HPG 2022
HIGH-PERFORMANCE GRAPHICS

High-Performance Polynomial Root Finding for Graphics

Cem Yuksel, *University of Utah*

Polynomials

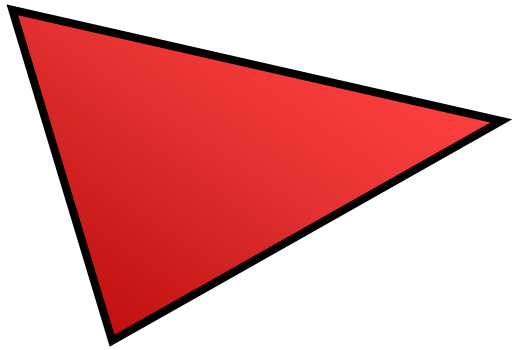
- Degree 2 (quadratic)

$$ax^2 + bx + c$$

Polynomials

- Degree 2 (quadratic)
- Degree 3 (cubic)

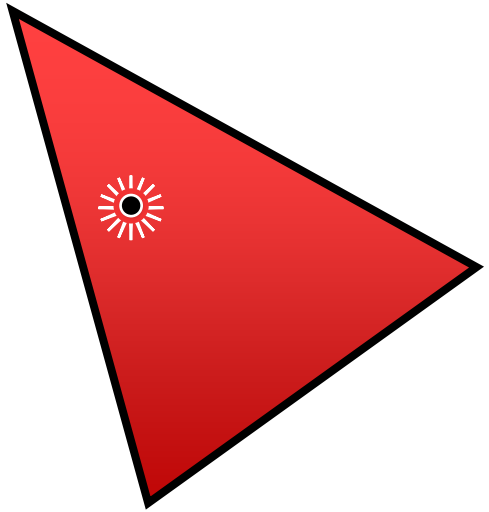
$$ax^3 + bx^2 + cx + d$$



Polynomials

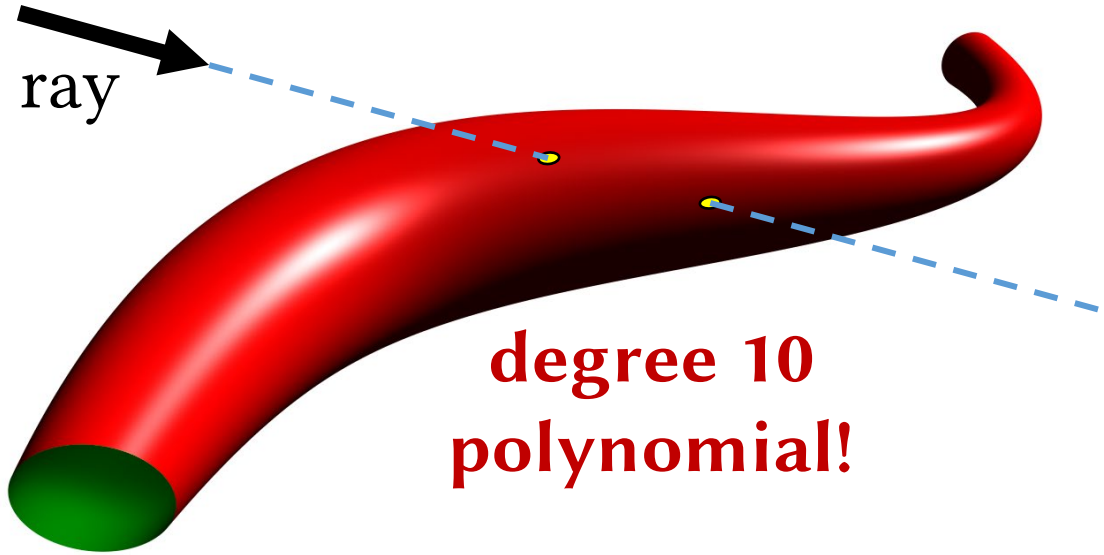
- Degree 2 (quadratic)
- Degree 3 (cubic)

$$ax^3 + bx^2 + cx + d$$



Polynomials

- Degree 2 (quadratic)
- Degree 3 (cubic)
- Degree 4
- Degree 5
- Degree 6
- Degree 7
- Degree 8
- Degree 9
- Degree 10



Degree 3 (cubic) Polynomials

- Bisection

A Class of C^2 Interpolating Splines

CEM YUKSEL, University of Utah

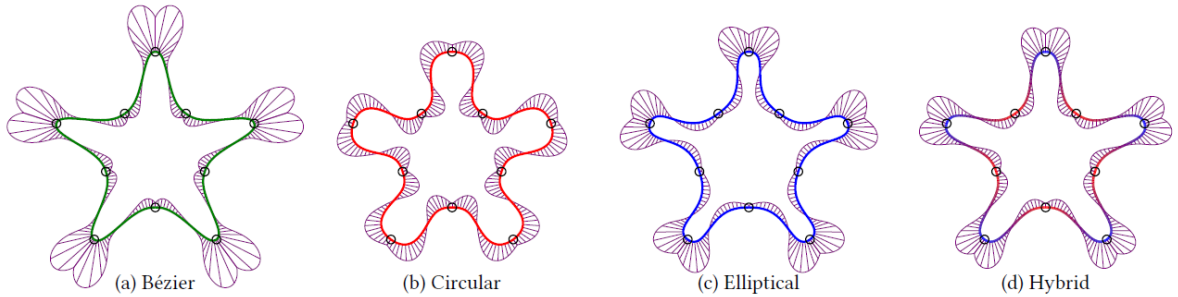


Fig. 1. *Example curves generated from the same control points using our formulation with (a) Bézier interpolation function, (b) circular interpolation function, (c) elliptical interpolation function, and (d) hybrid (circular-elliptical) interpolation function. All curves have guaranteed C^2 continuity and local support, but they produce different shapes from the same control points. The purple lines indicate the curvature of the curves.*

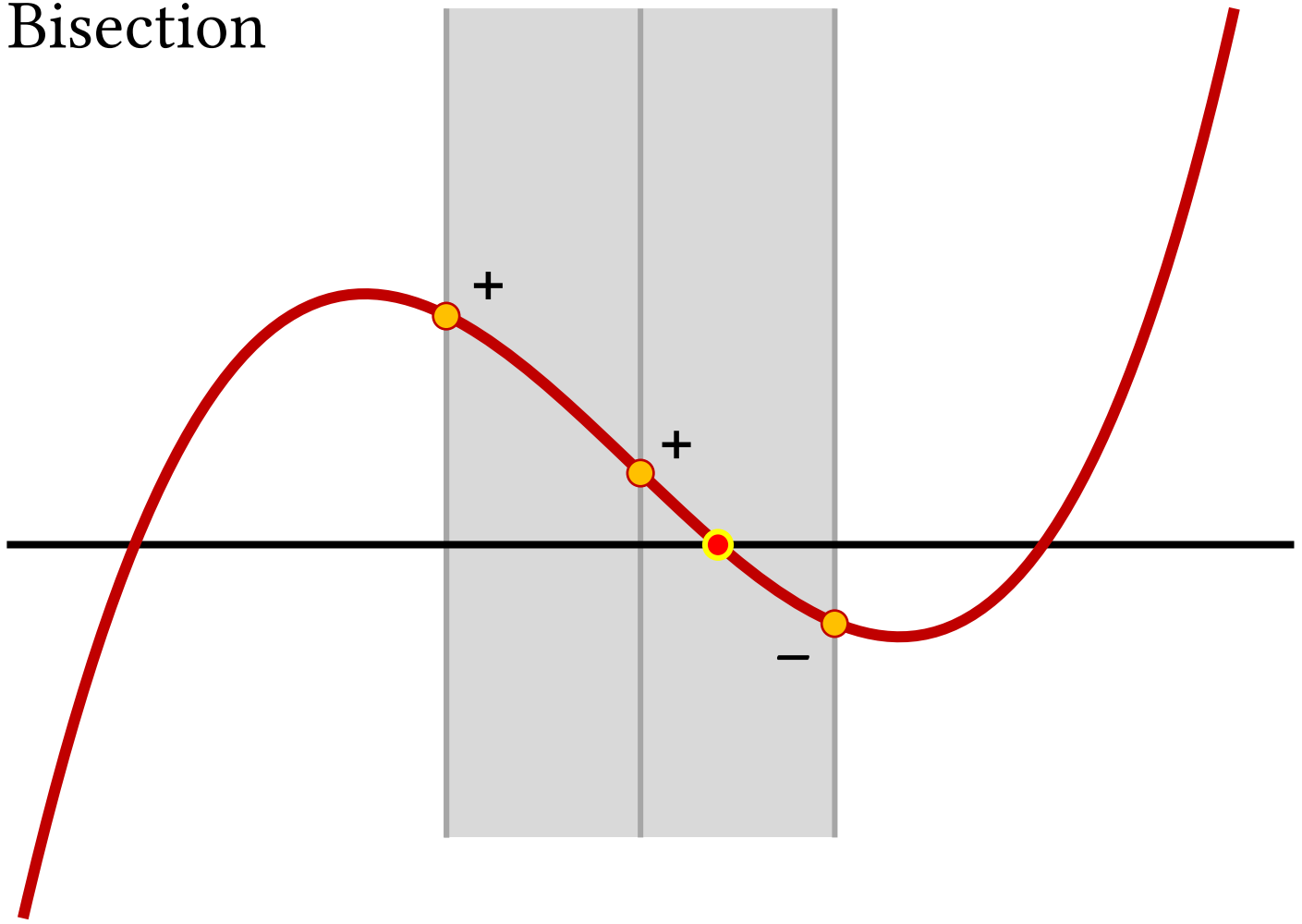
We present a class of non-polynomial parametric splines that interpolate the given control points and show that some curve types in this class have a set of highly desirable properties that were not previously demonstrated for interpolating curves before. In particular, the formulation of this class guarantees that the resulting curves have C^2 continuity everywhere and local support, such that only four control points define each curve segment between consecutive control points. These properties are achieved directly due to the mathematical formulation used for defining this class, without

1 INTRODUCTION

Polynomial parametric splines have served us well in computer graphics. Among them, approximating splines (such as B-splines and NURBS) have been particularly popular, as they can provide important properties like C^2 continuity and local support, and they are not prone to producing unintended cusps and self-intersections. However, approximating splines do not go through the control

