

MASSIMILIANO MANTIONE
GIANLUCA CARUCCI

the cost of abstractions

TSconf

25 OCTOBER 2019



@M_a_s_s_i



www.linkedin.com/in/massimilianomantione



fbk.com/caruccigianluca



gianlucacarucci5



www.linkedin.com/in/rucka

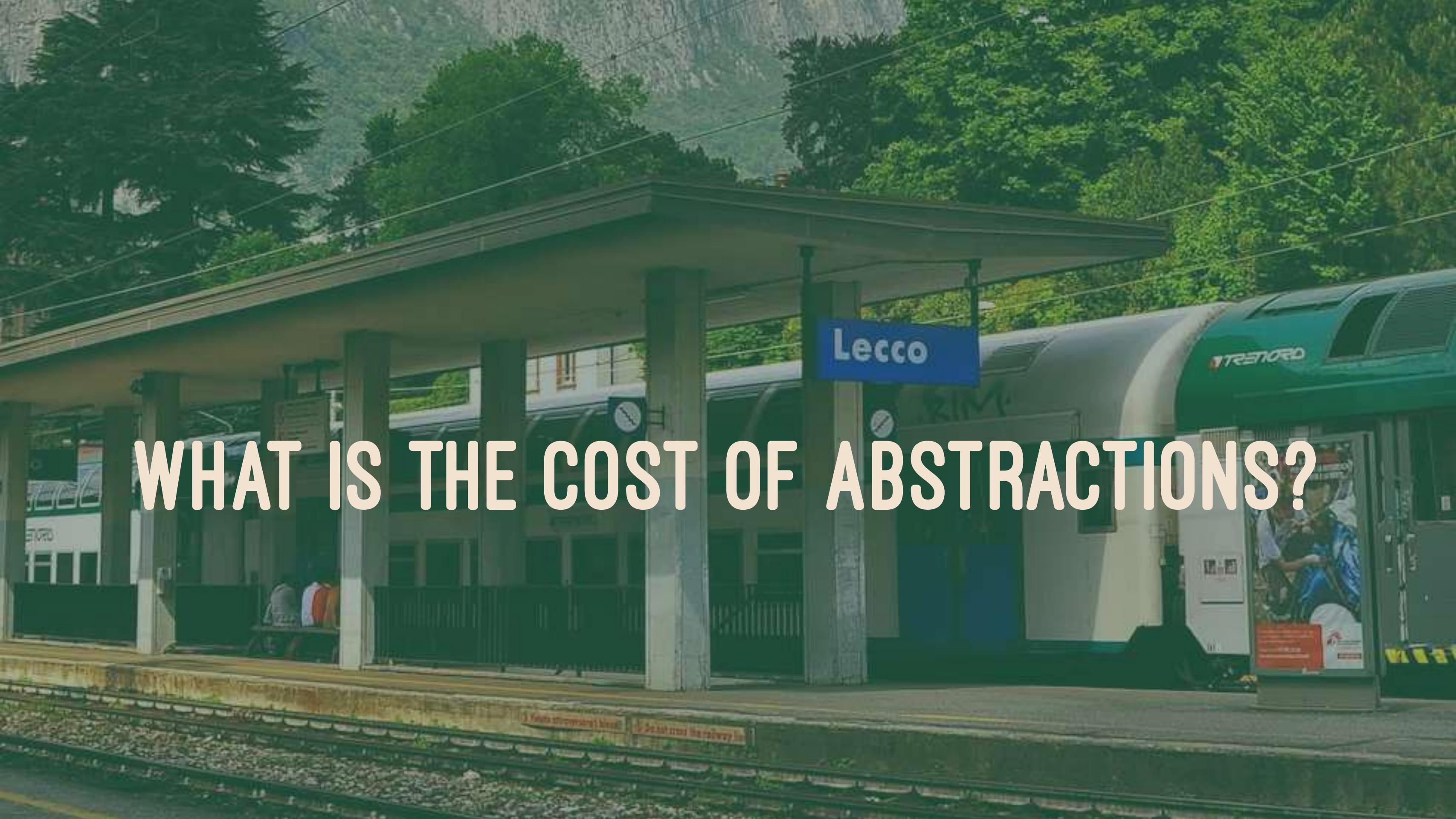


ONCE UPON A TIME...

Lecco

TRENIORD

Lecco
Trenord

A high-speed train, identified by the 'TRENIORD' logo on its side, is stopped at a station platform. The platform has a green roof and a blue sign that reads 'Lecco'. The background shows a lush green hillside with trees and power lines.

WHAT IS THE COST OF ABSTRACTIONS?



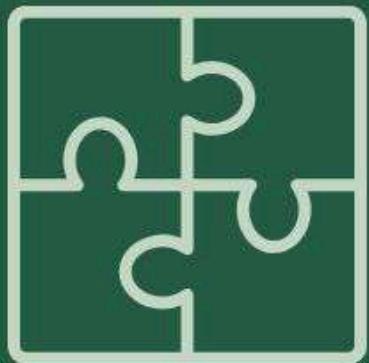
ehm...



POINT OF VIEW



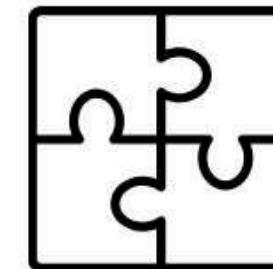
performance



maintainability



Gianluca Carucci



Software Engineer

"Computer programs are the most complex things that humans make."

- Douglas Crockford,
JavaScript: The Good Parts



Massimiliano Mantione



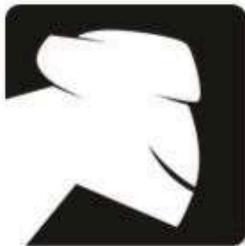
Software Engineer

"Knuth taught me the answers.
Dijkstra taught me the questions."

- Alexander Stepanov,
(main author of C++ STL)



Google



SIEMENS





WHAT DOES COST MEAN?



IT'S NOT ALL ABOUT PERFORMANCE

CONSIDER TYPESCRIPT
AS ABSTRACTION
OVER JAVASCRIPT

TRANSPILE FROM TYPESCRIPT

```
const processor: AsyncProcessor = async (
  orderId: string): Promise<PlacedOrderResult> => {
  const order = await orderService(orderId)
  if (order == null) {
    return {
      success: false
    }
  }
  const validationResult = await validationService(order)
  if (!validationResult.valid) {
    return placedOrderFailed
  }
  return await placeOrderService(order)
}
```

TO JAVASCRIPT

```
const processor = async (orderId) => {
  const order = await orderService(orderId)
  if (order == null) {
    return {
      success: false
    }
  }
  const validationResult = await validationService(order)
  if (!validationResult.valid) {
    return placedOrderFailed
  }
  return await placeOrderService(order)
}
```

CODE IS NOT MODIFIED
TYPES ARE STRIPPED

IT'S NOT ALL ABOUT PERFORMANCE



- › NO PERFORMANCE PENALTY

IT'S NOT ALL ABOUT PERFORMANCE



- > NO PERFORMANCE PENALTY
- > YOU WILL PAY A BUILD TIME COST



IT'S NOT ALL ABOUT PERFORMANCE

- NO PERFORMANCE PENALTY
- YOU WILL PAY A BUILD TIME COST
- COGNITIVE OVERHEAD

COGNITIVE OVERHEAD

COGNITIVE: SOMETHING WE SHOULD KNOW
OVERHEAD: AN EXTRA EFFORT

COGNITIVE OVERHEAD

COGNITIVE: SOMETHING WE SHOULD KNOW
OVERHEAD: AN EXTRA EFFORT

THE EXTRA LEARNING EFFORT THE WHOLE TEAM MUST SPEND!



