal{S}}(X) := \prod_{s \in \mathcal{S}} (X - s)$
  - $\mathbf{Z}_{\mathbb{H}}(s) = 0$ for $s \in S$, $\mathbf{Z}_{\mathbb{H}}(s) \neq 0$ otherwise; $\deg(\mathbf{Z}_{\mathcal{S}}) = n$
- Any polynomial $f(X)$ that vanishes on $\mathcal{S}$ has all $s_i \in \mathcal{S}$ as roots
- Since $(X - s) \mid f(X)$ for all $s \in \mathcal{S}$ => $\mathbf{Z}_{\mathcal{S}}(X) \mid f(X)$
  - Assume $f(s) = 0$. Use polynomial long division (Extended Euclidean) to write $f(X) = q(X)(X - s) + r$ for a polynomial $q(X)$ and remainder $r \in \mathbb{F}$.
  - Evaluating LHS and RHS at $X = s$, we get $f(s) = r$, thus $r = 0$
  - Thus, $f(X) = q(X)(X - s)$ and $(X - s) \mid f(X)$
- **Lemma.** If $\deg(f) = N > n - 1$ and $f(X)$ vanishes on $\mathcal{S}$, then $f(X) = q(X)\mathbf{Z}_{\mathcal{S}}(X)$ for some $q(X) \in \mathbb{F}_{\leq N-n}[X]$

Vanishing Polynomials



- **Defn. Vanishing polynomial of set** $\mathcal{S}$: $\mathbf{Z}_{\mathcal{S}}(X) := \prod_{s \in \mathcal{S}} (X - s)$
  - $\mathbf{Z}_{\mathbb{H}}(s) = 0$ for $s \in S$, $\mathbf{Z}_{\mathbb{H}}(s) \neq 0$ otherwise; $\deg(\mathbf{Z}_{\mathcal{S}}) = n$
- Any polynomial $f(X)$ that vanishes on $\mathcal{S}$ has all $s_i \in \mathcal{S}$ as roots
- Since $(X - s) \mid f(X)$ for all $s \in \mathcal{S}$ => $\mathbf{Z}_{\mathcal{S}}(X) \mid f(X)$
  - Assume $f(s) = 0$. Use polynomial long division (Extended Euclidean) to write $f(X) = q(X)(X - s) + r$ for a polynomial $q(X)$ and remainder $r \in \mathbb{F}$.
  - Evaluating LHS and RHS at $X = s$, we get $f(s) = r$, thus $r = 0$
  - Thus, $f(X) = q(X)(X - s)$ and $(X - s) \mid f(X)$
- **Lemma.** If $\deg(f) = N > n - 1$ and $f(X)$ vanishes on $\mathcal{S}$, then $f(X) = q(X)\mathbf{Z}_{\mathcal{S}}(X)$ for some $q(X) \in \mathbb{F}_{\leq N-n}[X]$
- $\mathbf{Z}_{\mathcal{S}}$ is **unique, minimal-degree, monic, non-zero** poly that vanishes exactly on $\mathcal{S}$

Vanishing Polynomials



- **Defn. Vanishing polynomial of set $\mathcal{S}$**: $\mathbf{Z}_{\mathcal{S}}(X) := \prod_{s \in \mathcal{S}} (X - s)$
  - $\mathbf{Z}_{\mathbb{H}}(s) = 0$ for $s \in S$, $\mathbf{Z}_{\mathbb{H}}(s) \neq 0$ otherwise; $\deg(\mathbf{Z}_{\mathcal{S}}) = n$
- Any polynomial $f(X)$ that vanishes on $\mathcal{S}$ has all $s_i \in \mathcal{S}$ as roots
- Since $(X - s) \mid f(X)$ for all $s \in \mathcal{S}$ => $\mathbf{Z}_{\mathcal{S}}(X) \mid f(X)$
  - Assume $f(s) = 0$. Use polynomial long division (Extended Euclidean) to write $f(X) = q(X)(X - s) + r$ for a polynomial $q(X)$ and remainder $r \in \mathbb{F}$.
  - Evaluating LHS and RHS at $X = s$, we get $f(s) = r$, thus $r = 0$
  - Thus, $f(X) = q(X)(X - s)$ and $(X - s) \mid f(X)$
- **Lemma.** If $\deg(f) = N > n - 1$ and $f(X)$ vanishes on $\mathcal{S}$, then $f(X) = q(X)\mathbf{Z}_{\mathcal{S}}(X)$ for some $q(X) \in \mathbb{F}_{\leq N-n}[X]$
- $\mathbf{Z}_{\mathcal{S}}$ is **unique, minimal-degree, monic, non-zero** poly that vanishes exactly on $\mathcal{S}$
- **Important fact:** $\mathbf{Z}_{\mathbb{H}}(X) = \prod_i (X - \omega^{i-1}) = X^n - 1$ since $(\omega^{i-1})^n = \omega^{n(i-1)} = 1$

Polynomial View of Product Check

- $\forall i \in [1,n] \,.\, a_i b_i = c_i$

$\mathscr{R} = \{(\mathsf{x}, \mathsf{w}) : \mathsf{x} = [\![\hat{a}(X), \hat{b}(X), \hat{c}(X)]\!] \wedge \mathsf{w} = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = (\mathsf{FFT}(\hat{a}(X)), \mathsf{FFT}(\hat{b}(X)), \mathsf{FFT}(\hat{c}(X))) \wedge \forall i \in [1,n] a_i b_i = c_i\}$

Polynomial View of Product Check

- $\forall i \in [1,n] \,.\, a_i b_i = c_i$
  $\Leftrightarrow \forall i \in [1,n] \,.\, a_i b_i - c_i = 0$

$\mathcal{R} = \{(\mathsf{x}, \mathsf{w}) : \mathsf{x} = [\![\hat{a}(X), \hat{b}(X), \hat{c}(X)]\!] \wedge \mathsf{w} = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = (\mathsf{FFT}(\hat{a}(X)), \mathsf{FFT}(\hat{b}(X)), \mathsf{FFT}(\hat{c}(X))) \wedge \forall i \in [1,n] a_i b_i = c_i\}$

Interpolation/polynomial evaluation: fast algorithms to get from witness to polynomial encoding and back

Polynomial View of Product Check

- $\forall i \in [1,n] \,.\, a_i b_i = c_i$

  $\Leftrightarrow \forall i \in [1,n] \,.\, a_i b_i - c_i = 0$

  $\Leftrightarrow \forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1})\hat{b}(\omega^{i-1}) - \hat{c}(\omega^{i-1}) = 0$

$\mathcal{R} = \{(\mathsf{x}, \mathsf{w}) : \mathsf{x} = [\![\hat{a}(X), \hat{b}(X), \hat{c}(X)]\!] \wedge \mathsf{w} = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = (\mathsf{FFT}(\hat{a}(X)), \mathsf{FFT}(\hat{b}(X)), \mathsf{FFT}(\hat{c}(X))) \wedge \forall i \in [1,n] a_i b_i = c_i\}$

Polynomial View of Product Check

Interpolation/polynomial evaluation: fast algorithms to get from witness to polynomial encoding and back

- $\forall i \in [1,n] \,.\, a_i b_i = c_i$

  $\Leftrightarrow \forall i \in [1,n] \,.\, a_i b_i - c_i = 0$

  $\Leftrightarrow \forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1})\hat{b}(\omega^{i-1}) - \hat{c}(\omega^{i-1}) = 0$

  // $\hat{a}(X) := \sum_{i=1}^{n} a_i \ell_i(X) = \mathsf{IFFT}(\boldsymbol{a}) \in \mathbb{F}_{\leq n-1}[X]$, etc

$\mathscr{R} = \{(\mathrm{x}, \mathrm{w}) : \mathrm{x} = [\![\hat{a}(X), \hat{b}(X), \hat{c}(X)]\!] \wedge \mathrm{w} = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = (\mathsf{FFT}(\hat{a}(X)), \mathsf{FFT}(\hat{b}(X)), \mathsf{FFT}(\hat{c}(X))) \wedge \forall i \in [1,n] a_i b_i = c_i\}$

Polynomial View of Product Check

Interpolation/polynomial evaluation: fast algorithms to get from witness to polynomial encoding and back