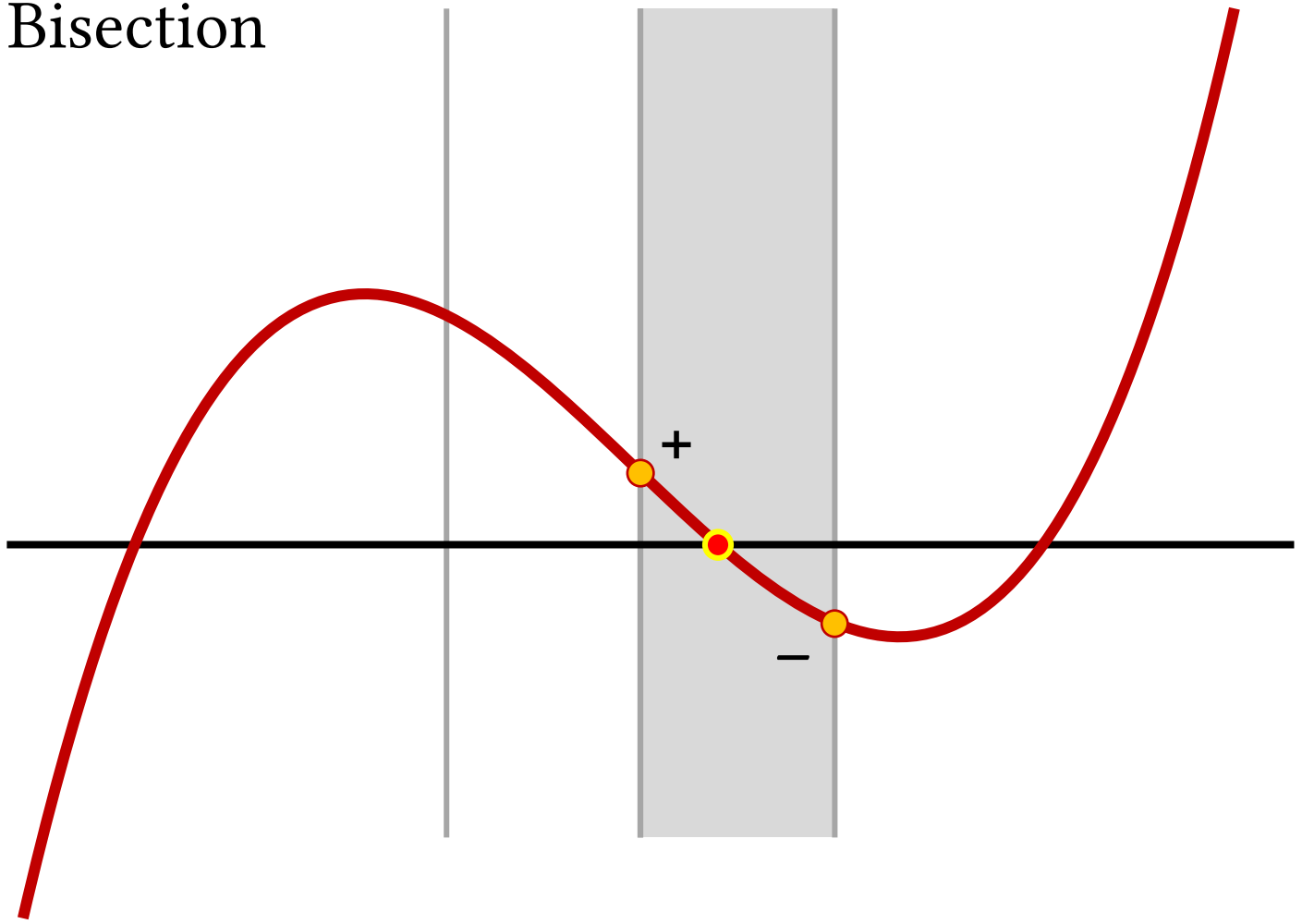
Degree 3 (cubic) Polynomials

- Bisection



Degree 3 (cubic) Polynomials

- Bisection



Degree 3 (cubic) Polynomials

- Bisection
- Newton iterations

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Degree 3 (cubic) Polynomials