PERFORMANCE MATTERS

LET'S MEASURE

"BUY A BOOK" USE CASE

> CREATE AN ORDER OF BOOKS



"BUY A BOOK" USE CASE

- > CREATE AN ORDER OF BOOKS
 - > VALIDATE THE ORDER

"BUY A BOOK" USE CASE

- > CREATE AN ORDER OF BOOKS
 - > VALIDATE THE ORDER
- > COMPUTE AND PLACE THE ORDER

BENCHMARK

> 'BUY A BOOK' USE CASE



BENCHMARK

- › 'BUY A BOOK' USE CASE
- › PROCESS 500K ORDERS



BENCHMARK

- > 'BUY A BOOK' USE CASE
- > PROCESS 500K ORDERS
- > 5% FAILURE RATE



BENCHMARK

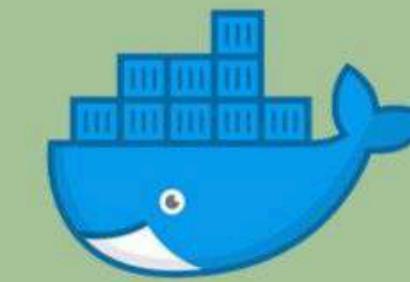
- > 'BUY A BOOK' USE CASE
- > PROCESS 500K ORDERS
- > 5% FAILURE RATE
- > MEASURE MEAN ORDER TIME



slides



code



environment



READY
STEADY
GO!

ASYNC TYPESCRIPT

```
const order = await orderService(orderId)
if (order == null) {
    return {
        success: false
    }
}
const validationResult = await validationService(order)
if (!validationResult.valid) {
    return placedOrderFailed
}
return await placeOrderService(order)
```



ASYNC





ASYNC

ASYNC TYPESCRIPT 6.772µS





NEXT STEP

ADD AN ABSTRACTION LAYER (FP-TS)

FUNCTIONAL TYPESCRIPT

```
return pipe(  
    orderService(orderId),  
    chain(validationService),  
    chain(placeOrderService)  
)
```



FROM ASYNC TO FP

ASYNC TYPESCRIPT 6.772 μ S



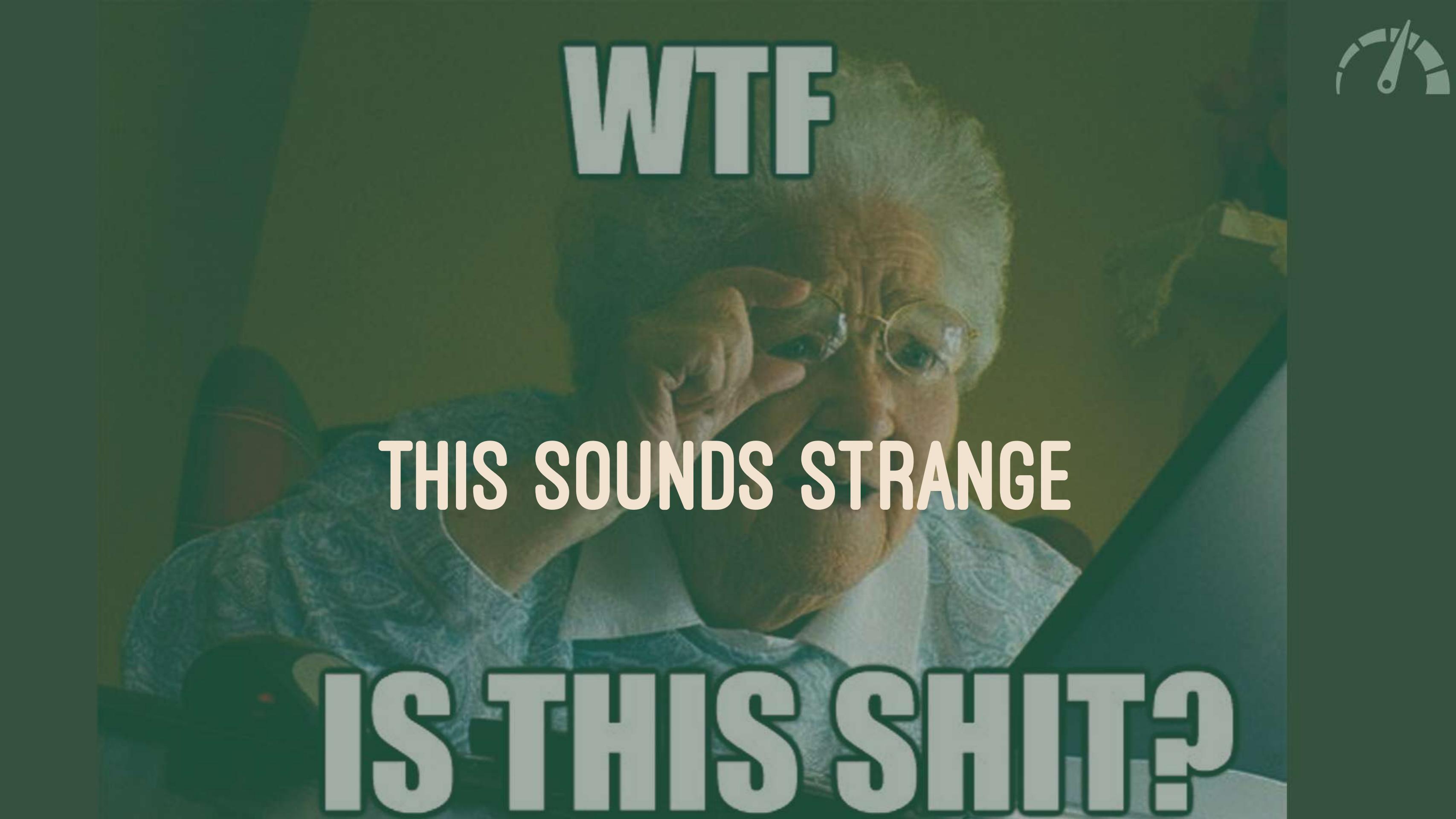


FROM ASYNC TO FP

ASYNC TYPESCRIPT 6.772 μ S

FUNCTIONAL TYPESCRIPT 5.952 μ S





WTF

THIS SOUNDS STRANGE

IS THIS SHIT?





FROM ASYNC TO FP

ASYNC TYPESCRIPT 6.772 μ S

FUNCTIONAL TYPESCRIPT 5.952 μ S





FROM ASYNC TO FP

ASYNC TYPESCRIPT (TARGET ES3) 6.772µS
FUNCTIONAL TYPESCRIPT (TARGET ES3) 5.952µS



FROM ASYNC TO FP (ES3->ES2018)