- $\forall i \in [1,n] \,.\, a_i b_i = c_i$

  $\Leftrightarrow \forall i \in [1,n] \,.\, a_i b_i - c_i = 0$

  $\Leftrightarrow \forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1})\hat{b}(\omega^{i-1}) - \hat{c}(\omega^{i-1}) = 0$

  // $\hat{a}(X) := \sum_{i=1}^{n} a_i \ell_i(X) = \mathsf{IFFT}(\boldsymbol{a}) \in \mathbb{F}_{\leq n-1}[X]$, etc

  $\Leftrightarrow f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X) \in \mathbb{F}_{\leq 2n-2}[X]$ vanishes on $\mathbb{H}$

$\mathscr{R} = \{(\mathsf{x}, \mathsf{w}) : \mathsf{x} = [\![\hat{a}(X), \hat{b}(X), \hat{c}(X)]\!] \wedge \mathsf{w} = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = (\mathsf{FFT}(\hat{a}(X)), \mathsf{FFT}(\hat{b}(X)), \mathsf{FFT}(\hat{c}(X))) \wedge \forall i \in [1,n] a_i b_i = c_i\}$

Polynomial View of Product Check

Interpolation/polynomial evaluation: fast algorithms to get from witness to polynomial encoding and back

- $\forall i \in [1,n] \,.\, a_i b_i = c_i$

  $\Leftrightarrow \forall i \in [1,n] \,.\, a_i b_i - c_i = 0$

  $\Leftrightarrow \forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1})\hat{b}(\omega^{i-1}) - \hat{c}(\omega^{i-1}) = 0$

  // $\hat{a}(X) := \sum_{i=1}^{n} a_i \ell_i(X) = \mathsf{IFFT}(\boldsymbol{a}) \in \mathbb{F}_{\leq n-1}[X]$, etc

  $\Leftrightarrow f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X) \in \mathbb{F}_{\leq 2n-2}[X]$ vanishes on $\mathbb{H}$

  $\Leftrightarrow \mathbf{Z}_{\mathbb{H}}(X) \mid f(X)$

$\mathscr{R} = \{(\mathrm{x}, \mathrm{w}) : \mathrm{x} = [\![\hat{a}(X), \hat{b}(X), \hat{c}(X)]\!] \wedge \mathrm{w} = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = (\mathsf{FFT}(\hat{a}(X)), \mathsf{FFT}(\hat{b}(X)), \mathsf{FFT}(\hat{c}(X))) \wedge \forall i \in [1,n] a_i b_i = c_i\}$

Polynomial View of Product Check

Interpolation/polynomial evaluation: fast algorithms to get from witness to polynomial encoding and back

- $\forall i \in [1,n] \,.\, a_i b_i = c_i$

  $\Leftrightarrow \forall i \in [1,n] \,.\, a_i b_i - c_i = 0$

  $\Leftrightarrow \forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1})\hat{b}(\omega^{i-1}) - \hat{c}(\omega^{i-1}) = 0$

  // $\hat{a}(X) := \sum_{i=1}^{n} a_i \ell_i(X) = \mathsf{IFFT}(\boldsymbol{a}) \in \mathbb{F}_{\leq n-1}[X]$, etc

  $\Leftrightarrow f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X) \in \mathbb{F}_{\leq 2n-2}[X]$ vanishes on $\mathbb{H}$

  $\Leftrightarrow \mathbf{Z}_{\mathbb{H}}(X) \mid f(X)$

  $\Leftrightarrow \exists q(X) \in \mathbb{F}_{\leq n-2}[X] \,.\, f(X) = q(X)\mathbf{Z}_{\mathbb{H}}(X)$ (1)

$\mathscr{R} = \{(\mathsf{x}, \mathsf{w}) : \mathsf{x} = [\![\hat{a}(X), \hat{b}(X), \hat{c}(X)]\!] \wedge \mathsf{w} = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = (\mathsf{FFT}(\hat{a}(X)), \mathsf{FFT}(\hat{b}(X)), \mathsf{FFT}(\hat{c}(X))) \wedge \forall i \in [1,n] a_i b_i = c_i\}$

Polynomial View of Product Check

Interpolation/polynomial evaluation: fast algorithms to get from witness to polynomial encoding and back

- $\forall i \in [1,n] \,.\, a_i b_i = c_i$

  $\Leftrightarrow \forall i \in [1,n] \,.\, a_i b_i - c_i = 0$

  $\Leftrightarrow \forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1})\hat{b}(\omega^{i-1}) - \hat{c}(\omega^{i-1}) = 0$

  // $\hat{a}(X) := \sum_{i=1}^{n} a_i \ell_i(X) = \mathsf{IFFT}(\boldsymbol{a}) \in \mathbb{F}_{\leq n-1}[X]$, etc

  $\Leftrightarrow f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X) \in \mathbb{F}_{\leq 2n-2}[X]$ vanishes on $\mathbb{H}$

  $\Leftrightarrow \mathbf{Z}_{\mathbb{H}}(X) \mid f(X)$

  $\Leftrightarrow \exists q(X) \in \mathbb{F}_{\leq n-2}[X] \,.\, f(X) = q(X)\mathbf{Z}_{\mathbb{H}}(X)$ (1)

- Prover needs to prove that it “knows” $\hat{a}(X), \hat{b}(X), \hat{c}(X)$ and $q(X)$ satisfying (1)

$\mathscr{R} = \{(\mathsf{x}, \mathsf{w}) : \mathsf{x} = [\![\hat{a}(X), \hat{b}(X), \hat{c}(X)]\!] \wedge \mathsf{w} = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = (\mathsf{FFT}(\hat{a}(X)), \mathsf{FFT}(\hat{b}(X)), \mathsf{FFT}(\hat{c}(X))) \wedge \forall i \in [1,n] a_i b_i = c_i\}$

Polynomial View of Product Check

Interpolation/polynomial evaluation: fast algorithms to get from witness to polynomial encoding and back

- NB! This is a standard way of using univariate polynomials —need to internalise it!

- $\forall i \in [1,n] \,.\, a_i b_i = c_i$

  $\Leftrightarrow \forall i \in [1,n] \,.\, a_i b_i - c_i = 0$

  $\Leftrightarrow \forall i \in [1,n] \,.\, \hat{a}(\omega^{i-1})\hat{b}(\omega^{i-1}) - \hat{c}(\omega^{i-1}) = 0$

  // $\hat{a}(X) := \sum_{i=1}^{n} a_i \ell_i(X) = \mathsf{IFFT}(\boldsymbol{a}) \in \mathbb{F}_{\leq n-1}[X]$, etc

  $\Leftrightarrow f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X) \in \mathbb{F}_{\leq 2n-2}[X]$ vanishes on $\mathbb{H}$

  $\Leftrightarrow \mathbf{Z}_{\mathbb{H}}(X) \mid f(X)$

  $\Leftrightarrow \exists q(X) \in \mathbb{F}_{\leq n-2}[X] \,.\, f(X) = q(X)\mathbf{Z}_{\mathbb{H}}(X)$ (1)

- Prover needs to prove that it “knows” $\hat{a}(X), \hat{b}(X), \hat{c}(X)$ and $q(X)$ satisfying (1)

$\mathscr{R} = \{(\mathsf{x}, \mathsf{w}) : \mathsf{x} = [\![\hat{a}(X), \hat{b}(X), \hat{c}(X)]\!] \wedge \mathsf{w} = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = (\mathsf{FFT}(\hat{a}(X)), \mathsf{FFT}(\hat{b}(X)), \mathsf{FFT}(\hat{c}(X))) \wedge \forall i \in [1,n] a_i b_i = c_i\}$

Product Check PIOP

$\mathsf{x} = [\![\hat{a}(X), \bar{b}(X), \bar{c}(X)]\!], \mathsf{w} = \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^n$

$\hat{a}(X) \leftarrow \mathsf{Interp}(\boldsymbol{a})$
$\hat{b}(X) \leftarrow \mathsf{Interp}(\boldsymbol{b})$
$\hat{c}(X) \leftarrow \mathsf{Interp}(\boldsymbol{c})$