- Bisection ϵ
- Newton iterations

- Analytical

  Cubic root + trigonometric functions

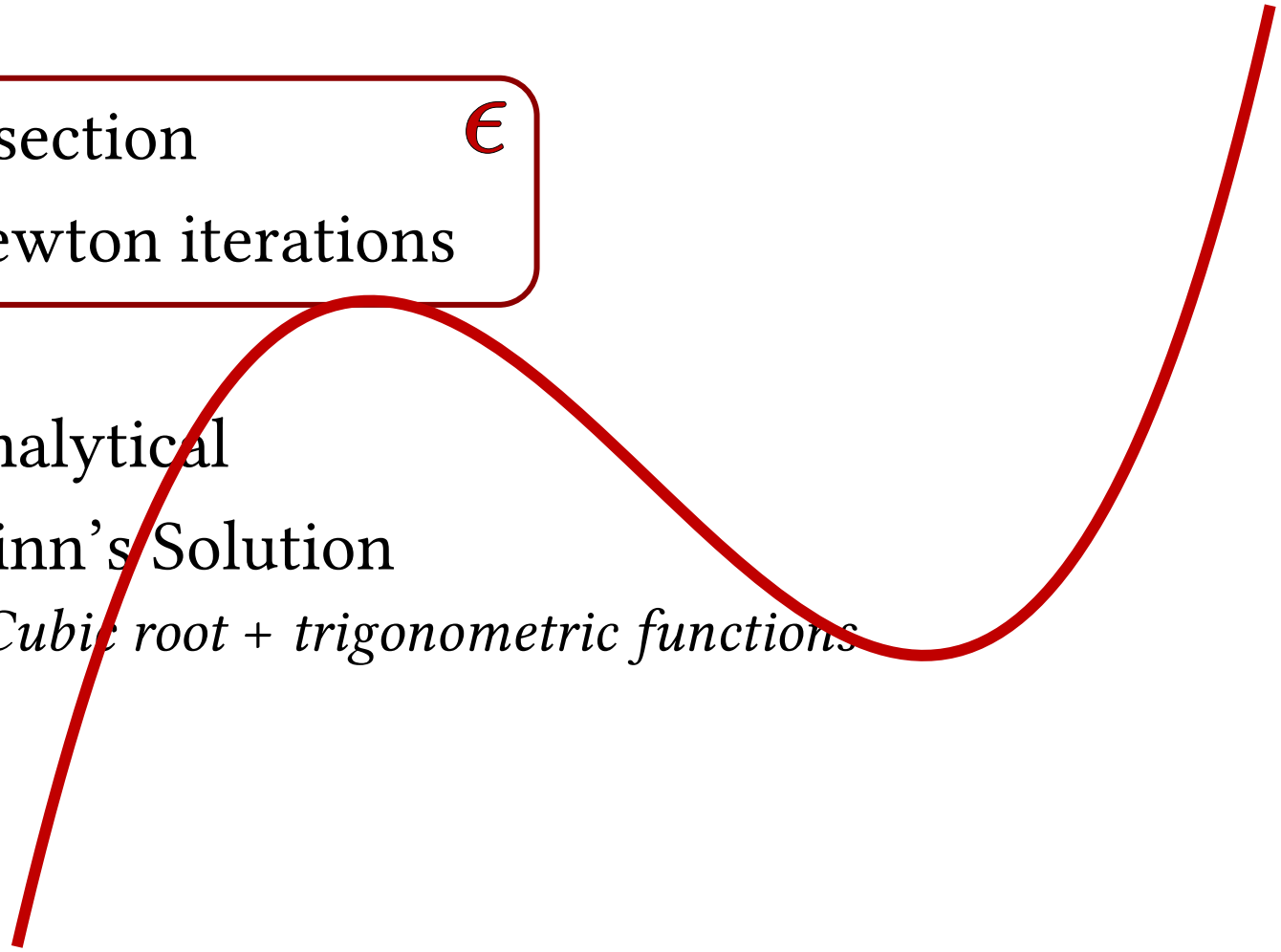
- Blinn's Solution

Degree 3 (cubic) Polynomials

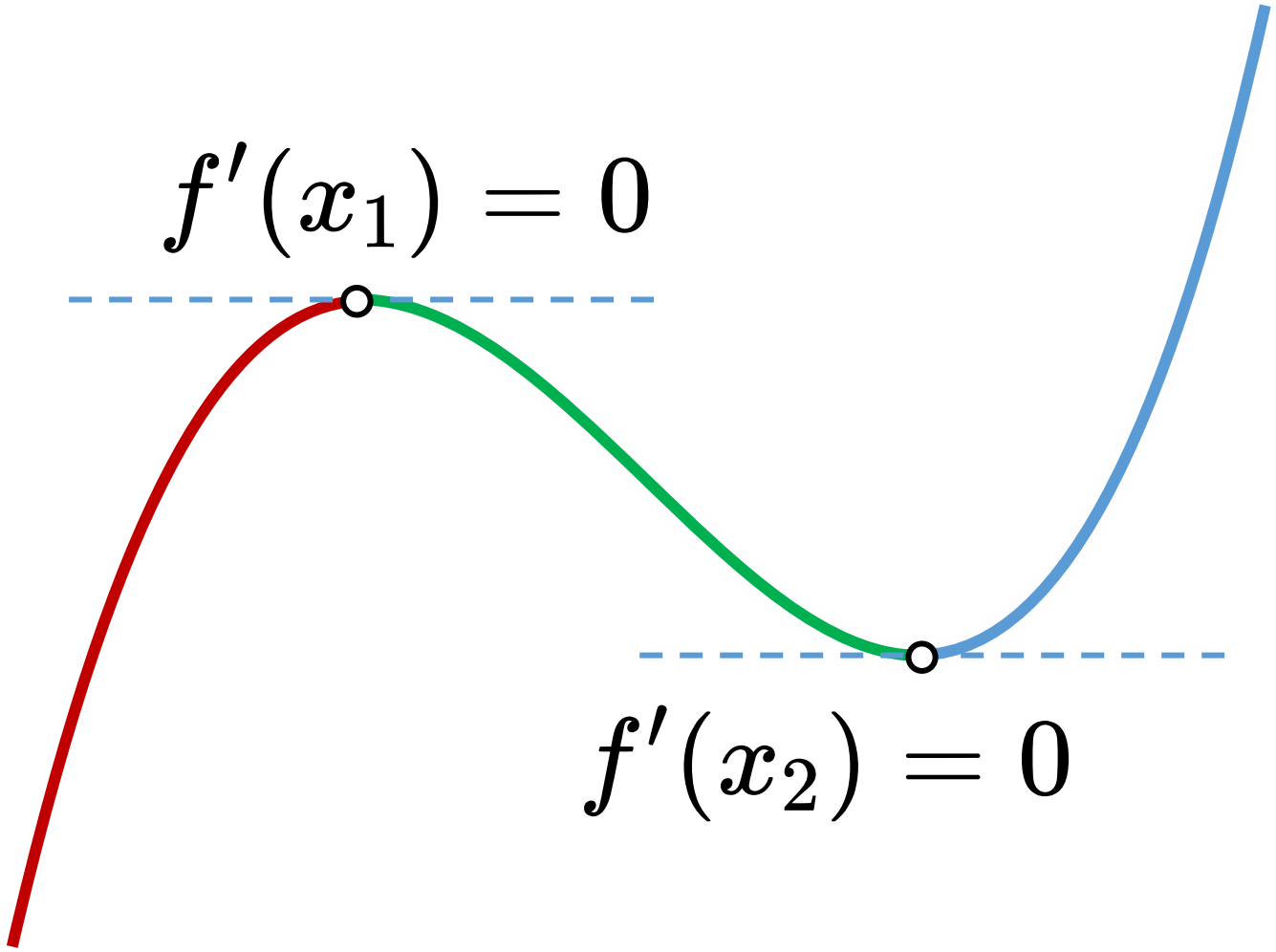
- Bisection ϵ
- Newton iterations

- Analytical
- Blinn's Solution

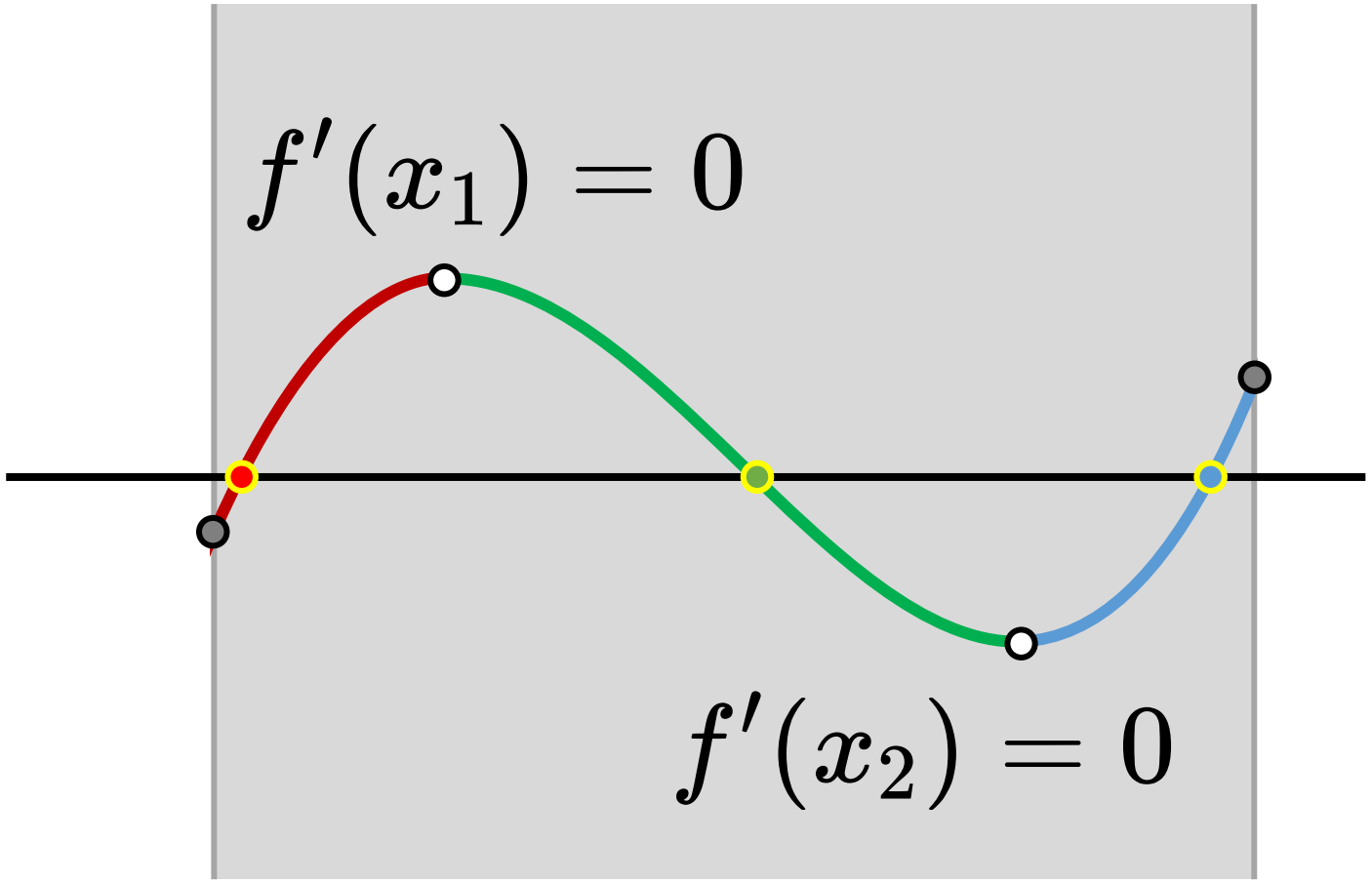
  Cubic root + trigonometric functions



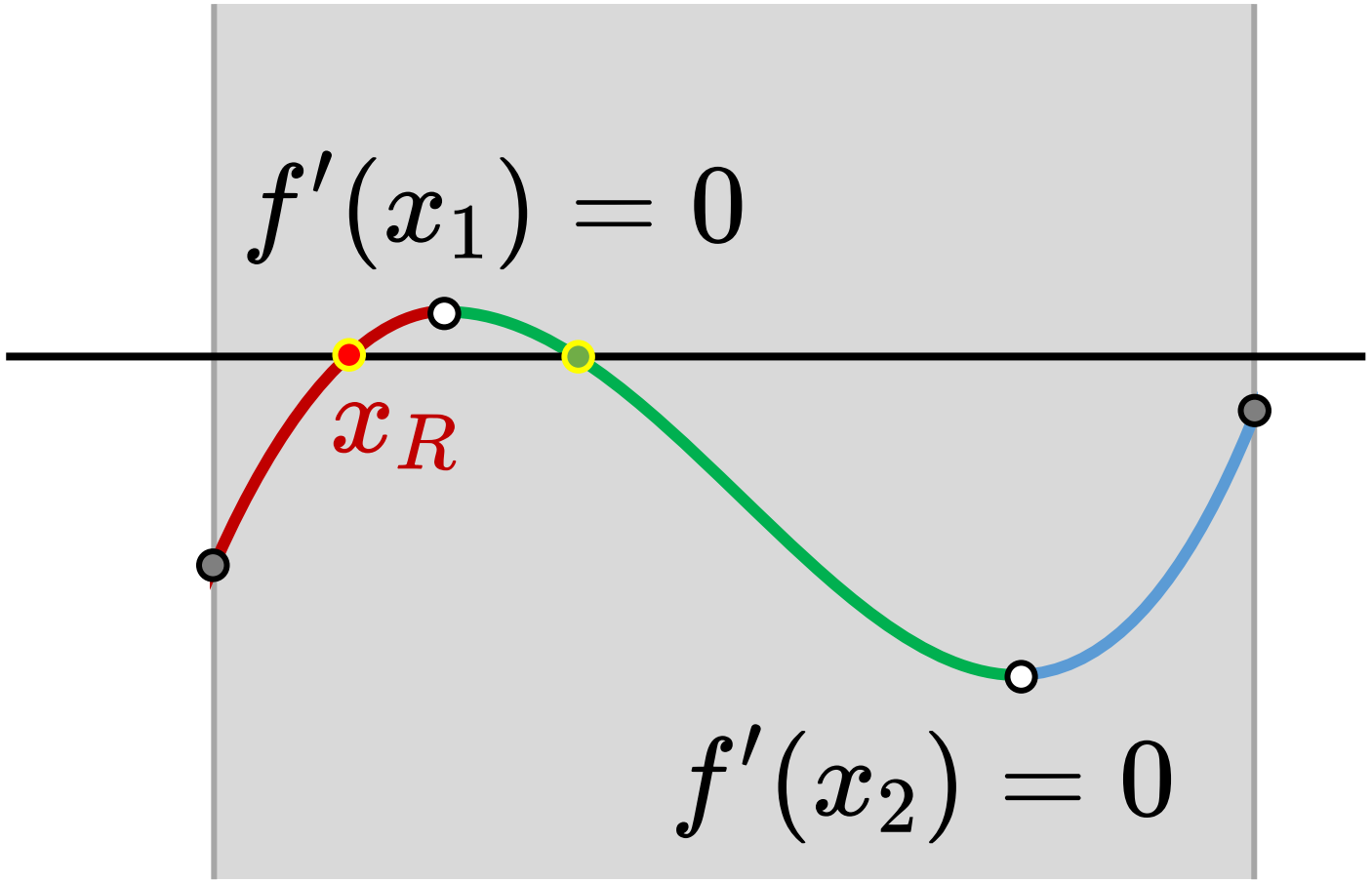
Degree 3 (cubic) Polynomials



Degree 3 (cubic) Polynomials



Degree 3 (cubic) Polynomials



Degree 3 (cubic) Polynomials

Deflation:

$$f(x) = a\,x^3 + b\,x^2 + c\,x + d$$

$$f(x) = (x - x_R)(a'x^2 + b'x + c')$$

$$f(x_R) = 0$$

Degree 3 (cubic) Polynomials

Deflation:

$$f(x) = (x - x_R)(a'x^2 + b'x + c')$$

$$a' = a$$

$$b' = b + a'x_R$$

$$c' = c + b'x_R$$

Degree 3 (cubic) Polynomials

1. Split

   Roots of a degree 2 (quadratic) polynomial

2. Find one root

   Newton iterations + Bisection

3. Deflate

   Using the one root found

4. Find the other roots

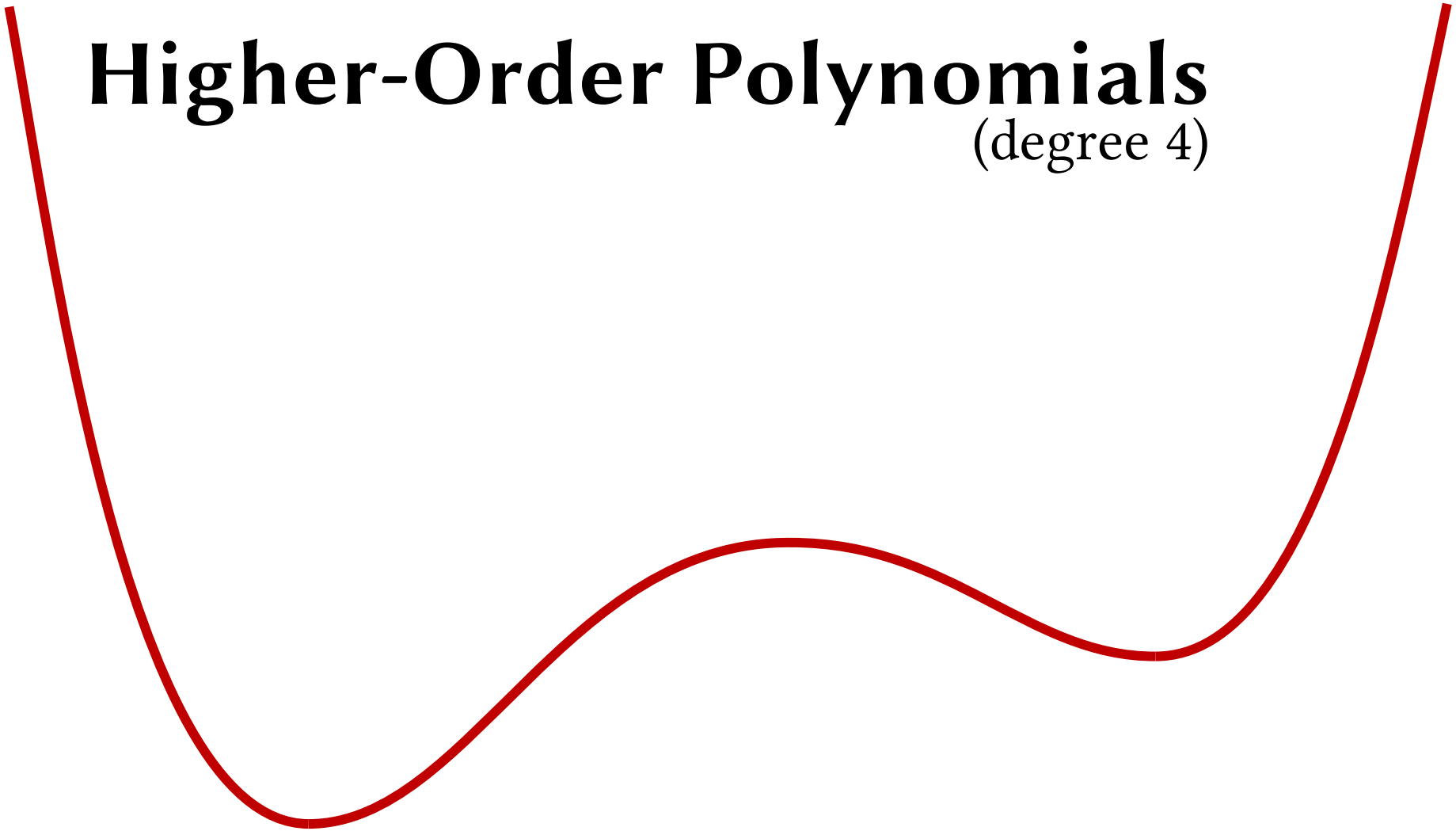
   Roots of a degree 2 (quadratic) polynomial

Higher-Order Polynomials

(degree 4+)

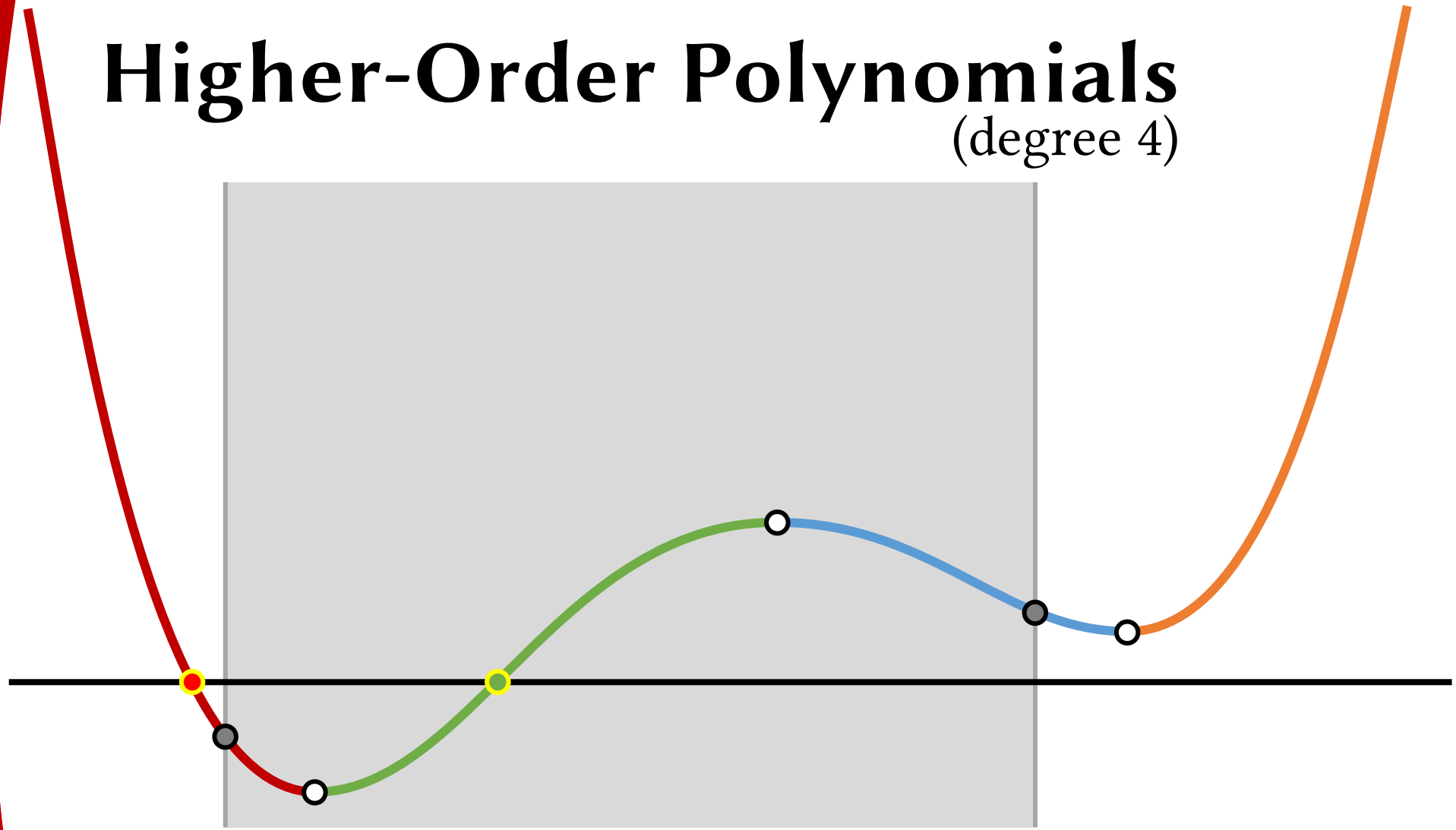
Higher-Order Polynomials

(degree 4)