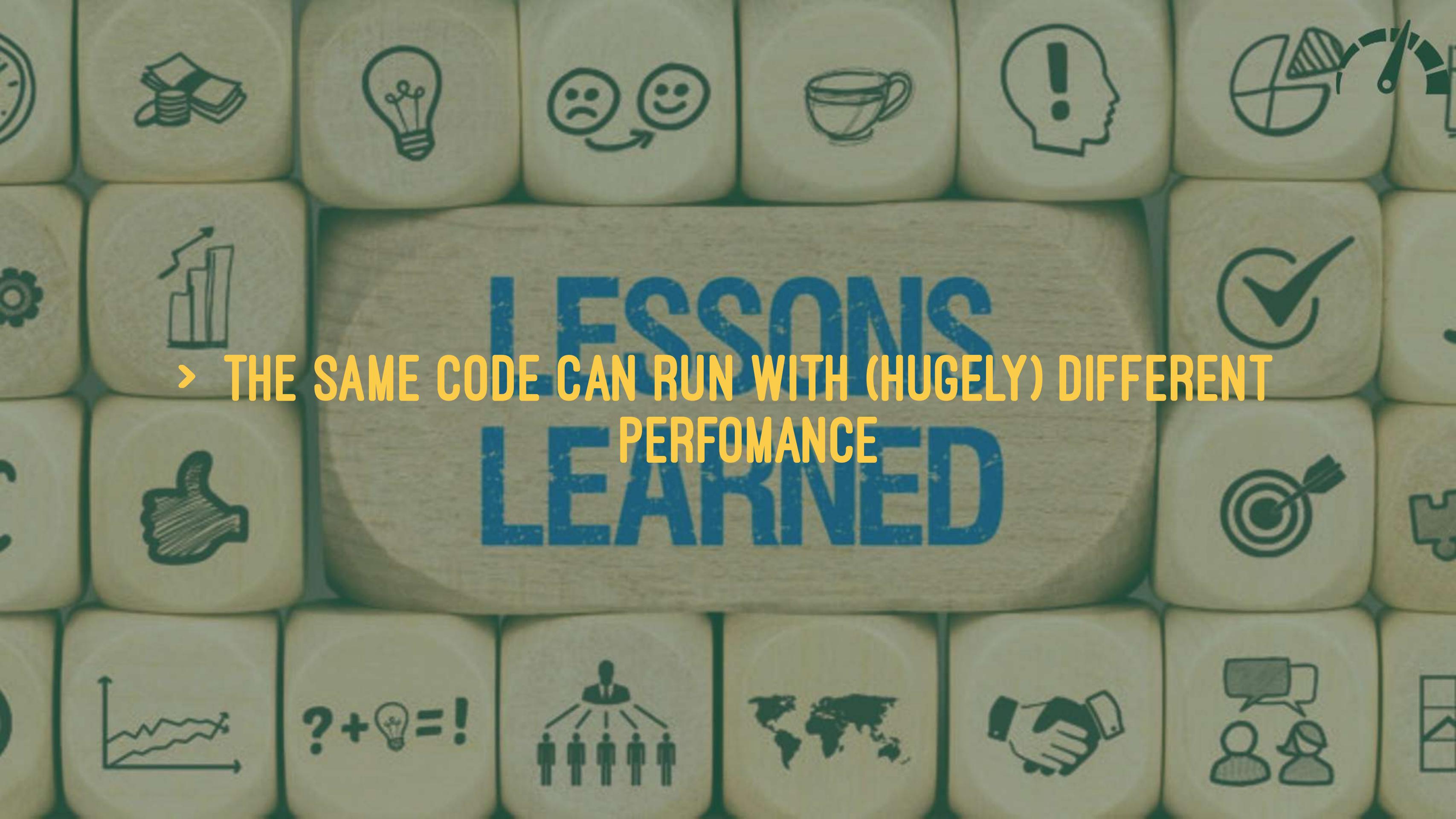
ASYNC TYPESCRIPT (TARGET ES3) $6.772\mu\text{s}$

ASYNC TYPESCRIPT (TARGET ES2018) $2.004\mu\text{s}$

FUNCTIONAL TYPESCRIPT (TARGET ES3) $5.952\mu\text{s}$

FUNCTIONAL TYPESCRIPT (TARGET ES2018) $5.636\mu\text{s}$

LESSONS LEARNED



> THE SAME CODE CAN RUN WITH (HUGELY) DIFFERENT
PERFOMANCE

LESSONS LEARNED

-
- LEARNED
- > THE SAME CODE CAN RUN WITH (HUGELY) DIFFERENT PERFORMANCE
 - > THE COMPILER IS PART OF THE ABSTRACTION IMPLEMENTATION



NEXT STEP

ENFORCE BUSINESS RULES AT COMPILE TIME

CHECKED FUNCTIONAL TYPESCRIPT

```
type NotValid = Left<Error>
type Valid<A> = Right<A>
type Validated<A> = Either<Error, A>

function validationService (o: Order): Validated<Order> {
  const r = validateOrder(order)
  if (r.valid) {
    return valid<Order>(order)
  } else {
    return notvalid(` ${r.error} `)
  }
}
```

CHECKED FUNCTIONAL TYPESCRIPT

```
function calculateAmountService (order: Valid<Order>) {
  return pipe(
    order.right.items.map(item =>
      pipe(
        bookService(item.bookId),
        map(b => b.price * item.quantity)
      )
    ),
    array.sequence(taskEither),
    map(amounts => {
      return amounts.reduce((a, b) => a + b, 0)
    })
  )
}
```

CHECKED FUNCTIONAL TYPESCRIPT

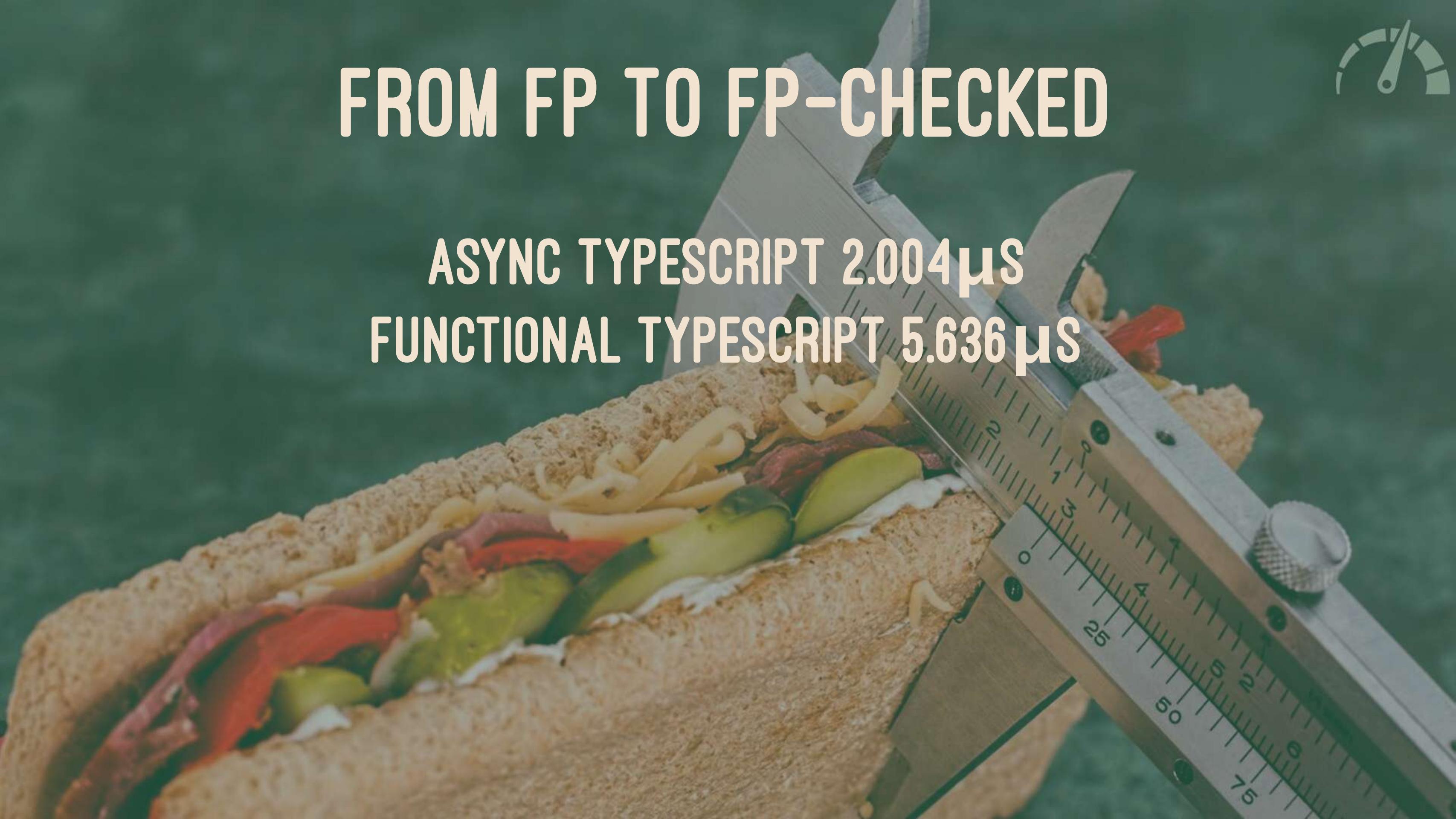
```
return pipe(  
    orderId,  
    orderService,  
    map(validationService),  
    chain(mapTask(placeOrderService))  
)
```