$\hat{a}(X), \hat{b}(X), \hat{c}(X)$

$\leq n - 1$

$f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X)$
Virtual oracle:
$g(X) := f(X) - q(X)Z_{\mathbb{H}}(X)$

$\mathsf{x} = [\![\hat{a}(X)]\!], \hat{b}(X), \hat{c}(X)]\!]$

Product Check PIOP

$f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X)$

Virtual oracle:

$g(X) := f(X) - q(X)\mathsf{Z}_{\mathbb{H}}(X)$

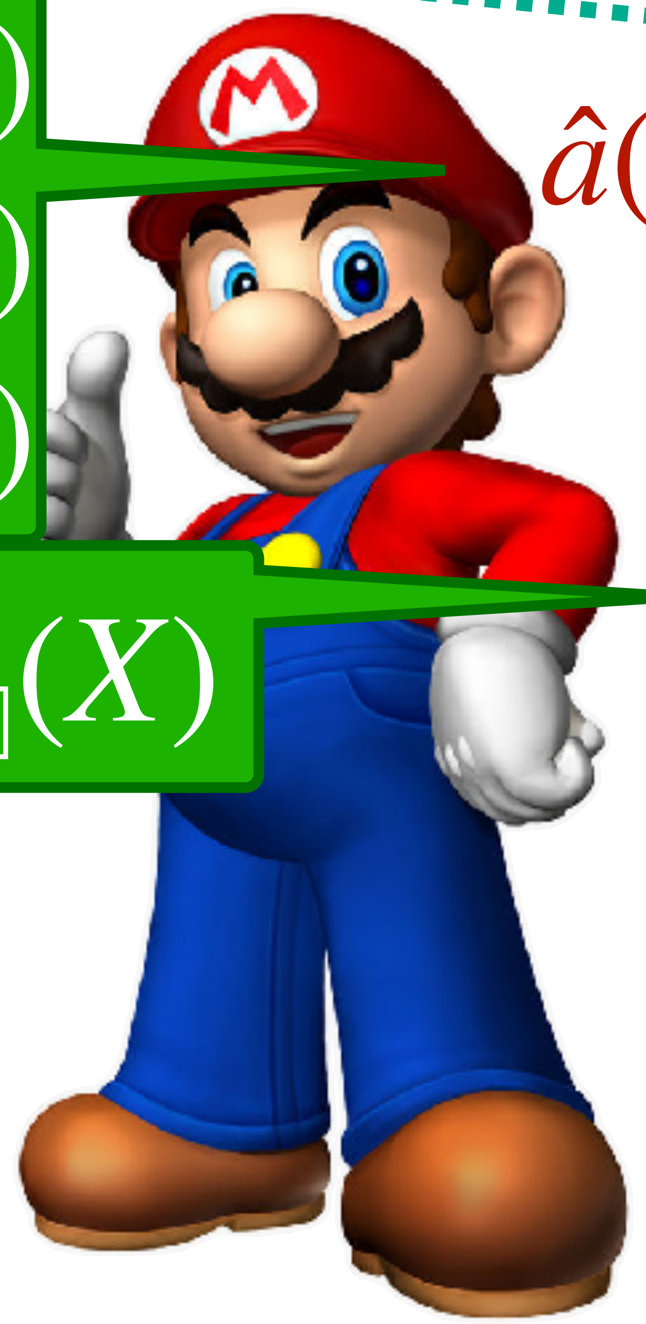
$\mathsf{x} = [\![\hat{a}(X), \bar{b}(X), \bar{c}(X)]\!], \mathsf{w} = \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^n$

$\hat{a}(X) \leftarrow \mathsf{Interp}(\boldsymbol{a})$

$\hat{b}(X) \leftarrow \mathsf{Interp}(\boldsymbol{b})$

$\hat{c}(X) \leftarrow \mathsf{Interp}(\boldsymbol{c})$

$q(X) \leftarrow f(X)/\mathsf{Z}_{\mathbb{H}}(X)$



$\hat{a}(X), \hat{b}(X), \hat{c}(X)$

$q(X)$

$\leq n-1$

$\mathsf{x} = [\![\hat{a}(X)]\!], \hat{b}(X), \hat{c}(X)]\!]$

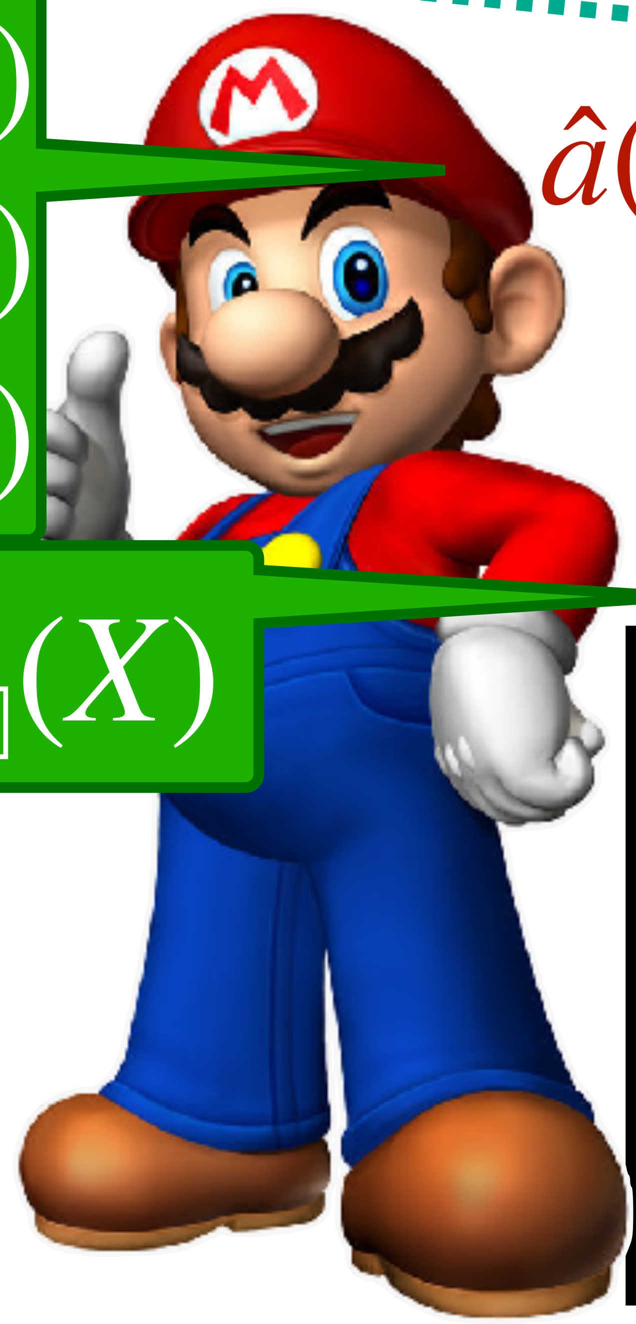
Product Check PIOP

$f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X)$

Virtual oracle:

$g(X) := f(X) - q(X)Z_{\mathbb{H}}(X)$

$\mathbb{x} = [\![\hat{a}(X), \bar{b}(X), \bar{c}(X)]\!], \mathbb{w} = \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^n$