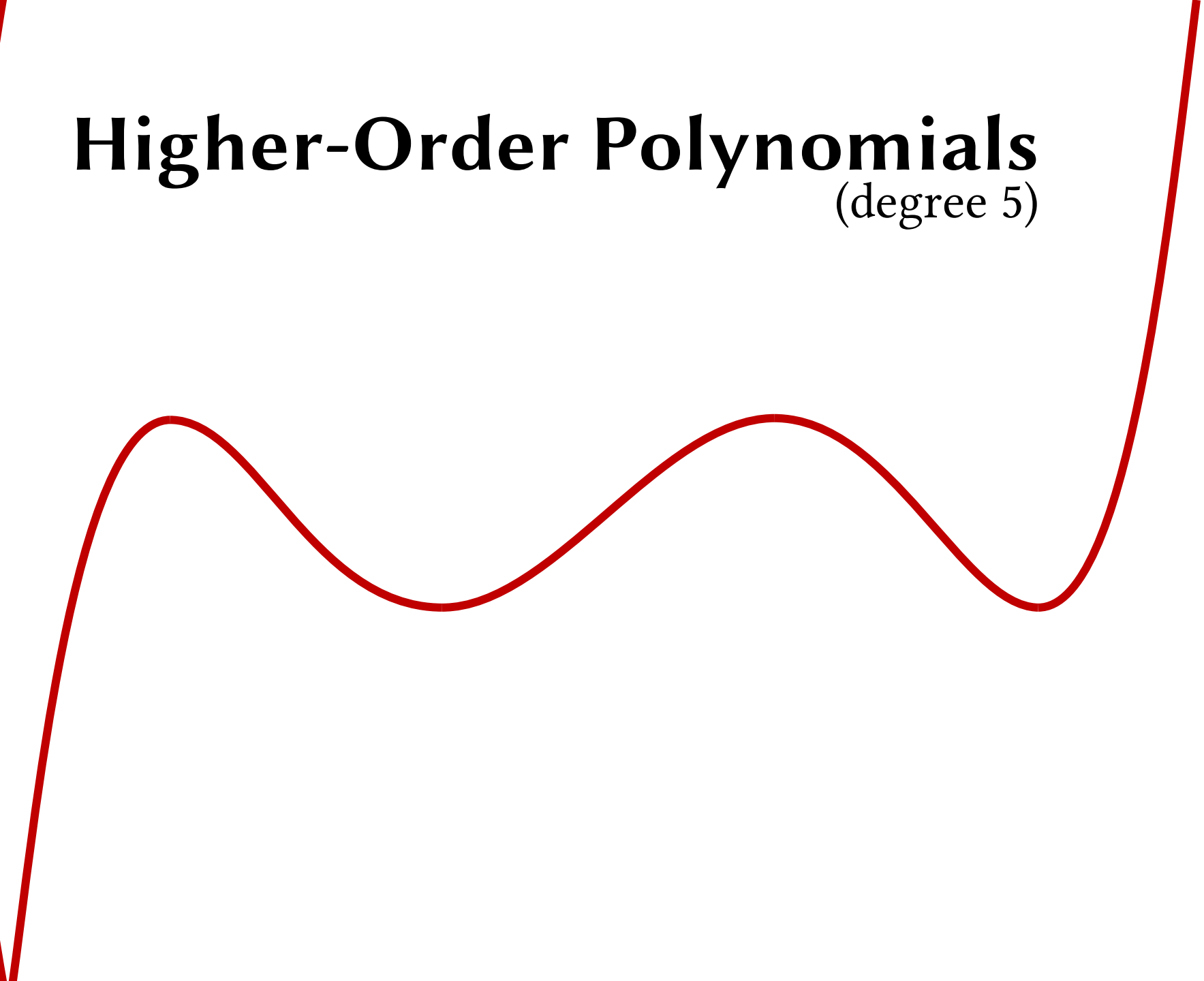


Higher-Order Polynomials

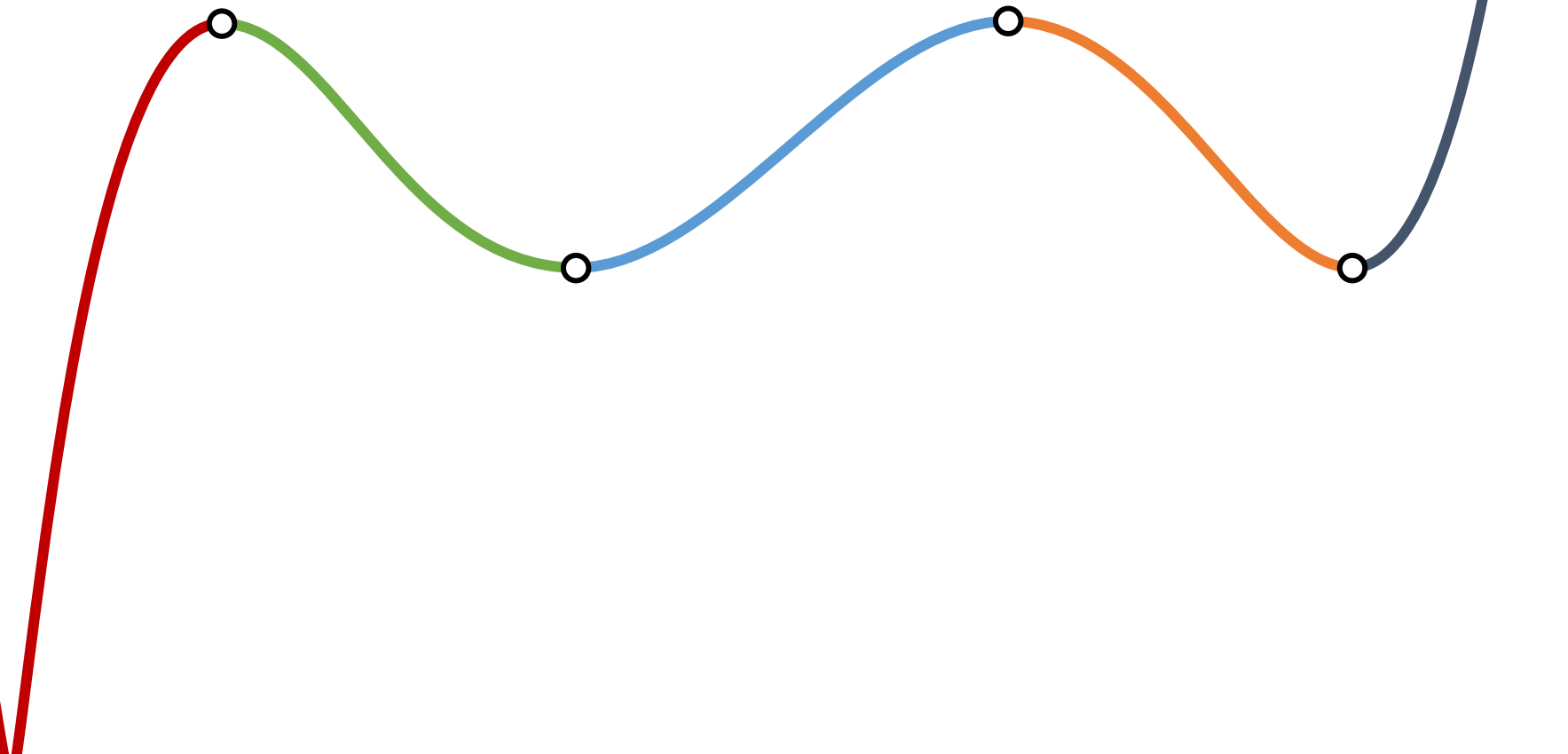
(degree 4)



Higher-Order Polynomials
(degree 5)



Higher-Order Polynomials
(degree 5)



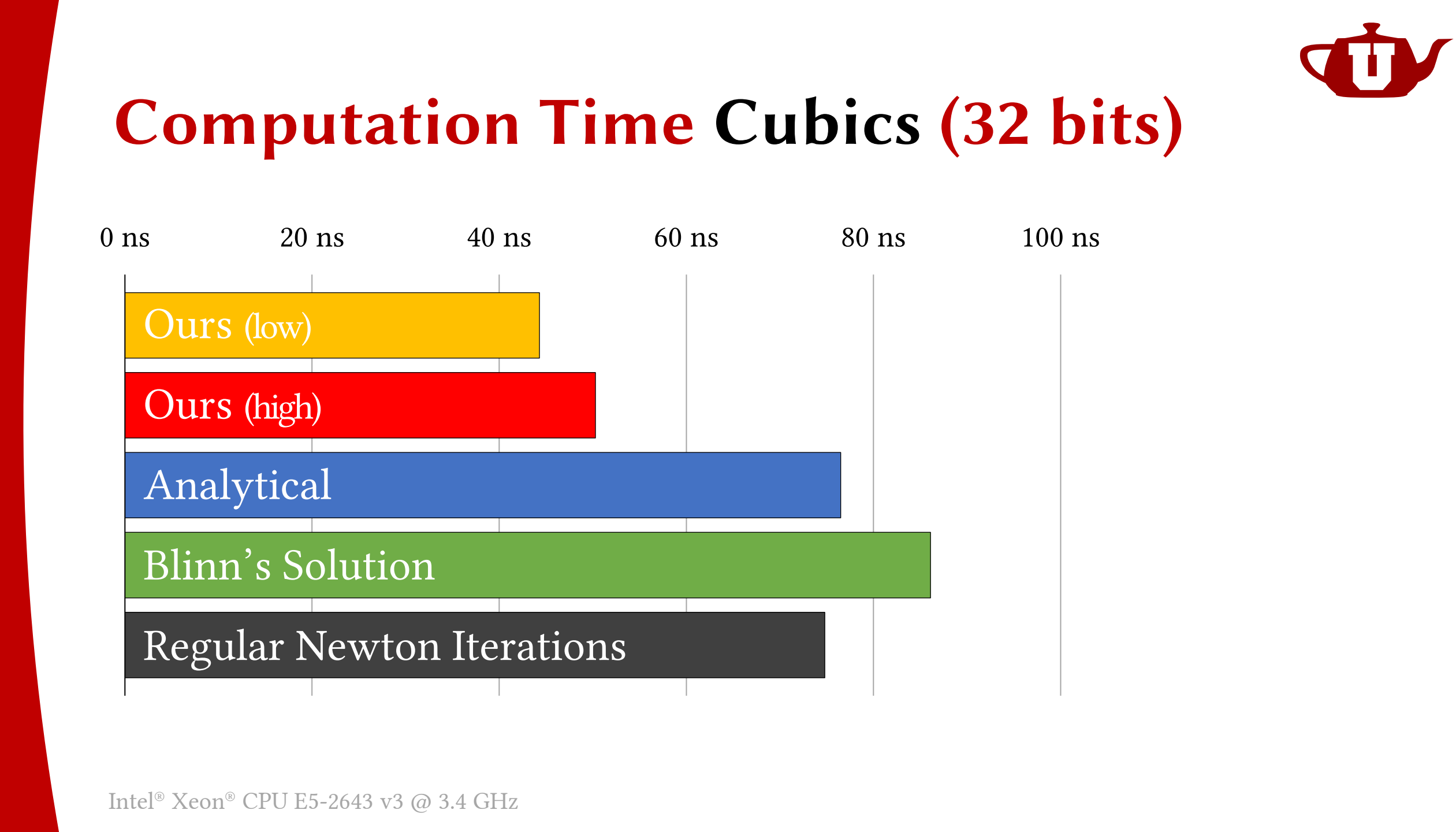
Higher-Order Polynomials

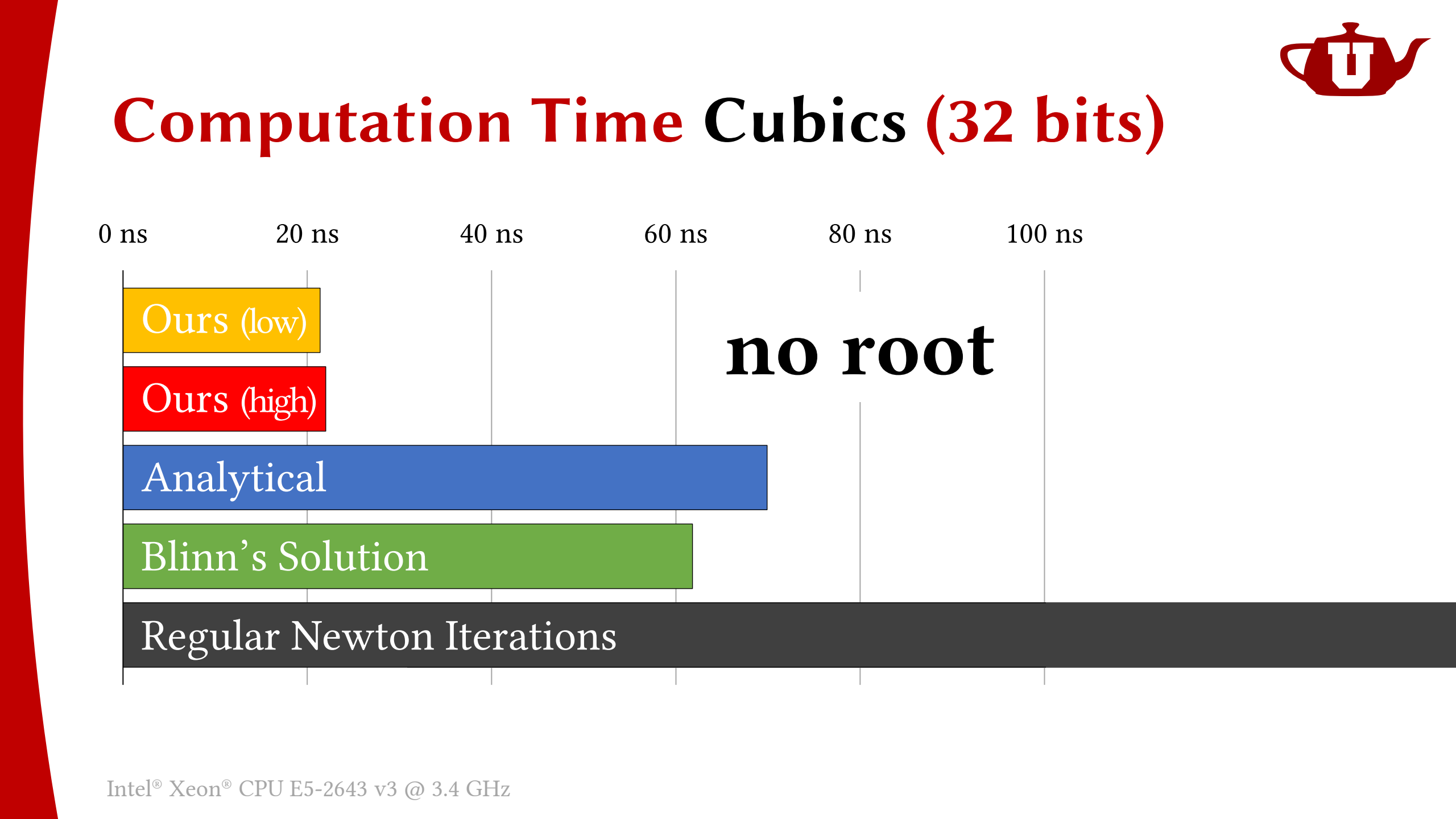
function PolynomialRoots(f, x_{start}, x_{end})