FROM FP TO FP-CHECKED

ASYNC TYPESCRIPT 2.004µS

FUNCTIONAL TYPESCRIPT 5.636µS





FROM FP TO FP-CHECKED

ASYNC TYPESCRIPT 2.004µS

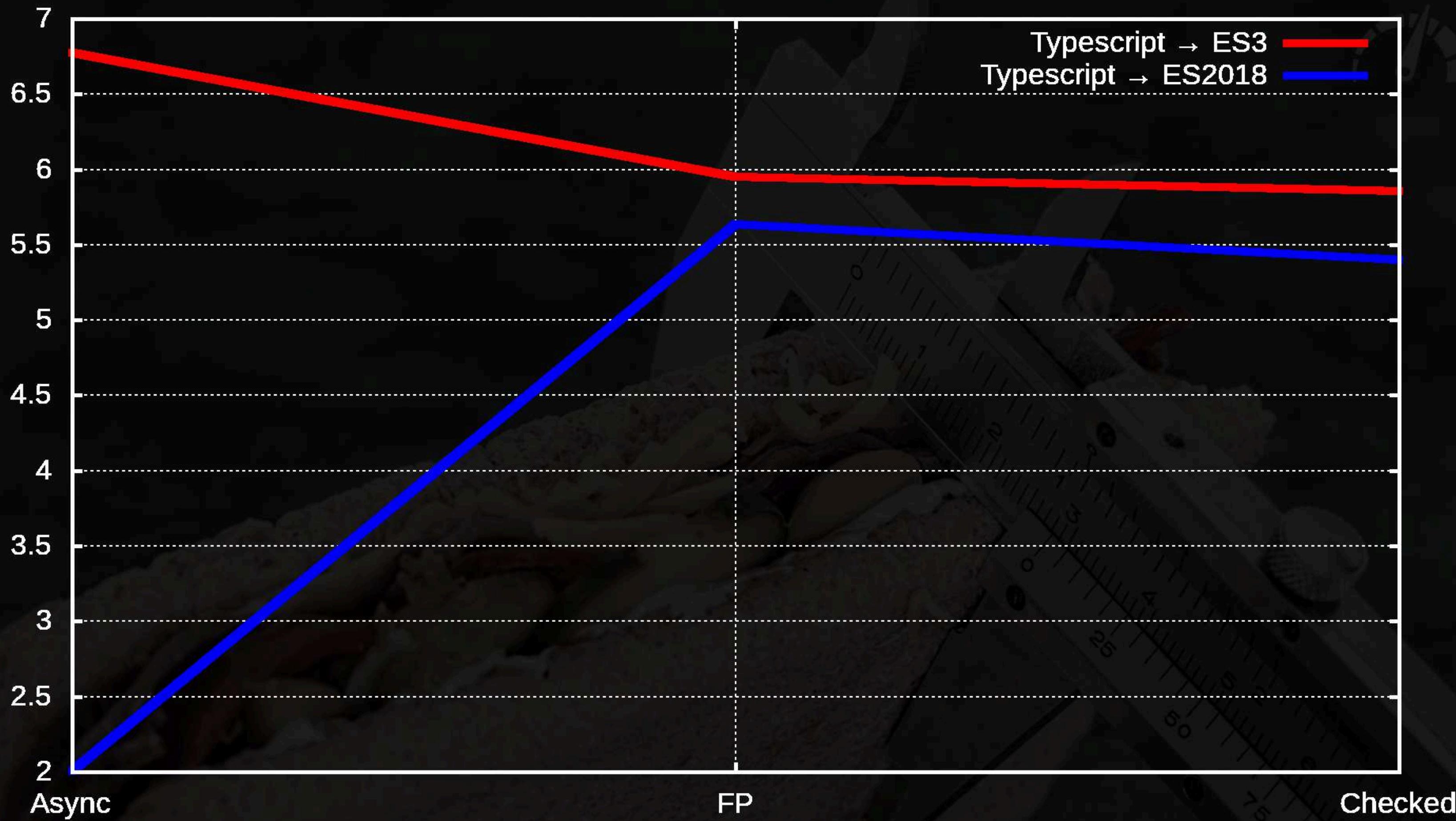
FUNCTIONAL TYPESCRIPT 5.636µS

CHECKED FUNCTIONAL TYPESCRIPT 5.402µS



FROM FP TO FP-CHECKED

- > NO PERFORMANCE PENALTY
- > SLIGHTLY FASTER (ONE LESS chain?)
- > COGNITIVE OVERHEAD



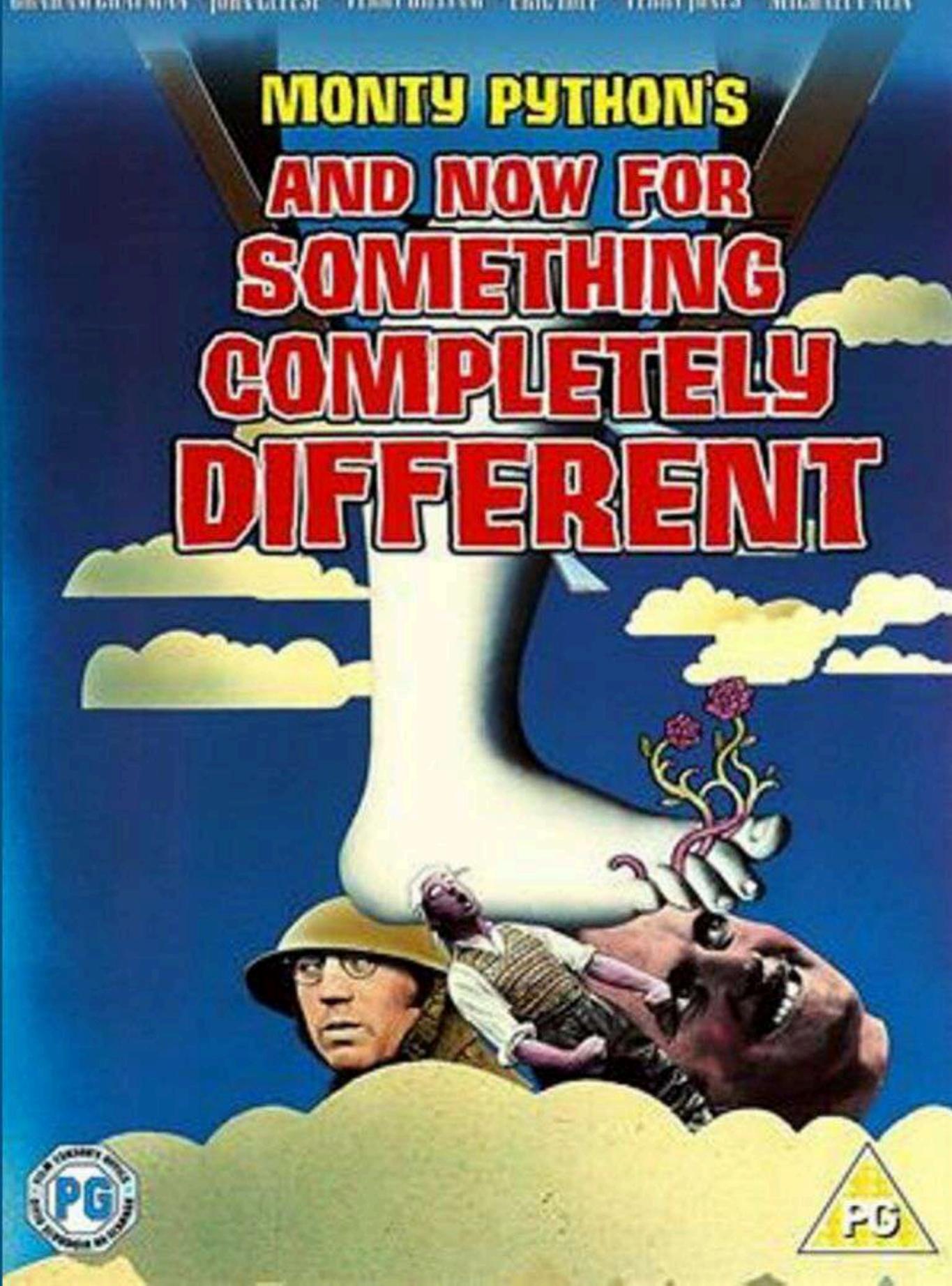


CAN WE HAVE
THE BEST OF
BOTH WORLDS?