$\hat{a}(X) \leftarrow \text{Interp}(\boldsymbol{a})$

$\hat{b}(X) \leftarrow \text{Interp}(\boldsymbol{b})$

$\hat{c}(X) \leftarrow \text{Interp}(\boldsymbol{c})$

$q(X) \leftarrow f(X)/Z_{\mathbb{H}}(X)$

$\hat{a}(X), \hat{b}(X), \hat{c}(X)$

$q(X)$

$\leq n-1$

$\mathbb{x} = [\![\hat{a}(X)]\!], \hat{b}(X), \hat{c}(X)]\!]$

Zero Check for virtual oracle $[\![g(X)]\!]$

$r \leftarrow_{\$} \mathbb{F}$

$\bar{a} \leftarrow \hat{a}(r), \bar{b} \leftarrow \hat{b}(r)$

$\bar{c} \leftarrow \hat{c}(r), \bar{q} \leftarrow q(r)$

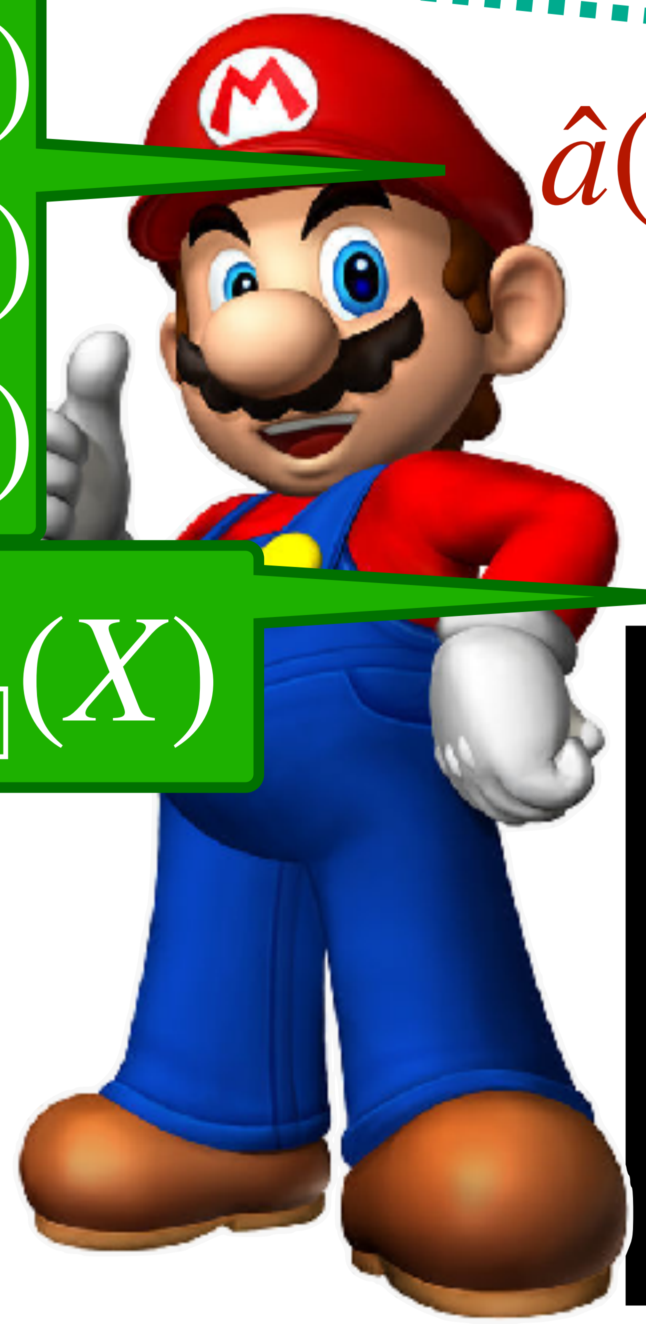
Product Check PIOP

$f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X)$
Virtual oracle:
$g(X) := f(X) - q(X)\mathbb{Z}_{\mathbb{H}}(X)$

$\mathbb{x} = [\![\hat{a}(X), \bar{b}(X), \bar{c}(X)]\!], \mathbb{w} = \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^n$

$\hat{a}(X) \leftarrow \text{Interp}(\boldsymbol{a})$
$\hat{b}(X) \leftarrow \text{Interp}(\boldsymbol{b})$
$\hat{c}(X) \leftarrow \text{Interp}(\boldsymbol{c})$

$q(X) \leftarrow f(X)/\mathbb{Z}_{\mathbb{H}}(X)$



$\hat{a}(X), \hat{b}(X), \hat{c}(X)$

$q(X)$

$\leq n - 1$

$\mathbb{x} = [\![\hat{a}(X)]\!], \hat{b}(X), \hat{c}(X)]\!]$

Zero Check for virtual oracle $[\![g(X)]\!]$

$r \leftarrow_\$ \mathbb{F}$

$\bar{a} \leftarrow \hat{a}(r), \bar{b} \leftarrow \hat{b}(r)$
$\bar{c} \leftarrow \hat{c}(r), \bar{q} \leftarrow q(r)$

Check $g(r) =^? 0$:
- $\mathbb{Z}_{\mathbb{H}}(r) \leftarrow r^n - 1$
- Check $\bar{a}\bar{b} - \bar{c} =^? \bar{q}\mathbb{Z}_{\mathbb{H}}(r)$

Product Check PIOP

$f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X)$

Virtual oracle:

$g(X) := f(X) - q(X)Z_{\mathbb{H}}(X)$

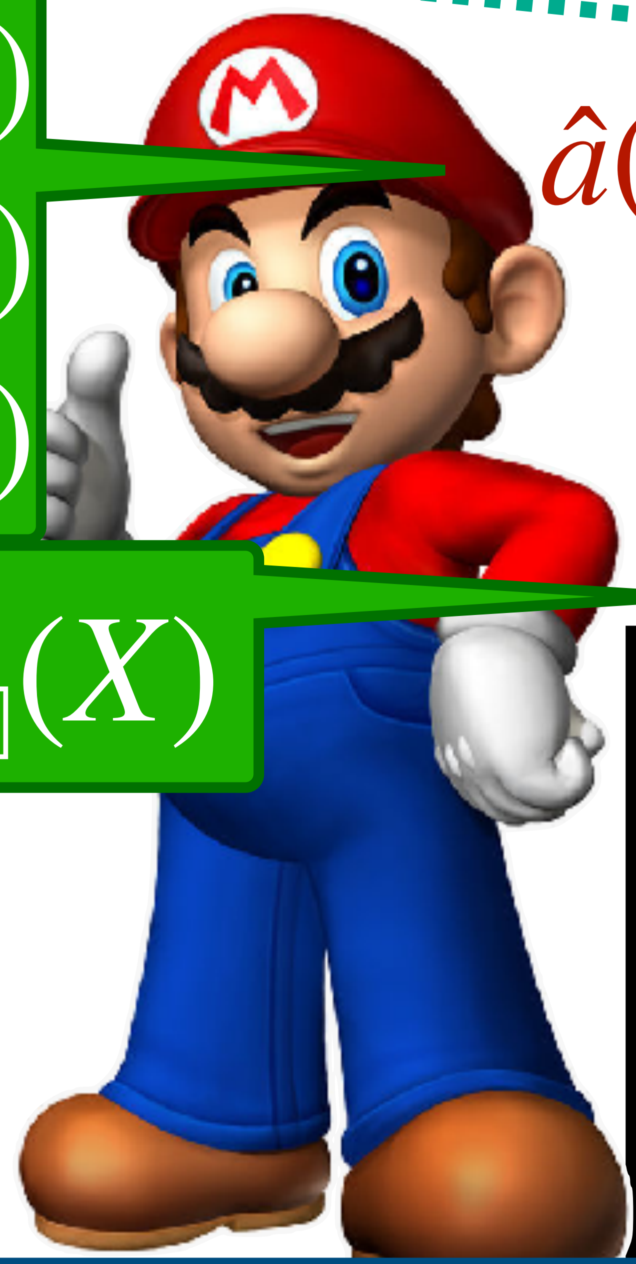
$\mathrm{x} = [\![\hat{a}(X), \bar{b}(X), \bar{c}(X)]\!], \mathrm{w} = \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^n$

$\hat{a}(X) \leftarrow \mathrm{Interp}(\boldsymbol{a})$

$\hat{b}(X) \leftarrow \mathrm{Interp}(\boldsymbol{b})$

$\hat{c}(X) \leftarrow \mathrm{Interp}(\boldsymbol{c})$

$q(X) \leftarrow f(X)/Z_{\mathbb{H}}(X)$



$\hat{a}(X), \hat{b}(X), \hat{c}(X)$

$q(X)$

$\leq n - 1$

$\mathrm{x} = [\![\hat{a}(X)], \hat{b}(X), \hat{c}(X)]\!]$

Zero Check for virtual oracle $[\![g(X)]\!]$

$r \leftarrow_{\$} \mathbb{F}$

$\bar{a} \leftarrow \hat{a}(r), \bar{b} \leftarrow \hat{b}(r)$

$\bar{c} \leftarrow \hat{c}(r), \bar{q} \leftarrow q(r)$

Check $g(r) =^? 0$:

- $Z_{\mathbb{H}}(r) \leftarrow r^n - 1$
- Check $\bar{a}\bar{b} - \bar{c} =^? \bar{q}Z_{\mathbb{H}}(r)$