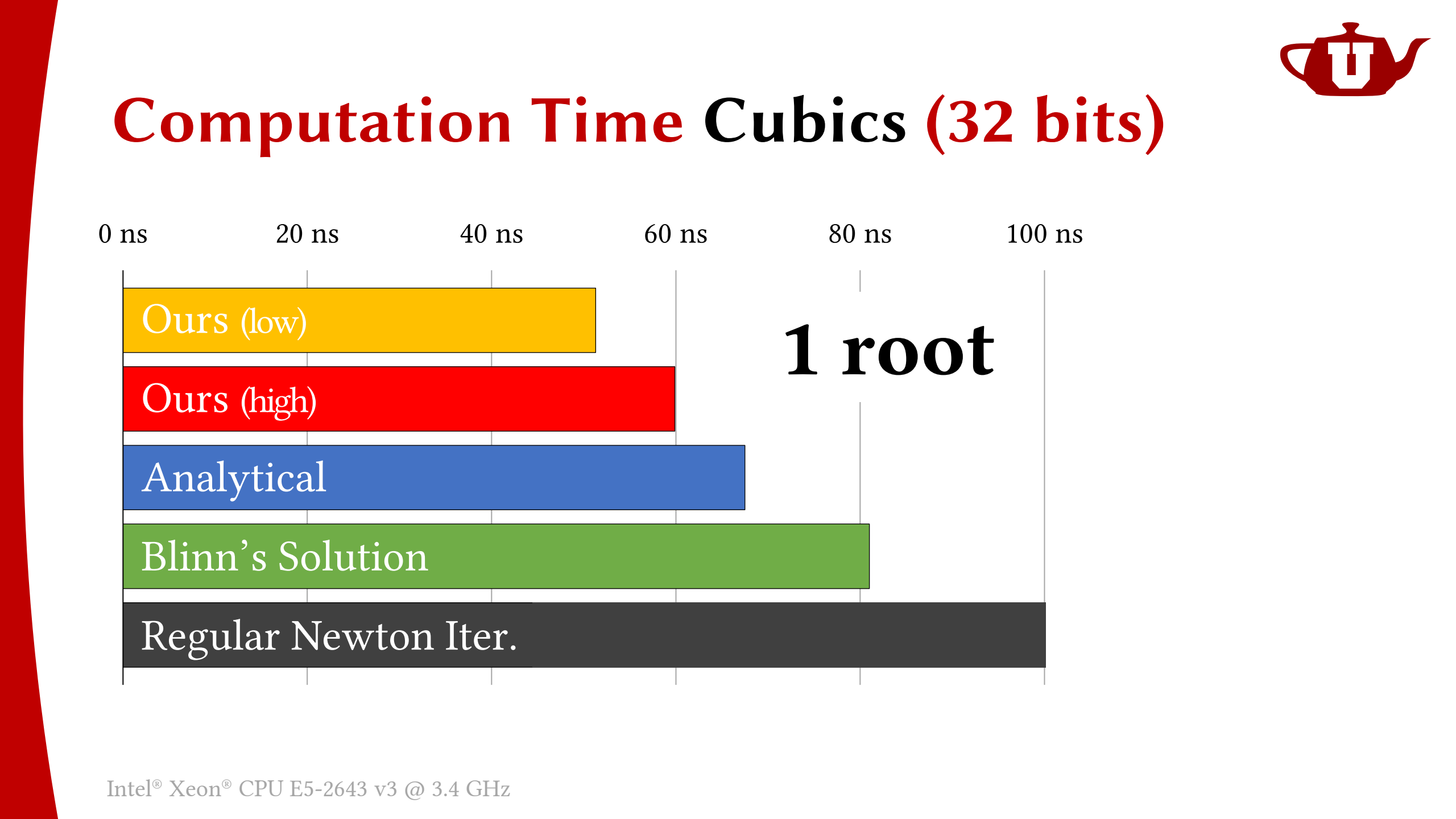
- $\mathbf{X}_C \leftarrow$ PolynomialRoots(f', x_{start}, x_{end})
- **foreach** interval x_1 to x_2 **do**
  - **if** $\operatorname{sgn} f(x_1) \neq \operatorname{sgn} f(x_2)$ **then**
    - $x_R \leftarrow$ FindRoot(f, f', x_1, x_2)
    - $\mathbf{X}_R \leftarrow \mathbf{X}_R \cup \{x_R\}$
  - **end**
- **end**
- **return** $\mathbf{X}_R$

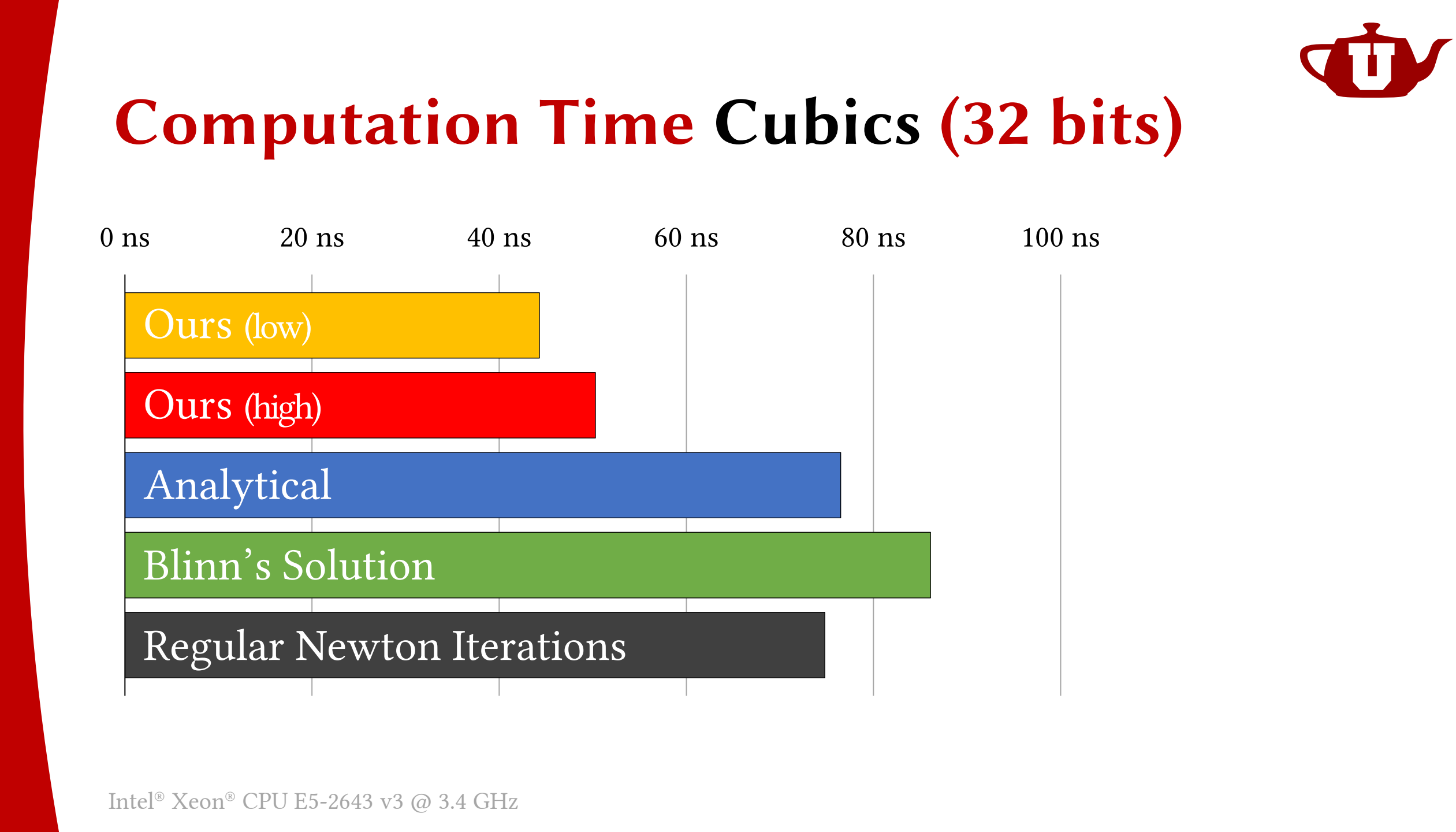
No Deflation!

Computation Time


Computation Time Cubics (32 bits)
0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
Ours (low)
Ours (high)
Analytical
Blinn's Solution
Regular Newton Iterations
Intel® Xeon® CPU E5-2643 v3 @ 3.4 GHz


Computation Time Cubics (32 bits)
0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
Ours (low)
Ours (high)
Analytical
Blinn's Solution
Regular Newton Iterations
no root
Intel® Xeon® CPU E5-2643 v3 @ 3.4 GHz


Computation Time Cubics (32 bits)
0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
Ours (low)
Ours (high)
Analytical
Blinn's Solution
Regular Newton Iter.
1 root
Intel® Xeon® CPU E5-2643 v3 @ 3.4 GHz


Computation Time Cubics (32 bits)
0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
Ours (low)
Ours (high)
Analytical
Blinn's Solution
Regular Newton Iterations
Intel® Xeon® CPU E5-2643 v3 @ 3.4 GHz

Computation Time (32 bits)