GRAHAM CHAPMAN JOHN CLESEY TERRY GILLIAM ERIC IDLE TERRY JONES MICHAEL PALIN

MONTY PYTHON'S
AND NOW FOR
SOMETHING
COMPLETELY
DIFFERENT





NEXT STEP

A DIFFERENT LANGUAGE

The Rust programming language



Jump to: [Rust and the future of systems programming](#) [Unlocking the power of parallel](#)

[Safe systems programming with Rust](#) [Growing the Rust community](#) [Putting Rust into p](#)



☞ Rust is a new open-source systems programming language created by Mozilla and a community of volunteers, designed to help developers create fast, secure applications which take full advantage of the powerful features of modern multi-core processors. It prevents segmentation faults and guarantees thread safety, all with an easy-to-learn syntax.

In addition, Rust offers zero-cost abstractions, move semantics, guaranteed memory safety, threads with no data races, trait-based generics, pattern matching, type inference, and efficient C bindings, with a minimum runtime size.

overhead



To learn more about Rust, you can:

- Watch the videos below for a closer look at the power and benefits Rust provides.
- Read the book ☞ *The Rust Programming Language* online.
- Download the Rust compiler, check out examples, and learn everything you could



WHAT DOES ZERO COST MEAN?

- › ZERO COST FOR THE ABSTRACTIONS YOU DO NOT USE
- › WHAT YOU DO USE, CANNOT BE DONE ANY BETTER



WHAT DOES ZERO COST MEAN?

- › ZERO COST FOR THE ABSTRACTIONS YOU DO NOT USE
- › WHAT YOU DO USE, CANNOT BE DONE ANY BETTER
 - › THIS MEANS 'ZERO' RUNTIME OVERHEAD



WHAT DOES ZERO COST MEAN?

- ZERO COST FOR THE ABSTRACTIONS YOU DO NOT USE
- WHAT YOU DO USE, CANNOT BE DONE ANY BETTER
 - THIS MEANS 'ZERO' RUNTIME OVERHEAD
 - YOU WILL PAY A BUILD TIME COST
 - PLUS COGNITIVE OVERHEAD...



A RUST IMPLEMENTATION

- FAITHFUL TO THE TYPESCRIPT ONE



A RUST IMPLEMENTATION

- > FAITHFUL TO THE TYPESCRIPT ONE
 - > LINE BY LINE ADAPTATION



A RUST IMPLEMENTATION

- > FAITHFUL TO THE TYPESCRIPT ONE
 - > LINE BY LINE ADAPTATION
 - > LET'S BENCHMARK!



FROM TS TO RUST

ASYNC TYPESCRIPT 2.004µS





FROM TS TO RUST

ASYNC TYPESCRIPT 2.004µS
WHAT DO YOU EXPECT?





FROM TS TO RUST

ASYNC TYPESCRIPT $2.004\mu\text{s}$

ASYNC RUST (NATIVE) $0.2410\mu\text{s}$





FROM TS TO RUST

ASYNC TYPESCRIPT 2.004 μ S
ASYNC RUST (NATIVE) 0.2410 μ S
AND 8X SPEEDUP?



WAT?

DOES THIS MAKE SENSE?



WAT?

DOES THIS MAKE SENSE?
IS IT A FAIR COMPARISON?



WAT?

DOES THIS MAKE SENSE?
IS IT A FAIR COMPARISON?
LET'S INVESTIGATE



WAT?

DOES THIS MAKE SENSE?
IS IT A FAIR COMPARISON?
LET'S INVESTIGATE
WE START FROM SCRATCH



WAT?

DOES THIS MAKE SENSE?
IS IT A FAIR COMPARISON?
LET'S INVESTIGATE
WE START FROM SCRATCH
WITH A SYNCHRONOUS TYPESCRIPT VERSION