- $[\![g(X)]\!]$ is a <u>virtual</u> oracle
  - Not kept in an oracle but computed from other oracles
- V “virtually" queries $[\![g(X)]\!]$:
  - query $\bar{a} \leftarrow \hat{a}(r), \bar{b} \leftarrow \hat{b}(r), \bar{c} \leftarrow \hat{c}(r), \bar{q} \leftarrow q(r)$
  - set $g(r) \leftarrow \bar{a}\bar{b} - \bar{c} - \bar{q}Z_{\mathbb{H}}(r)$
  - check $g(r) =^? 0$

Efficiency

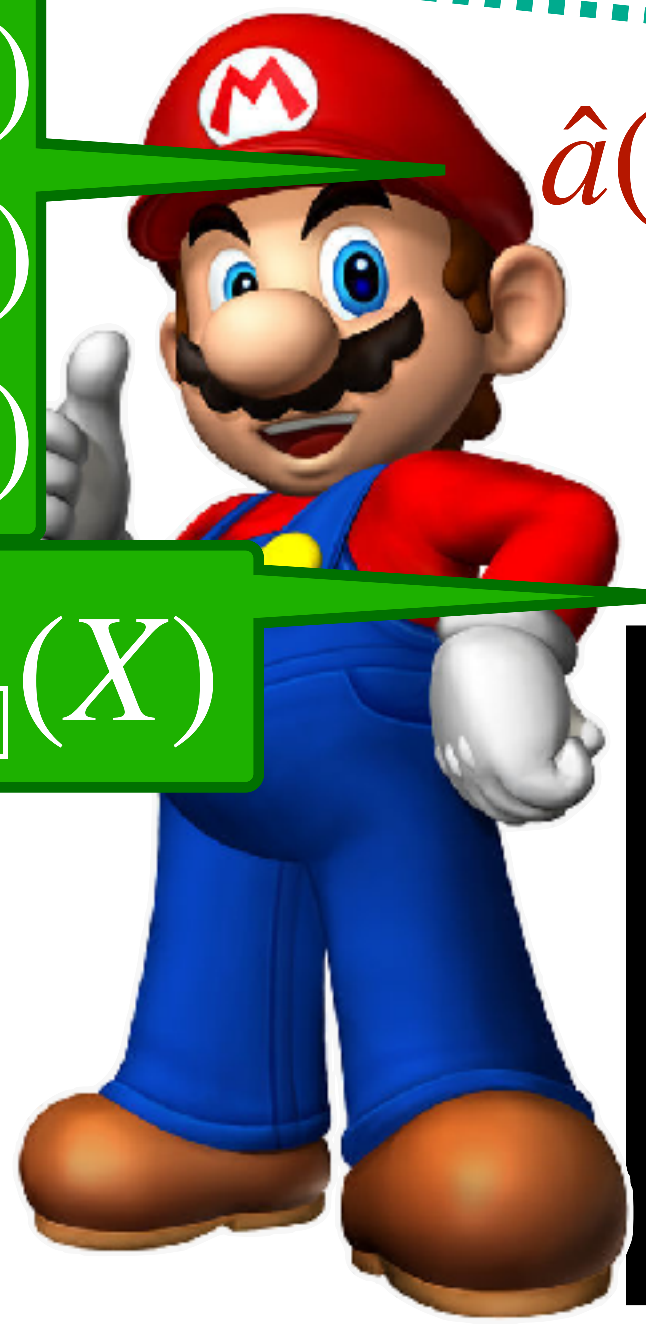
Product Check: Efficiency

$f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X)$
Virtual oracle:
$g(X) := f(X) - q(X)Z_{\mathbb{H}}(X)$

$\mathsf{x} = [\![\hat{a}(X), \bar{b}(X), \bar{c}(X)]\!], \mathsf{w} = \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^n$

$\hat{a}(X) \leftarrow \text{Interp}(\boldsymbol{a})$
$\hat{b}(X) \leftarrow \text{Interp}(\boldsymbol{b})$
$\hat{c}(X) \leftarrow \text{Interp}(\boldsymbol{c})$

$q(X) \leftarrow f(X)/Z_{\mathbb{H}}(X)$



$\hat{a}(X), \hat{b}(X), \hat{c}(X)$

$\leq n - 1$

$q(X)$

$\mathsf{x} = [\![\hat{a}(X)]\!], \hat{b}(X), \hat{c}(X)]\!]$

Zero Check for virtual oracle $[\![g(X)]\!]$

$r \leftarrow_{\$} \mathbb{F}$

$\bar{a} \leftarrow \hat{a}(r), \bar{b} \leftarrow \hat{b}(r)$

$\bar{c} \leftarrow \hat{c}(r), \bar{q} \leftarrow$

Check $g(r) \stackrel{?}{=} 0$:
- $Z_{\mathbb{H}}(r) \leftarrow r^n - 1$
- Check $\bar{a}\bar{b} - \bar{c} \stackrel{?}{=} \bar{q}Z_{\mathbb{H}}(r)$

- One polynomial multiplication: $f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X)$
  - $O(n \log n)$ field ops // includes FFT & inverse FFT
- 3 interpolations: $\boldsymbol{a} \mapsto \hat{a}(X)$, …
  - Uses **inverse FFT**
- Total computation: $O(n \log n)$ field ops

Product Check: Efficiency

$f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X)$

Virtual oracle:

$g(X) := f(X) - q(X)Z_{\mathbb{H}}(X)$

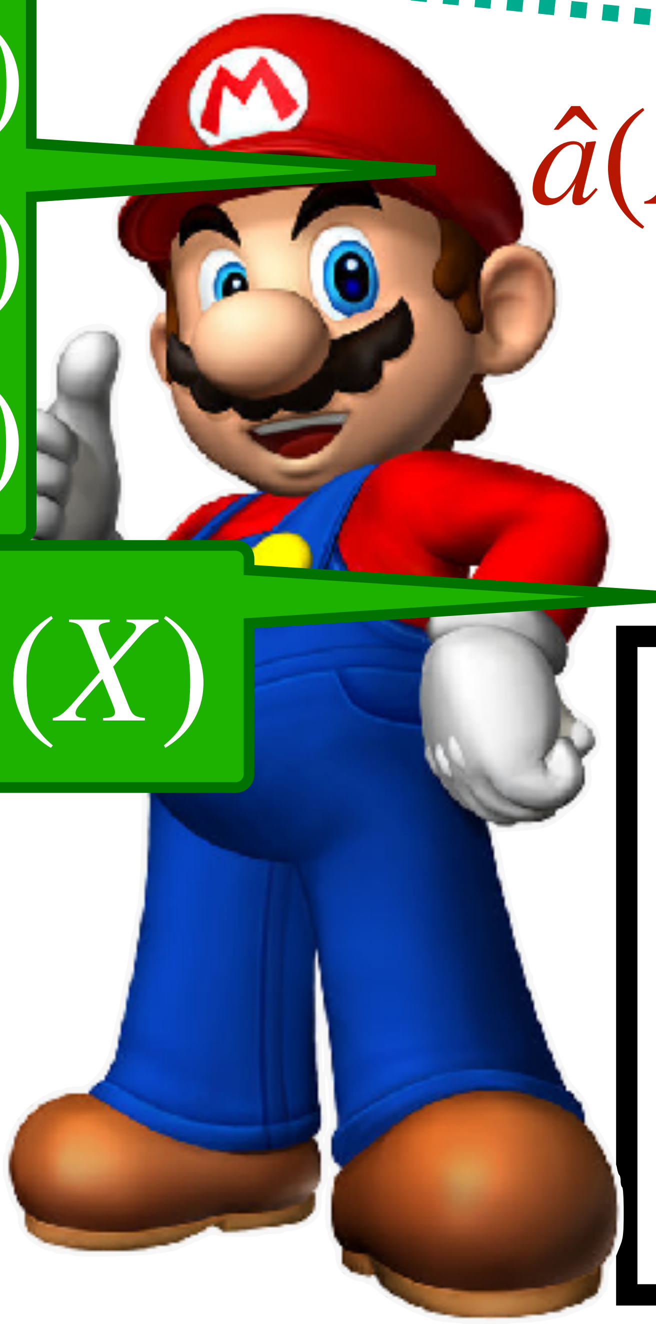
$\mathsf{x} = [\![\hat{a}(X), \bar{b}(X), \bar{c}(X)]\!], \mathsf{w} = \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^n$