degree 2: <10 ns

degree 3: ~45 ns

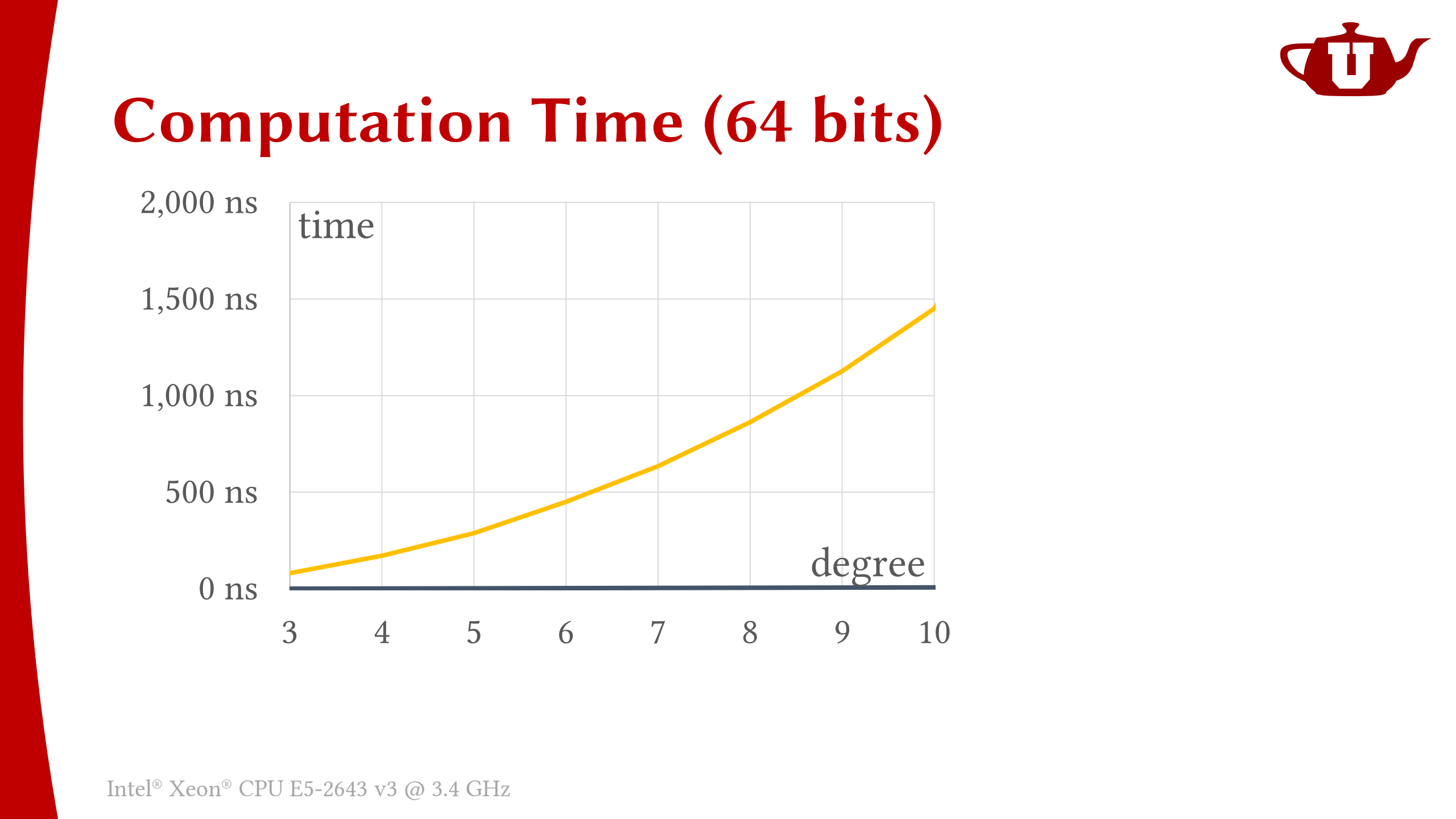
degree 4: ~110 ns

degree 5: ~200 ns

degree 6: ~300 ns

degree 7: ~420 ns

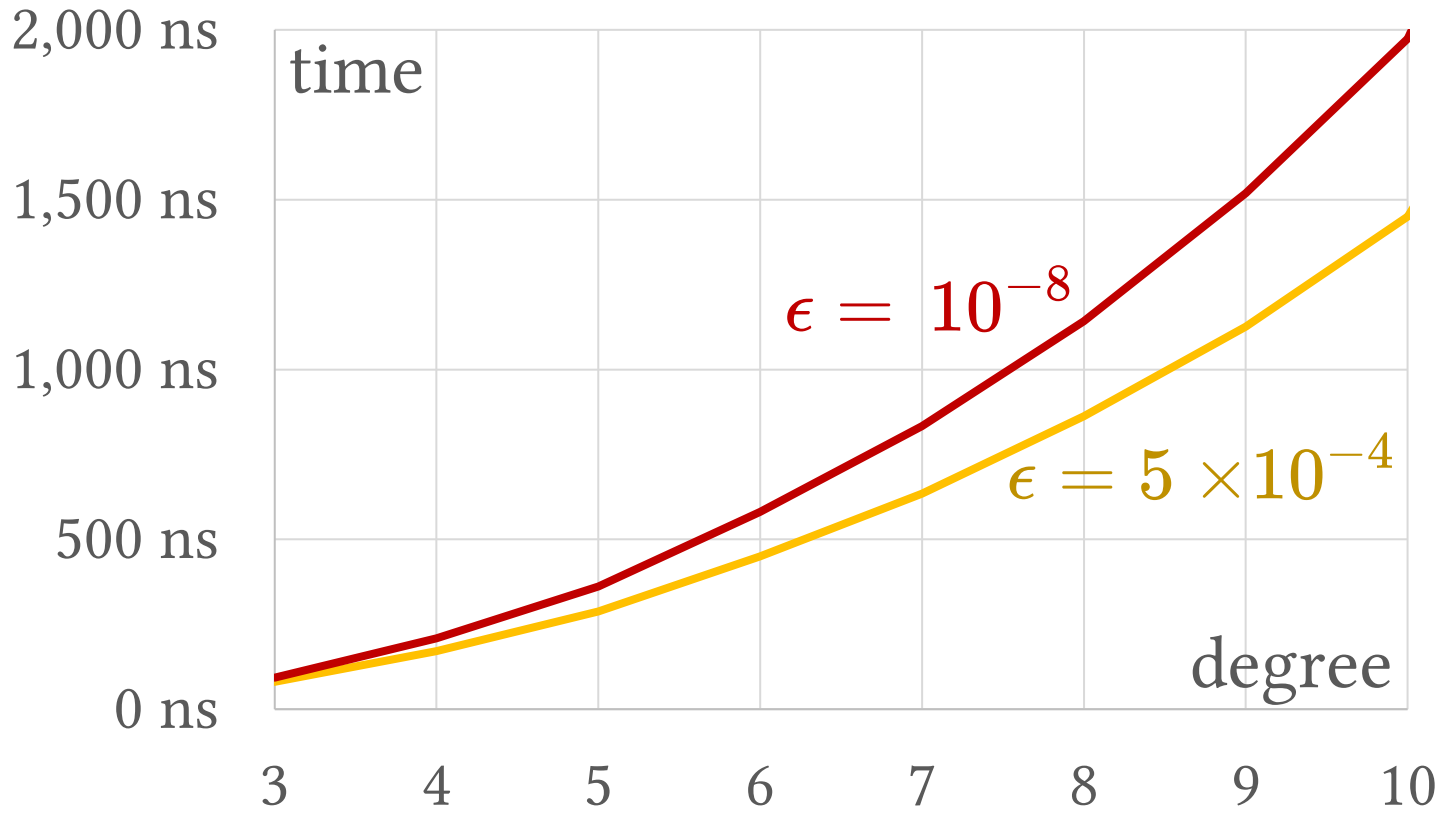
Intel® Xeon® CPU E5-2643 v3 @ 3.4 GHz



Computation Time (64 bits)

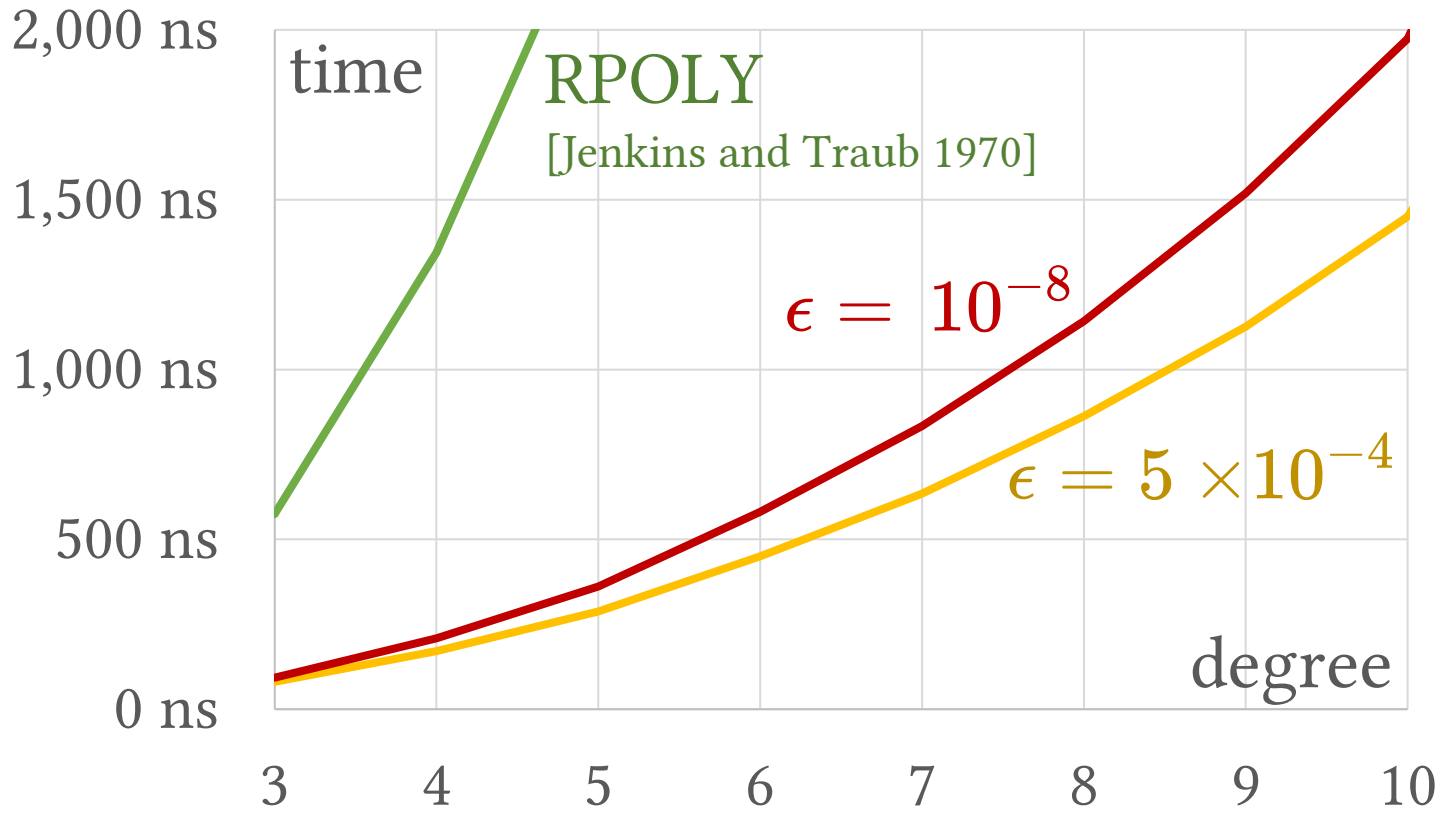
Intel® Xeon® CPU E5-2643 v3 @ 3.4 GHz

Computation Time (64 bits)



Intel® Xeon® CPU E5-2643 v3 @ 3.4 GHz

Computation Time (64 bits)