START FROM SCRATCH

FROM SYNCHRONOUS **TYPESCRIPT VERSION**



START FROM SCRATCH

FROM SYNCHRONOUS TYPESCRIPT VERSION
THEN WE ADD MINIMAL ABSTRACTIONS



START FROM SCRATCH

FROM SYNCHRONOUS TYPESCRIPT VERSION
THEN WE ADD MINIMAL ABSTRACTIONS
ONE BY ONE



START FROM SCRATCH

FROM SYNCHRONOUS TYPESCRIPT VERSION
THEN WE ADD MINIMAL ABSTRACTIONS
ONE BY ONE
AND BENCHMARK EACH STEP



ABSTRACTION STEPS



ABSTRACTION STEPS

SYNCHRONOUS **TYPESCRIPT**



ABSTRACTION STEPS

SYNCHRONOUS **TYPESCRIPT**
SYNCHRONOUS FUNCTIONAL **TYPESCRIPT**



ABSTRACTION STEPS

SYNCHRONOUS **TYPESCRIPT**

SYNCHRONOUS FUNCTIONAL **TYPESCRIPT**

SYNCHRONOUS *CALLBACKS*



ABSTRACTION STEPS

SYNCHRONOUS TYPESCRIPT

SYNCHRONOUS FUNCTIONAL TYPESCRIPT

SYNCHRONOUS CALLBACKS

NEXTTICK CALLBACKS



ABSTRACTION STEPS

SYNCHRONOUS TYPESCRIPT

SYNCHRONOUS FUNCTIONAL TYPESCRIPT

SYNCHRONOUS CALLBACKS

NEXTTICK CALLBACKS

PROMISES



ABSTRACTION STEPS

SYNCHRONOUS TYPESCRIPT

SYNCHRONOUS FUNCTIONAL TYPESCRIPT

SYNCHRONOUS CALLBACKS

NEXTTICK CALLBACKS

PROMISES

ASYNC AWAIT



ABSTRACTION STEPS