$\hat{a}(X) \leftarrow \text{Interp}(\boldsymbol{a})$

$\hat{b}(X) \leftarrow \text{Interp}(\boldsymbol{b})$

$\hat{c}(X) \leftarrow \text{Interp}(\boldsymbol{c})$

$q(X) \leftarrow f(X)/Z_{\mathbb{H}}(X)$



$\hat{a}(X), \hat{b}(X), \hat{c}(X)$

$\leq n-1$

$\mathsf{x} = [\![\hat{a}(X)], \hat{b}(X), \hat{c}(X)]\!]$

$q(X)$

Zero Check for virtual oracle $[\![g(X)]\!]$

$r \leftarrow_{\$} \mathbb{F}$

$\bar{a} \leftarrow \hat{a}(r), \bar{b} \leftarrow \hat{b}(r)$

$\bar{c} \leftarrow \hat{c}(r), \bar{q}$

Check $g(r) =^? 0$:

- $Z_{\mathbb{H}}(r) \leftarrow r^n - 1$
- Check $\bar{a}\bar{b} - \bar{c} =^? \bar{q}Z_{\mathbb{H}}(r)$

- One polynomial multiplication: $f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X)$
  - $O(n \log n)$ field ops // includes FFT & inverse FFT
- 3 interpolations: $\boldsymbol{a} \mapsto \hat{a}(X), \ldots$
  - Uses **inverse FFT**
- Total computation: $O(n \log n)$ field ops

- Computing $Z_{\mathbb{H}}(r)$ // $O(\log n)$ f.o.
- +2 multiplications
- Total time $O(\log n)$ field ops

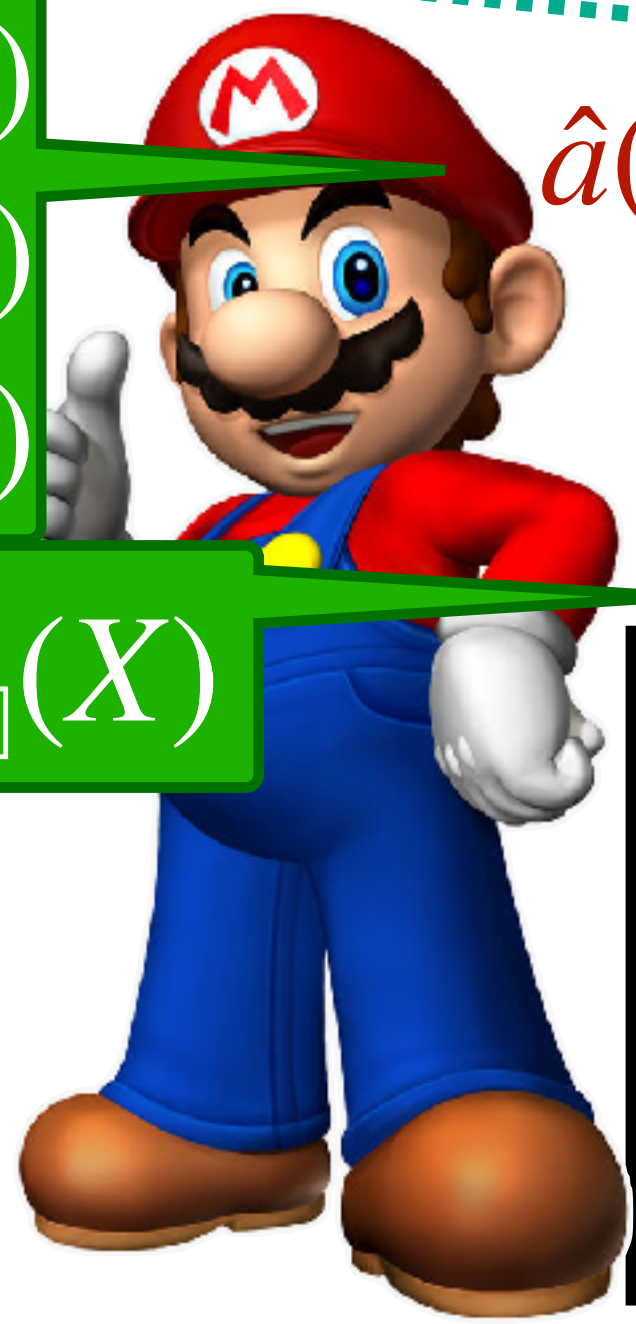
Product Check: Efficiency

$f(X) := \hat{a}(X)\hat{b}(X) - \hat{c}(X)$
Virtual oracle:
$g(X) := f(X) - q(X)Z_{\mathbb{H}}(X)$

$\mathbb{x} = [\![\hat{a}(X), \bar{b}(X), \bar{c}(X)]\!], \mathbb{w} = \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^n$

$\hat{a}(X) \leftarrow \text{Interp}(\boldsymbol{a})$
$\hat{b}(X) \leftarrow \text{Interp}(\boldsymbol{b})$
$\hat{c}(X) \leftarrow \text{Interp}(\boldsymbol{c})$

$q(X) \leftarrow f(X)/Z_{\mathbb{H}}(X)$



$\hat{a}(X), \hat{b}(X), \hat{c}(X)$

$\leq n - 1$

$q(X)$

$\mathbb{x} = [\![\hat{a}(X)]\!], \hat{b}(X), \hat{c}(X)]\!]$

Zero Check for virtual oracle $[\![g(X)]\!]$

$r \leftarrow_{\$} \mathbb{F}$

$\bar{a} \leftarrow \hat{a}(r), \bar{b} \leftarrow \hat{b}(r)$
$\bar{c} \leftarrow \hat{c}(r), \bar{q} \leftarrow q(r)$

Check $g(r) \stackrel{?}{=} 0$:
- $Z_{\mathbb{H}}(r) \leftarrow r^n - 1$
- Check $\bar{a}\bar{b} - \bar{c} \stackrel{?}{=} \bar{q}Z_{\mathbb{H}}(r)$

FFT And ZK Applications

- In ZK applications, FFT is used to **interpolate** input vectors but also to **multiply** polynomials

FFT And ZK Applications

- In ZK applications, FFT is used to **interpolate** input vectors but also to **multiply** polynomials
- Crucial to understand FFT since it is often prover's **dominant** cost of PIOPs

FFT And ZK Applications

- In ZK applications, FFT is used to **interpolate** input vectors but also to **multiply** polynomials
- Crucial to understand FFT since it is often prover's **dominant** cost of PIOPs
  - Also, **why** FFT is often the bottleneck

FFT And ZK Applications

- In ZK applications, FFT is used to **interpolate** input vectors but also to **multiply** polynomials
- Crucial to understand FFT since it is often prover's **dominant** cost of PIOPs
  - Also, **why** FFT is often the bottleneck
- Most applications of FFT run with relatively small inputs

FFT And ZK Applications

- In ZK applications, FFT is used to **interpolate** input vectors but also to **multiply** polynomials
- Crucial to understand FFT since it is often prover's **dominant** cost of PIOPs
  - Also, **why** FFT is often the bottleneck
- Most applications of FFT run with relatively small inputs
- In ZK application case, ideally, $n \approx 2^{32}$

FFT And ZK Applications

- In ZK applications, FFT is used to **interpolate** input vectors but also to **multiply** polynomials
- Crucial to understand FFT since it is often prover's **dominant** cost of PIOPs
  - Also, **why** FFT is often the bottleneck
- Most applications of FFT run with relatively small inputs
- In ZK application case, ideally, $n \approx 2^{32}$
- $n = 2^{32}$ => IFFT is dominated by $\frac{n}{2} \log_2 n = 16n = 2^{36}$ field ops

FFT And ZK Applications

- In ZK applications, FFT is used to **interpolate** input vectors but also to **multiply** polynomials
- Crucial to understand FFT since it is often prover's **dominant** cost of PIOPs
  - Also, **why** FFT is often the bottleneck
- Most applications of FFT run with relatively small inputs
- In ZK application case, ideally, $n \approx 2^{32}$
- $n = 2^{32}$ => IFFT is dominated by $\frac{n}{2} \log_2 n = 16n = 2^{36}$ field ops
  - (5*)16x slowdown is not so bad? // At least n ops is needed by prover!