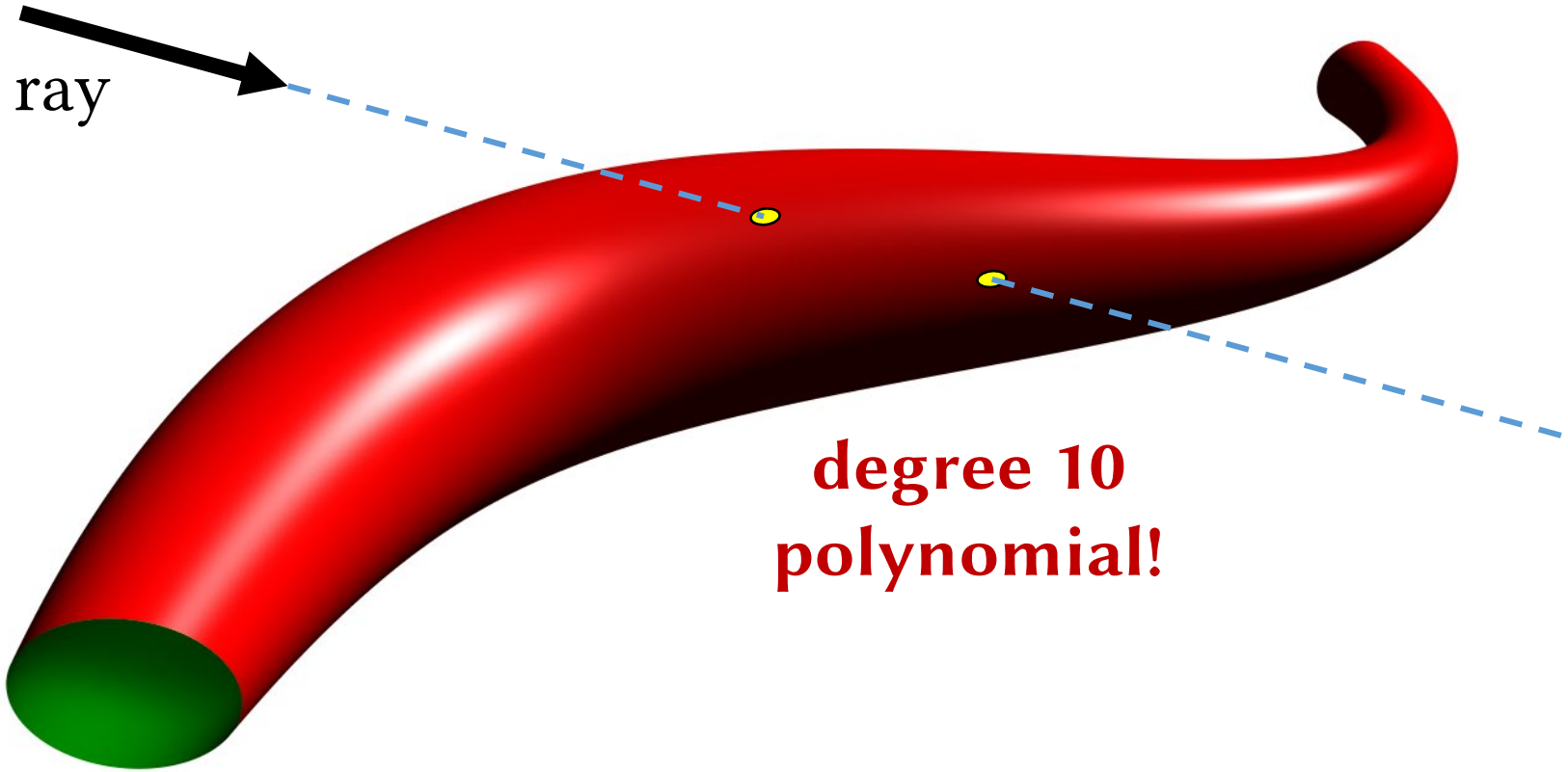


Intel® Xeon® CPU E5-2643 v3 @ 3.4 GHz

Challenge:
Hair Rendering

Challenge: Hair Rendering

Challenge: Hair Rendering



Challenge: Hair Rendering

Brute-Force Polynomial Root Finder

vs.

Phantom Ray-Hair Intersector [Reshetov and Luebke 2018]

Challenge: Hair Rendering

Challenge: Hair Rendering

Cubic hair curves:

Brute-Force Polynomial Root Finder (degree 10)

1.0x

Phantom Ray-Hair Intersector

1.3x

Challenge: Hair Rendering

Cubic hair curves (5x):

Brute-Force Polynomial Root Finder (degree 10)

1.0x

Phantom Ray-Hair Intersector

1.8x

Challenge: Hair Rendering

Quadratic hair curves (5x):

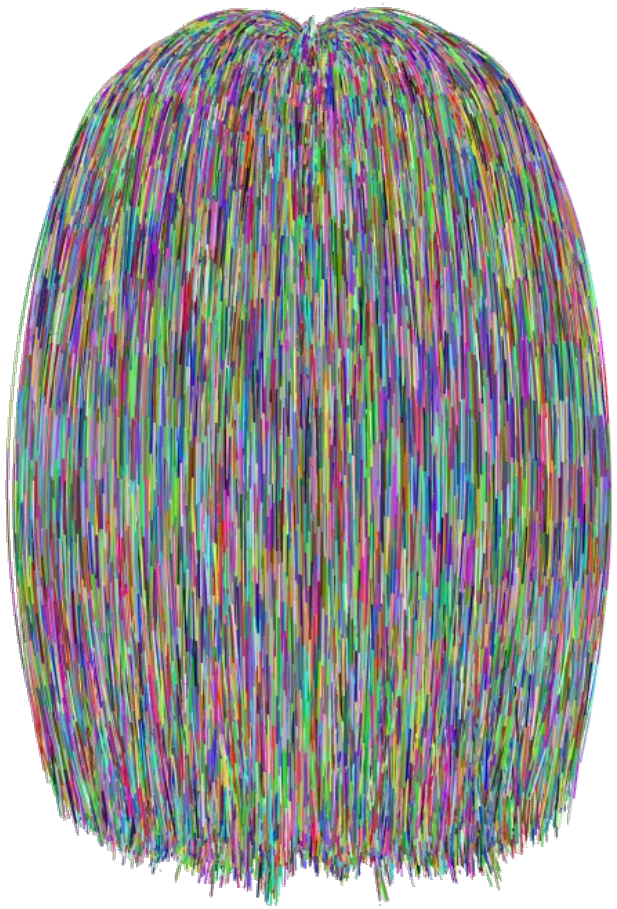
Brute-Force Polynomial Root Finder (degree 6)

1.0x

Phantom Ray-Hair Intersector

3.6x

Challenge: Hair Rendering



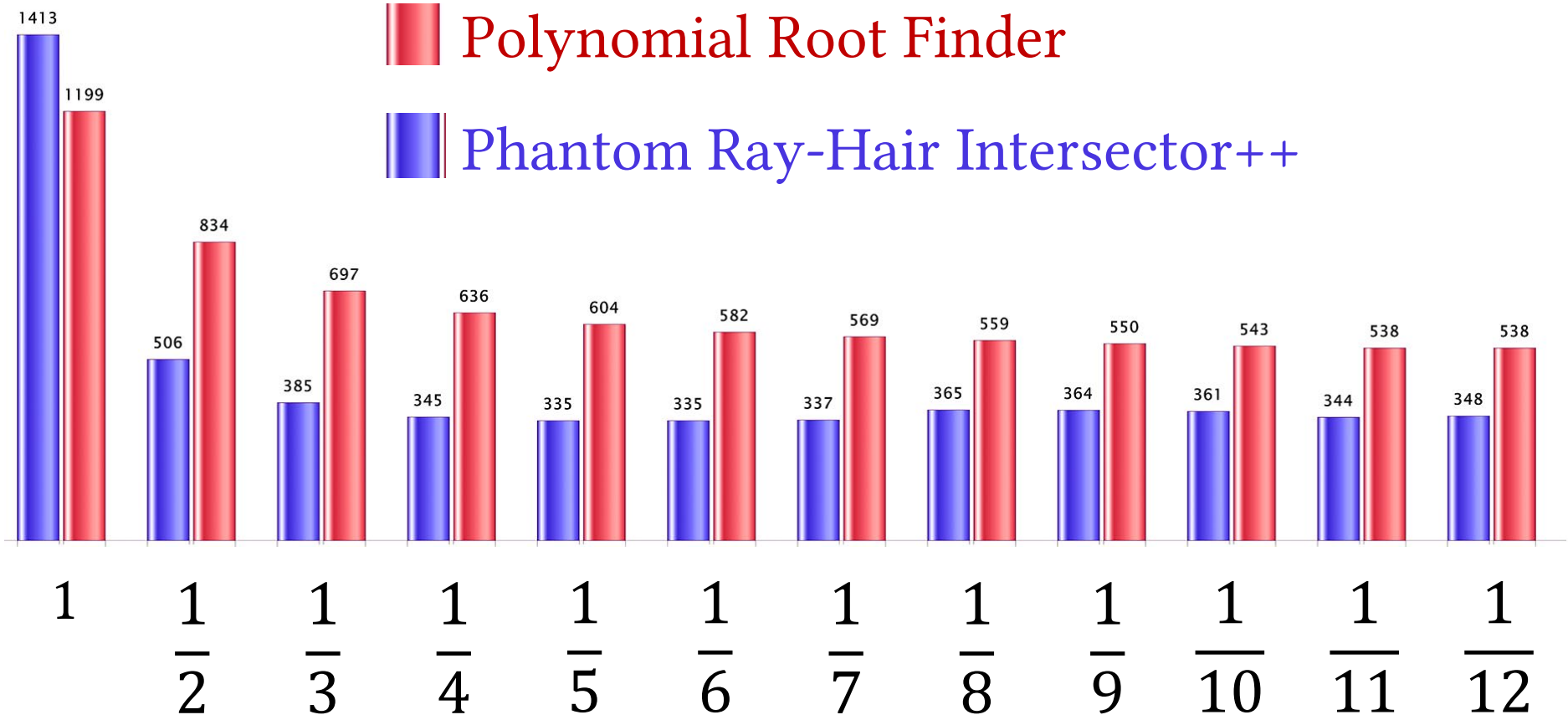
Challenge: Hair Rendering

Brute-Force Polynomial Root Finder

vs.

Phantom Ray-Hair Intersector [Reshetov and Luebke 2018]

Random Thick Curve Intersection



tests by Alexander Reshetov

Source Code & More

http://www.cemyuksel.com/?x=polynomials

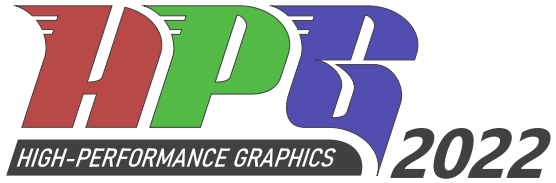
```
include <cyPolynomial.h>

...

cy::Polynomial<double,5> poly =
    { -0.01, 1.0, -2.0, 3.0, -4.0, 0.5 };
double roots[5];
double intervalMin = 0.0;
double intervalMax = 1.0;
double errorThreshold = 0.0001;
int numRoots = poly.Roots( roots,
                           intervalMin,
                           intervalMax,
                           errorThreshold );
```

Acknowledgements

- Alexander Reshetov
- Jim McCann, Ian Mallett, Kui Wu, Daqi Lin, Peter Shirley, Nathan Morrical, Stefan Zellmann
- HPG 2022 IPC/Reviewers
- NSF Grant #1764071

SIGGRAPH 2022
VANCOUVER+ 8-11 AUG

Cem Yuksel. 2022.
A Fast & Robust Solution
for Cubic & Higher-Order Polynomials.
SIGGRAPH 2022 Talks.



High-Performance Polynomial Root Finding for Graphics

Cem Yuksel, *University of Utah*

http://www.cemyuksel.com/?x=polynomials