SYNCHRONOUS **TYPESCRIPT**

SYNCHRONOUS FUNCTIONAL **TYPESCRIPT**

SYNCHRONOUS CALLBACKS

NEXTTICK CALLBACKS

PROMISES

ASYNC AWAIT

FUNCTIONAL **TASKS**



ABSTRACTION STEPS

SYNCHRONOUS **TYPESCRIPT**

SYNCHRONOUS FUNCTIONAL **TYPESCRIPT**

SYNCHRONOUS CALLBACKS

NEXTTICK CALLBACKS

PROMISES

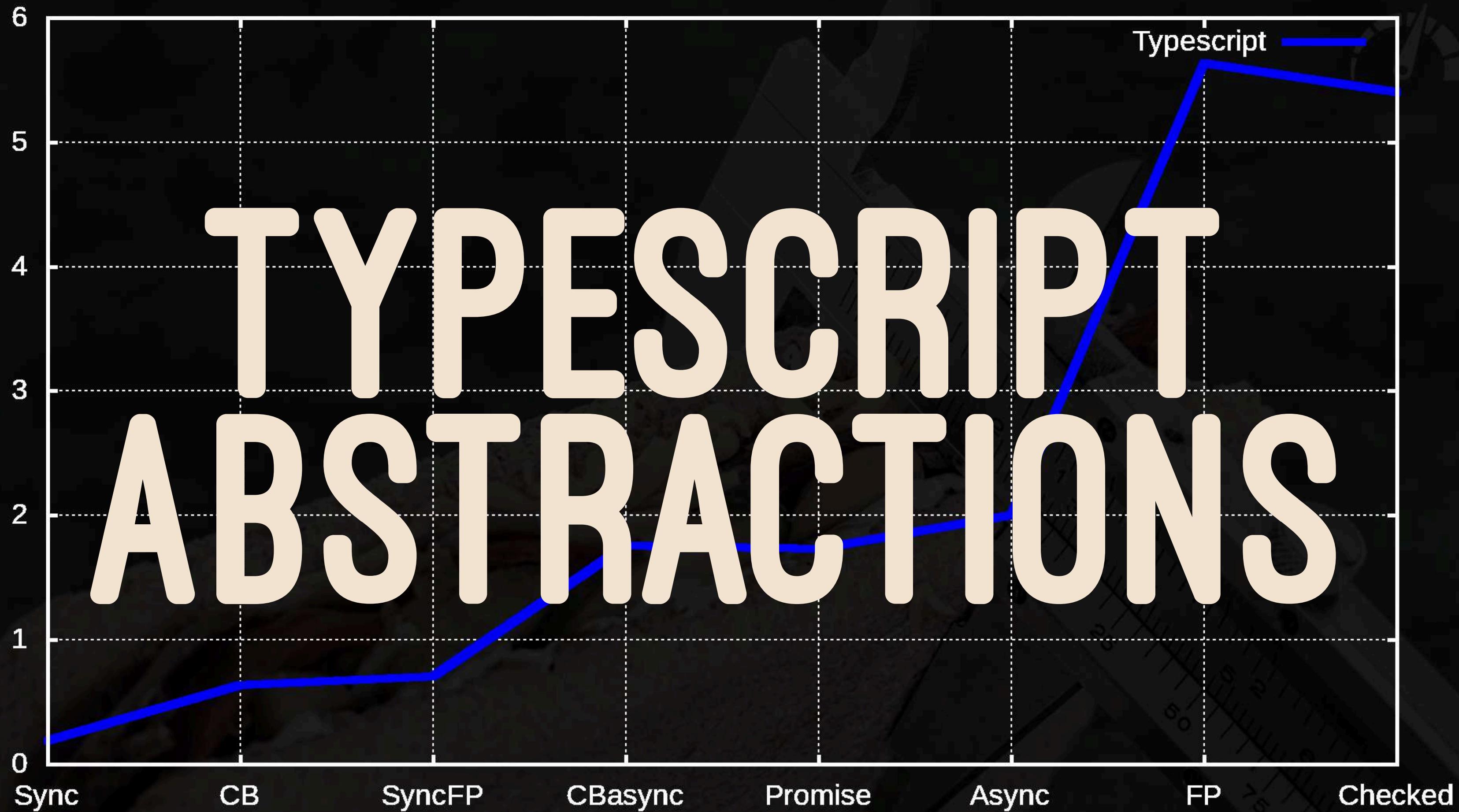
ASYNC AWAIT

FUNCTIONAL **TASKS**

CHECKED **TYPES**

TypeScript

TYPESCRIPT ABSTRACTIONS

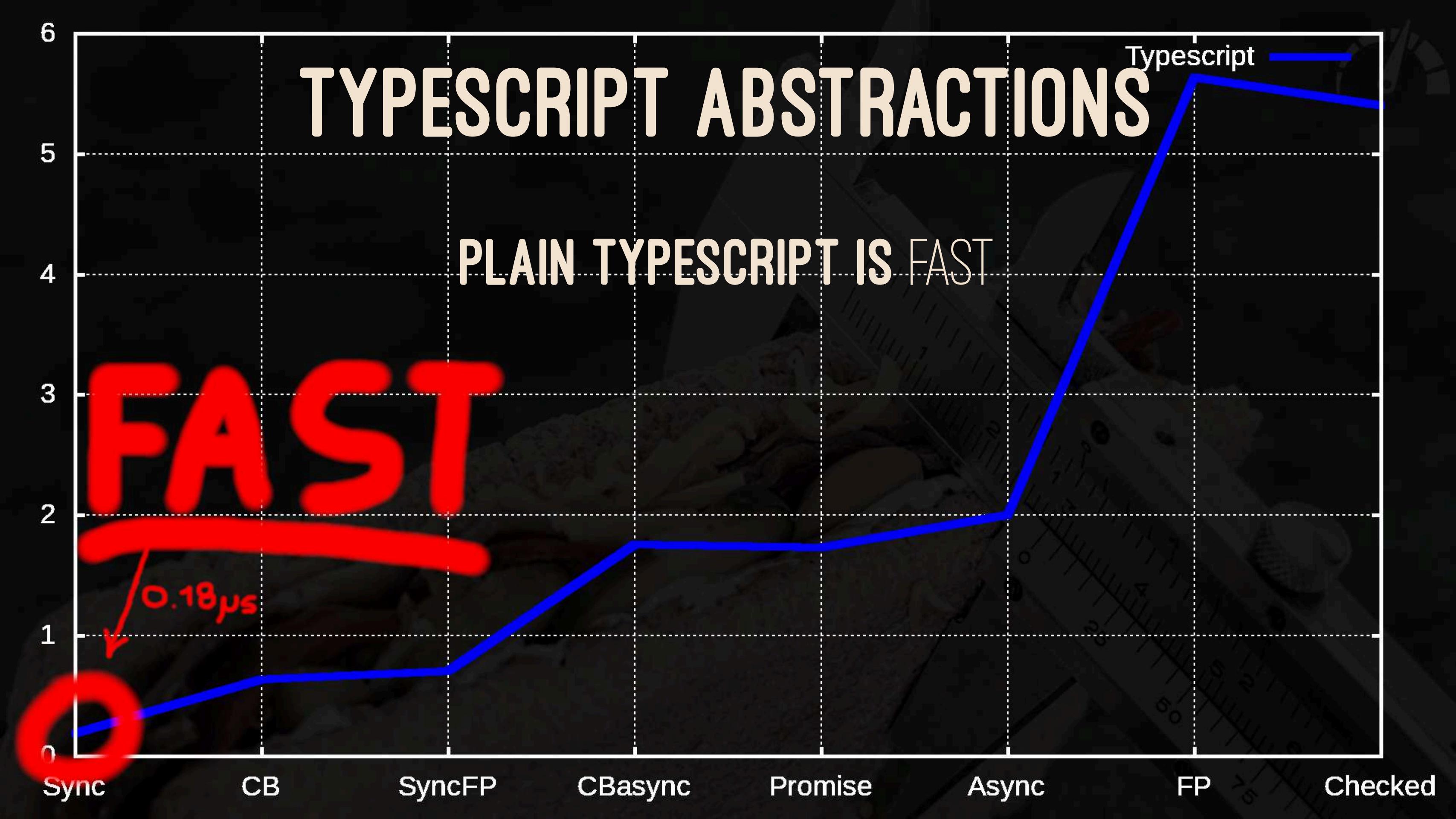


TYPESCRIPT ABSTRACTIONS

PLAIN TYPESCRIPT IS FAST

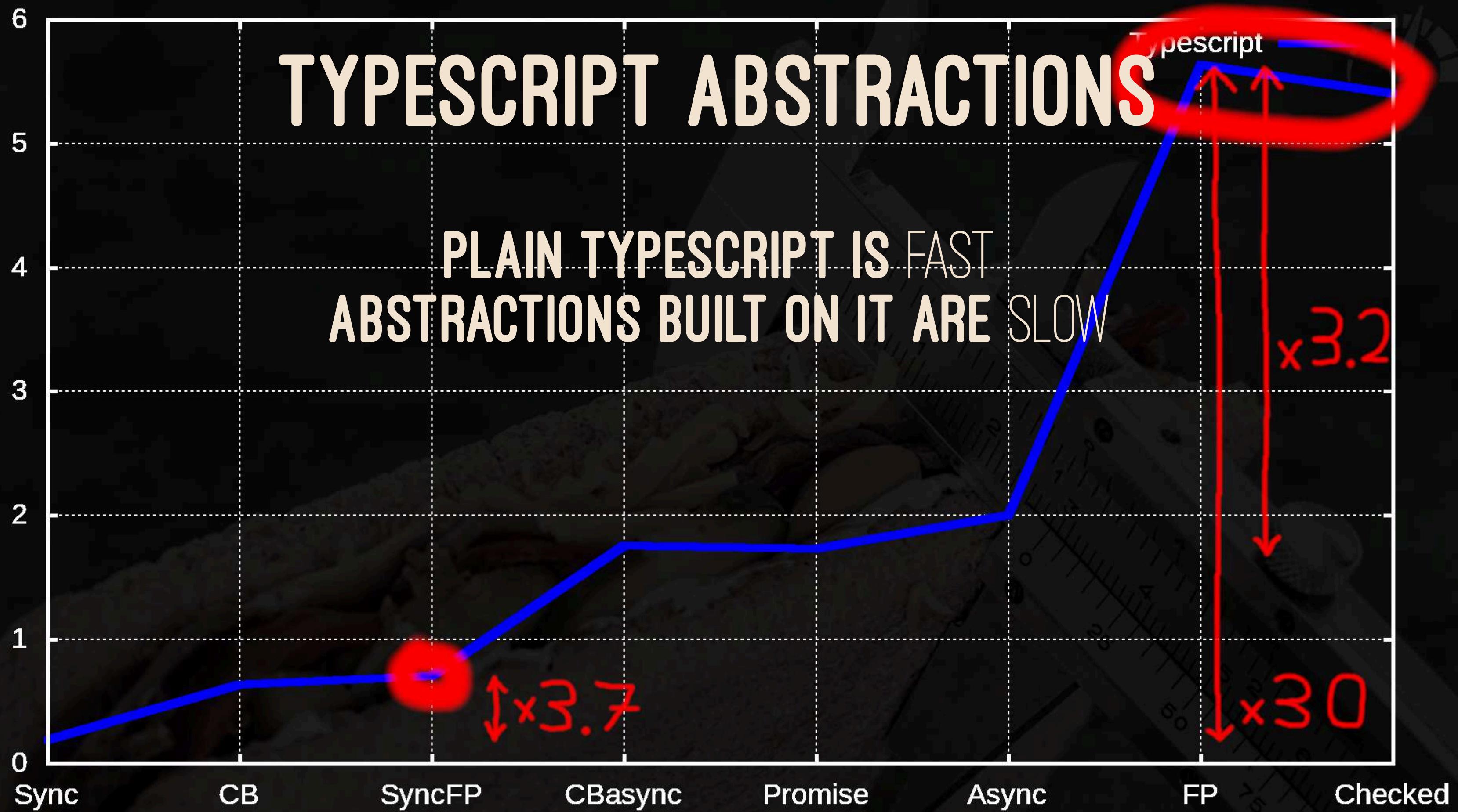
FAST

0.18 μ s



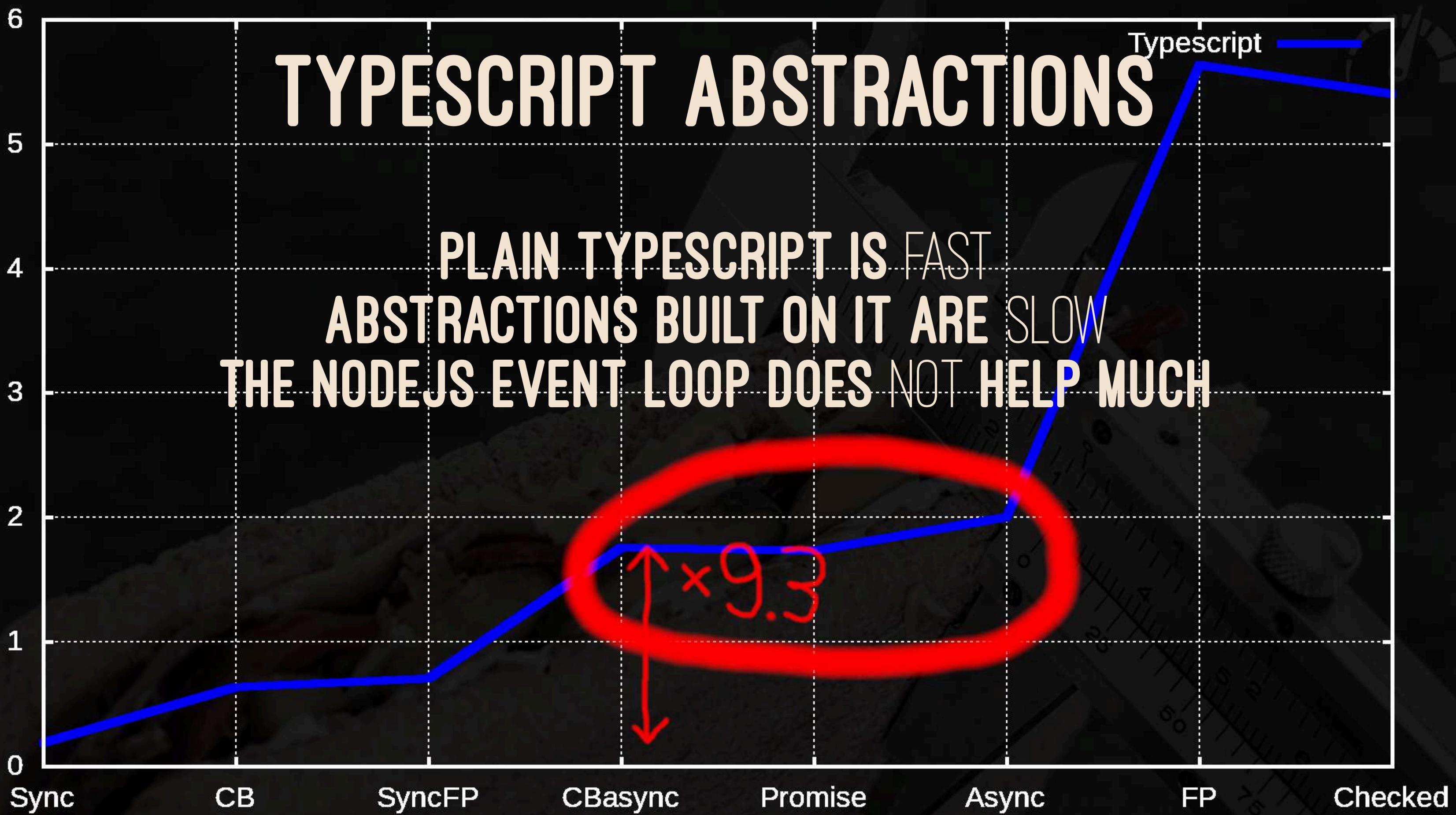
TYPESCRIPT ABSTRACTIONS

PLAIN TYPESCRIPT IS FAST
ABSTRACTIONS BUILT ON IT ARE SLOW



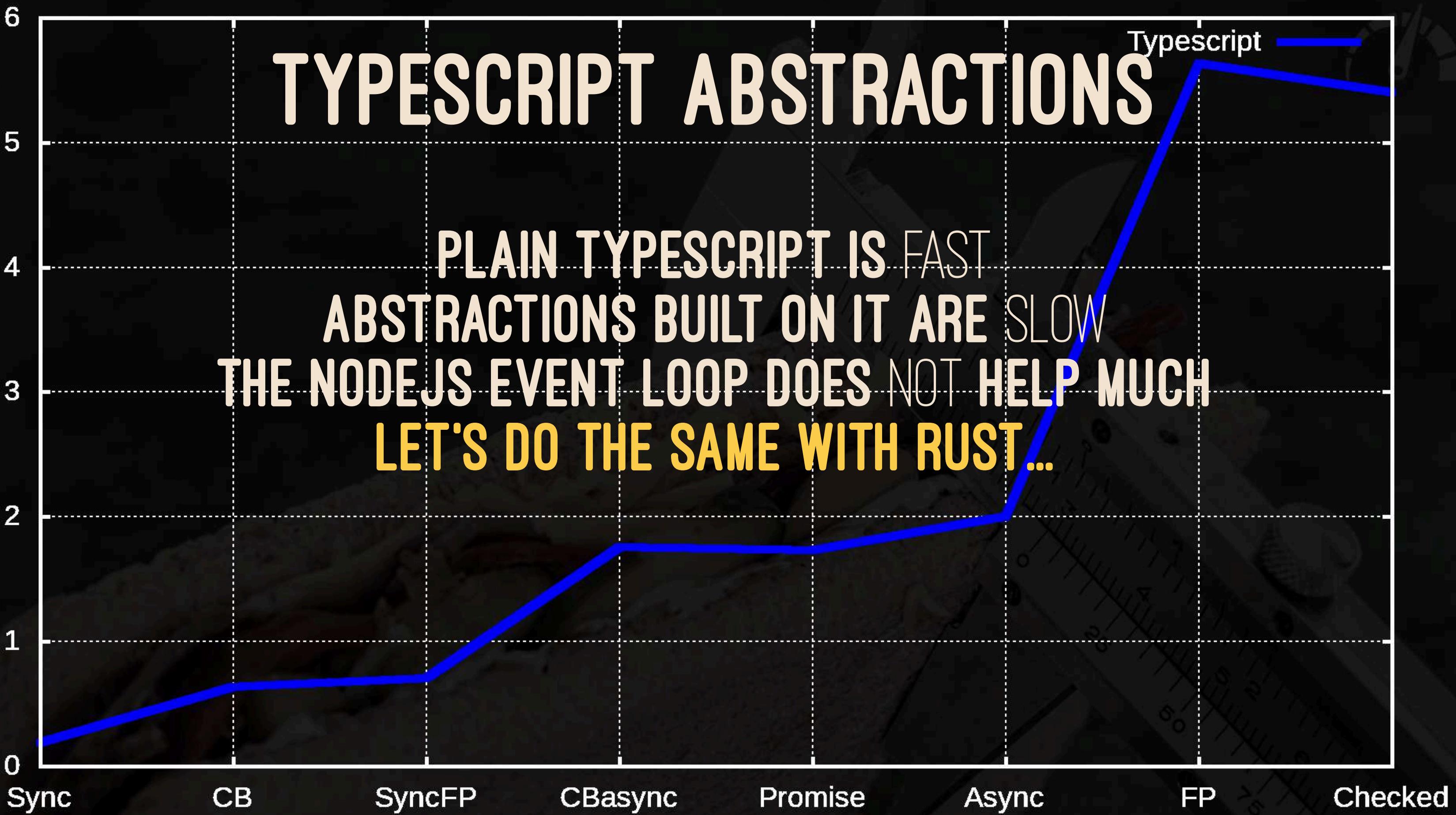
TYPESCRIPT ABSTRACTIONS

PLAIN TYPESCRIPT IS FAST
ABSTRACTIONS BUILT ON IT ARE SLOW
THE NODEJS EVENT LOOP DOES NOT HELP MUCH



TYPESCRIPT ABSTRACTIONS

PLAIN TYPESCRIPT IS FAST
ABSTRACTIONS BUILT ON IT ARE SLOW
THE NODEJS EVENT LOOP DOES NOT HELP MUCH
LET'S DO THE SAME WITH RUST...



RUST. PLAIN

```