FFT: Memory Problems

FFT: Memory Problems

- (5*)16x slowdown is not so bad? // At least n ops is needed by prover!

FFT: Memory Problems

- (5*)16x slowdown is not so bad? // At least n ops is needed by prover!
  - But it's in **field ops** (and we have a 256-bit field)

FFT: Memory Problems

- (5*)16x slowdown is not so bad? // At least n ops is needed by prover!
  - But it's in **field ops** (and we have a 256-bit field)
- **Memory consumption:** at least $n = 2^{32}$ field elements

FFT: Memory Problems

- (5*)16x slowdown is not so bad? // At least n ops is needed by prover!
  - But it's in **field ops** (and we have a 256-bit field)
- **Memory consumption:** at least $n = 2^{32}$ field elements
  - big f.e.: 256 bits, so $n = 2^{32} \cdot (256/8) = 2^{37}$ bytes = 128 GigaBytes

FFT: Memory Problems

- (5*)16x slowdown is not so bad? // At least n ops is needed by prover!
  - But it's in **field ops** (and we have a 256-bit field)
- **Memory consumption:** <u>at least</u> $n = 2^{32}$ field elements
  - big f.e.: 256 bits, so $n = 2^{32} \cdot (256/8) = 2^{37}$ bytes = 128 GigaBytes
  - small f.e.: 32 bits, so $n = 2^{32} \cdot (32/8) = 2^{34}$ bytes = 16 GigaBytes

FFT: Memory Problems

- (5*)16x slowdown is not so bad? // At least n ops is needed by prover!
  - But it's in **field ops** (and we have a 256-bit field)
- **Memory consumption:** <u>at least</u> $n = 2^{32}$ field elements
  - big f.e.: 256 bits, so $n = 2^{32} \cdot (256/8) = 2^{37}$ bytes = 128 GigaBytes
  - small f.e.: 32 bits, so $n = 2^{32} \cdot (32/8) = 2^{34}$ bytes = 16 GigaBytes
  - Standard FFT requires more than n field elements of memory

FFT: Memory Problems

- (5*)16x slowdown is not so bad? // At least n ops is needed by prover!
  - But it’s in **field ops** (and we have a 256-bit field)
- **Memory consumption:** <u>at least</u> $n = 2^{32}$ field elements
  - big f.e.: 256 bits, so $n = 2^{32} \cdot (256/8) = 2^{37}$ bytes = 128 GigaBytes
  - small f.e.: 32 bits, so $n = 2^{32} \cdot (32/8) = 2^{34}$ bytes = 16 GigaBytes
  - Standard FFT requires more than n field elements of memory
  - And in-place FFT algorithms require more time

FFT: Memory Problems

- (5*)16x slowdown is not so bad? // At least n ops is needed by prover!
  - But it's in **field ops** (and we have a 256-bit field)
- **Memory consumption:** <u>at least</u> $n = 2^{32}$ field elements
  - big f.e.: 256 bits, so $n = 2^{32} \cdot (256/8) = 2^{37}$ bytes = 128 GigaBytes
  - small f.e.: 32 bits, so $n = 2^{32} \cdot (32/8) = 2^{34}$ bytes = 16 GigaBytes
  - Standard FFT requires more than n field elements of memory
  - And in-place FFT algorithms require more time
- **Memory locality:** butterfly-like memory access causes cache misses

Why *Polynomial* IOPs?

- Compared to “non-P“IOPs, polynomial IOPs offer two related benefits

Why *Polynomial* IOPs?

- Compared to “non-P“IOPs, polynomial IOPs offer two related benefits

1. **Schwartz-Zippel** makes it possible to implement zero check efficiently

Why *Polynomial* IOPs?

- Compared to “non-P“IOPs, polynomial IOPs offer two related benefits

1. **Schwartz-Zippel** makes it possible to implement zero check efficiently
    - In general, Schwartz-Zippel is the main reason one can get succinctness

Why *Polynomial* IOPs?

- Compared to “non-P“IOPs, polynomial IOPs offer two related benefits

1. **Schwartz-Zippel** makes it possible to implement zero check efficiently
    - In general, Schwartz-Zippel is the main reason one can get succinctness
    - Instead of sending “long" polynomials, can send “succinct” evaluations

Why *Polynomial* IOPs?

- Compared to “non-P“IOPs, polynomial IOPs offer two related benefits

1. **Schwartz-Zippel** makes it possible to implement zero check efficiently
    - In general, Schwartz-Zippel is the main reason one can get succinctness
    - Instead of sending “long" polynomials, can send “succinct” evaluations
2. **“Out of bounds” evaluation:**

Why *Polynomial* IOPs?

- Compared to “non-P“IOPs, polynomial IOPs offer two related benefits

1. **Schwartz-Zippel** makes it possible to implement zero check efficiently
    - In general, Schwartz-Zippel is the main reason one can get succinctness
    - Instead of sending “long" polynomials, can send “succinct” evaluations
2. **“Out of bounds” evaluation:**
    - While the length of the input vector is N, the polynomial can be evaluated on $|\mathbb{F}| \gg N$ points

Why *Polynomial* IOPs?

- Compared to “non-P“IOPs, polynomial IOPs offer two related benefits

1. **Schwartz-Zippel** makes it possible to implement zero check efficiently
    - In general, Schwartz-Zippel is the main reason one can get succinctness
    - Instead of sending “long" polynomials, can send “succinct” evaluations
2. **“Out of bounds” evaluation:**
    - While the length of the input vector is N, the polynomial can be evaluated on $|\mathbb{F}| \gg N$ points
    - This gives one huge freedom in designing more efficient protocols

Why *Polynomial* IOPs?

- Compared to “non-P“IOPs, polynomial IOPs offer two related benefits

1. **Schwartz-Zippel** makes it possible to implement zero check efficiently
   - In general, Schwartz-Zippel is the main reason one can get succinctness
   - Instead of sending “long" polynomials, can send “succinct” evaluations
2. **“Out of bounds” evaluation:**
   - While the length of the input vector is N, the polynomial can be evaluated on $|\mathbb{F}| \gg N$ points
   - This gives one huge freedom in designing more efficient protocols

- We will see later that the best “non-P“IOP for Zero Check is far from efficient

Why *Polynomial* IOPs?

- Compared to “non-P“IOPs, polynomial IOPs offer two related benefits

1. **Schwartz-Zippel** makes it possible to implement zero check efficiently
   - In general, Schwartz-Zippel is the main reason one can get succinctness
   - Instead of sending “long" polynomials, can send “succinct” evaluations
2. **“Out of bounds” evaluation:**
   - While the length of the input vector is N, the polynomial can be evaluated on $|\mathbb{F}| \gg N$ points
   - This gives one huge freedom in designing more efficient protocols

- We will see later that the best “non-P“IOP for Zero Check is far from efficient
- **Trade-off:** in “non-P“IOPs, one can instantiate crypto more efficiently

What Did We Miss?

What Did We Miss?

- In two more seminars, we could describe how to construct a PIOP to very an arbitrary arithmetic circuit with given complexity + prove security

What Did We Miss?

- In two more seminars, we could describe how to construct a PIOP to very an arbitrary arithmetic circuit with given complexity + prove security
  - **Prover:** a few FFTs + polynomial multiplications of size $n \geq 2^{24}$

What Did We Miss?

- In two more seminars, we could describe how to construct a PIOP to very an arbitrary arithmetic circuit with given complexity + prove security
  - **Prover:** a few FFTs + polynomial multiplications of size $n \geq 2^{24}$
  - **Verifier:** constant number of field operations

What Did We Miss?

- In two more seminars, we could describe how to construct a PIOP to very an arbitrary arithmetic circuit with given complexity + prove security
  - **Prover:** a few FFTs + polynomial multiplications of size $n \geq 2^{24}$
  - **Verifier:** constant number of field operations
- One more seminar: implementing the oracle by using **KZG**, an elliptic-curve based polynomial commitment scheme

What Did We Miss?

- In two more seminars, we could describe how to construct a PIOP to very an arbitrary arithmetic circuit with given complexity + prove security
  - **Prover:** a few FFTs + polynomial multiplications of size $n \geq 2^{24}$
  - **Verifier:** constant number of field operations
- One more seminar: implementing the oracle by using **KZG**, an elliptic-curve based polynomial commitment scheme
  - Adds to costs

What Did We Miss?

- In two more seminars, we could describe how to construct a PIOP to very an arbitrary arithmetic circuit with given complexity + prove security
  - **Prover:** a few FFTs + polynomial multiplications of size $n \geq 2^{24}$
  - **Verifier:** constant number of field operations
- One more seminar: implementing the oracle by using **KZG**, an elliptic-curve based polynomial commitment scheme
  - Adds to costs
  - **Prover:** $O(n)$ e.c. group operations, each group op > 256 field operations

What Did We Miss?

- In two more seminars, we could describe how to construct a PIOP to very an arbitrary arithmetic circuit with given complexity + prove security
  - **Prover:** a few FFTs + polynomial multiplications of size $n \geq 2^{24}$
  - **Verifier:** constant number of field operations
- One more seminar: implementing the oracle by using **KZG**, an elliptic-curve based polynomial commitment scheme
  - Adds to costs
  - **Prover:** $O(n)$ e.c. group operations, each group op > 256 field operations
  - **Verifier:** constant number of group operations

What Did We Miss?

- In two more seminars, we could describe how to construct a PIOP to very an arbitrary arithmetic circuit with given complexity + prove security
  - **Prover:** a few FFTs + polynomial multiplications of size $n \geq 2^{24}$
  - **Verifier:** constant number of field operations
- One more seminar: implementing the oracle by using **KZG**, an elliptic-curve based polynomial commitment scheme
  - Adds to costs
  - **Prover:** $O(n)$ e.c. group operations, each group op > 256 field operations
  - **Verifier:** constant number of group operations
  - Crypto part is costly!

What Did We Miss?

- In two more seminars, we could describe how to construct a PIOP to very an arbitrary arithmetic circuit with given complexity + prove security
  - **Prover:** a few FFTs + polynomial multiplications of size $n \geq 2^{24}$
  - **Verifier:** constant number of field operations
- One more seminar: implementing the oracle by using **KZG**, an elliptic-curve based polynomial commitment scheme
  - Adds to costs
  - **Prover:** $O(n)$ e.c. group operations, each group op > 256 field operations
  - **Verifier:** constant number of group operations
  - Crypto part is costly!
- One more seminar: Fiat-Shamir (how to make it non-interactive)

Way Forward: Efficiency

A very short list

- Described techniques + extra seminars:

Way Forward: Efficiency

A very short list

- Described techniques + extra seminars:
  - An ideal-for-verifier solution, but slow for the prover

Way Forward: Efficiency

A very short list

- Described techniques + extra seminars:
  - An ideal-for-verifier solution, but slow for the prover
  - Crypto is slow

Way Forward: Efficiency

A very short list

- Described techniques + extra seminars:
  - An ideal-for-verifier solution, but slow for the prover
  - Crypto is slow
  - FFT is slow

Way Forward: Efficiency

A very short list

- Described techniques + extra seminars:
  - An ideal-for-verifier solution, but slow for the prover
  - Crypto is slow
  - FFT is slow
  - Converting arbitrary computation to finite field ops and circuits is slow

Way Forward: Efficiency

A very short list

- Described techniques + extra seminars:
  - An ideal-for-verifier solution, but slow for the prover
  - Crypto is slow
  - FFT is slow
  - Converting arbitrary computation to finite field ops and circuits is slow
- Univariate polynomials => multilinear polynomials: no need to interpolate

Way Forward: Efficiency

A very short list

- Described techniques + extra seminars:
  - An ideal-for-verifier solution, but slow for the prover
  - Crypto is slow
  - FFT is slow
  - Converting arbitrary computation to finite field ops and circuits is slow
- Univariate polynomials => multilinear polynomials: no need to interpolate
- GKR protocol => need to cryptographically commit to less values

Way Forward: Efficiency

A very short list

- Described techniques + extra seminars:
  - An ideal-for-verifier solution, but slow for the prover
  - Crypto is slow
  - FFT is slow
  - Converting arbitrary computation to finite field ops and circuits is slow
- Univariate polynomials => multilinear polynomials: no need to interpolate
- GKR protocol => need to cryptographically commit to less values
- Lookups => store valid gate I/Os in a table, prove all gate I/Os are in that table

Way Forward: Efficiency

A very short list

- Described techniques + extra seminars:
  - An ideal-for-verifier solution, but slow for the prover
  - Crypto is slow
  - FFT is slow
  - Converting arbitrary computation to finite field ops and circuits is slow
- Univariate polynomials => multilinear polynomials: no need to interpolate
- GKR protocol => need to cryptographically commit to less values
- Lookups => store valid gate I/Os in a table, prove all gate I/Os are in that table
- Folding => fold inputs and witnesses together before doing an operation

Way Forward: Efficiency

A very short list

- Described techniques + extra seminars:
  - An ideal-for-verifier solution, but slow for the prover
  - Crypto is slow
  - FFT is slow
  - Converting arbitrary computation to finite field ops and circuits is slow
- Univariate polynomials => multilinear polynomials: no need to interpolate
- GKR protocol => need to cryptographically commit to less values
- Lookups => store valid gate I/Os in a table, prove all gate I/Os are in that table
- Folding => fold inputs and witnesses together before doing an operation
- Code&hash-based => using any fields, hash is fast, post-quantum

Way Forward: Security

A very short list

- More stringent security notions

Way Forward: Security

A very short list

- More stringent security notions
  - Security in environments, where adversary sees arbitrary communication?

Way Forward: Security

A very short list

- More stringent security notions
  - Security in environments, where adversary sees arbitrary communication?
- Weaker cryptographic assumptions

Way Forward: Security

A very short list

- More stringent security notions
  - Security in environments, where adversary sees arbitrary communication?
- Weaker cryptographic assumptions
  - Weaker elliptic-curve assumptions?

Way Forward: Security

A very short list

- More stringent security notions
  - Security in environments, where adversary sees arbitrary communication?
- Weaker cryptographic assumptions
  - Weaker elliptic-curve assumptions?
  - Post-quantum?

Way Forward: Security

A very short list

- More stringent security notions
  - Security in environments, where adversary sees arbitrary communication?
- Weaker cryptographic assumptions
  - Weaker elliptic-curve assumptions?
  - Post-quantum?
- It was just found in 2025 that even Fiat-Shamir is not secure in the case of actually used protocols

Way Forward: Security

A very short list

- More stringent security notions
  - Security in environments, where adversary sees arbitrary communication?
- Weaker cryptographic assumptions
  - Weaker elliptic-curve assumptions?
  - Post-quantum?
- It was just found in 2025 that even Fiat-Shamir is not secure in the case of actually used protocols
- Formal verification and automated security proofs

Way Forward: Applications

A very short list

- Make better ZK for diverse applications

Way Forward: Applications

A very short list

- Make better ZK for diverse applications
- Big right now:

Way Forward: Applications

A very short list

- Make better ZK for diverse applications
- Big right now:
  - L2 blockchain, zkRollup

Way Forward: Applications

A very short list

- Make better ZK for diverse applications
- Big right now:
  - L2 blockchain, zkRollup
  - zkVM

Way Forward: Applications

A very short list

- Make better ZK for diverse applications
- Big right now:
  - L2 blockchain, zkRollup
  - zkVM
  - zkML

Questions?

Important References

- (FFT) James W. Cooley, John W. Tukey: **An algorithm for the machine calculation of complex Fourier series** (1965)
    - Classic algorithm, many brilliant presentations, including on YouTube
- (Good book on polynomial algorithms) Joachim von zur Gathen, Jürgen Gerhard: **Modern Computer Algebra (3. ed.).** Cambridge University Press 2013
- PIOP:
    - Benedikt Bünz, Ben Fisch, Alan Szepieniec. **Transparent SNARKs from DARK compilers** (2020)
    - Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, Nicholas Ward. **Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS** (2020)