pub async fn process(order_id: &String) -> Result<f64, ()> {
    match order_service(order_id).await {
        Some(order) => {
            let validation = validation_service(&order).await;
            match validation {
                Ok(order) => match place_order_service(order).await {
                    Ok(res) => Ok(res.amount),
                    Err(_) => Err(()),
                },
                _ => Err(()),
            }
        }
        _ => Err(()),
    }
}
```

RUST. IDIOMATIC

```
pub async fn process(order_id: &String) -> Result<f64, ()> {  
    let order = order_service(order_id).await;  
    let validated = validation_service(order).await.map_err(|_| ())?;  
    Ok(place_order_service(validated).await.map_err(|_| ())?.amount)  
}
```

RUST. COMPOSABLE

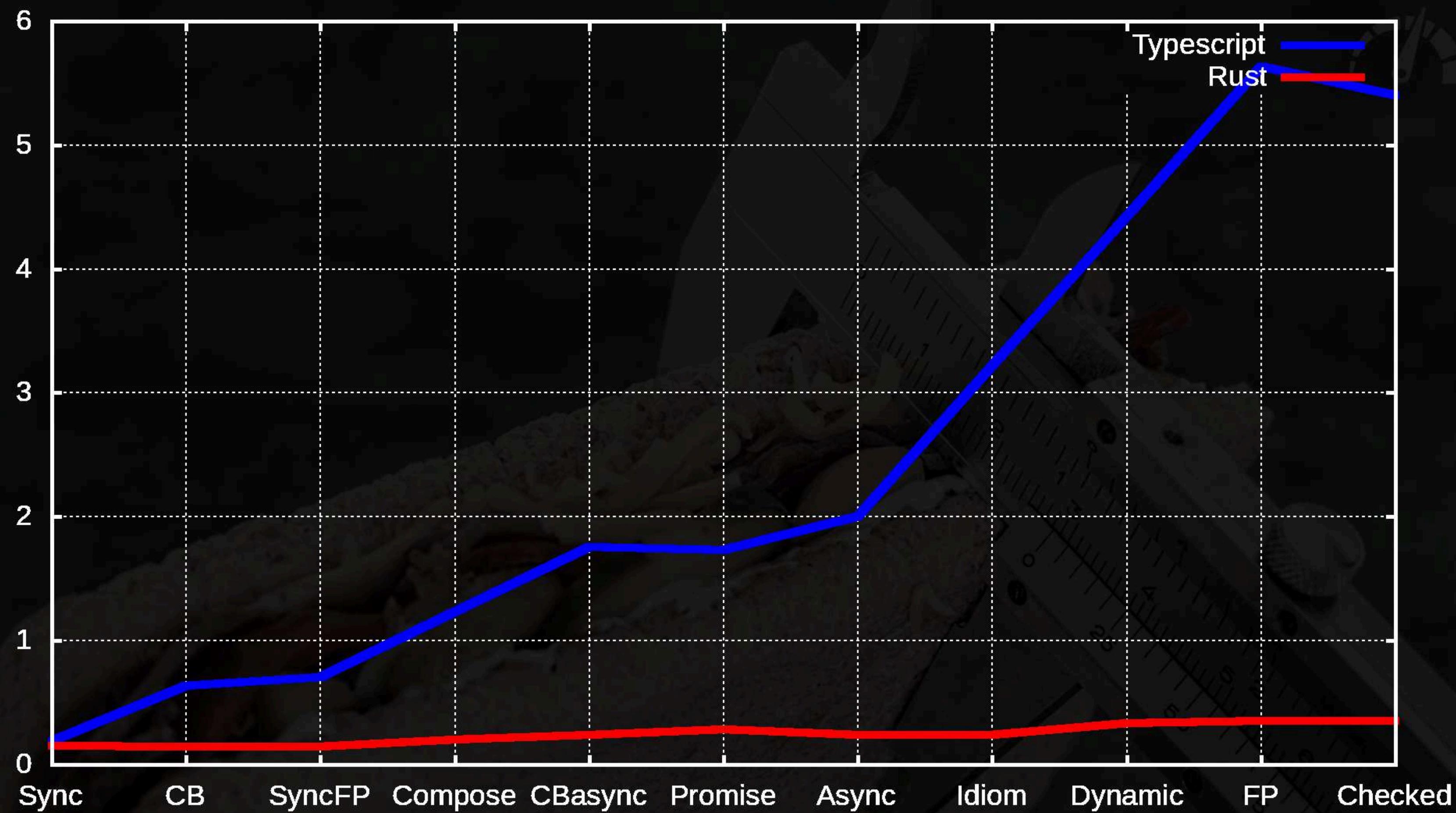
```
pub fn process(order_id: &'static String) ->
    impl Future<Output = Result<f64, ()>> {
compose!(
    order_service,
    validation_service,
    place_order_service,
    map_order_amount
)(order_id)
}
```



WE HAVE SEEN RUST...

SHOULD WE MEASURE IT?





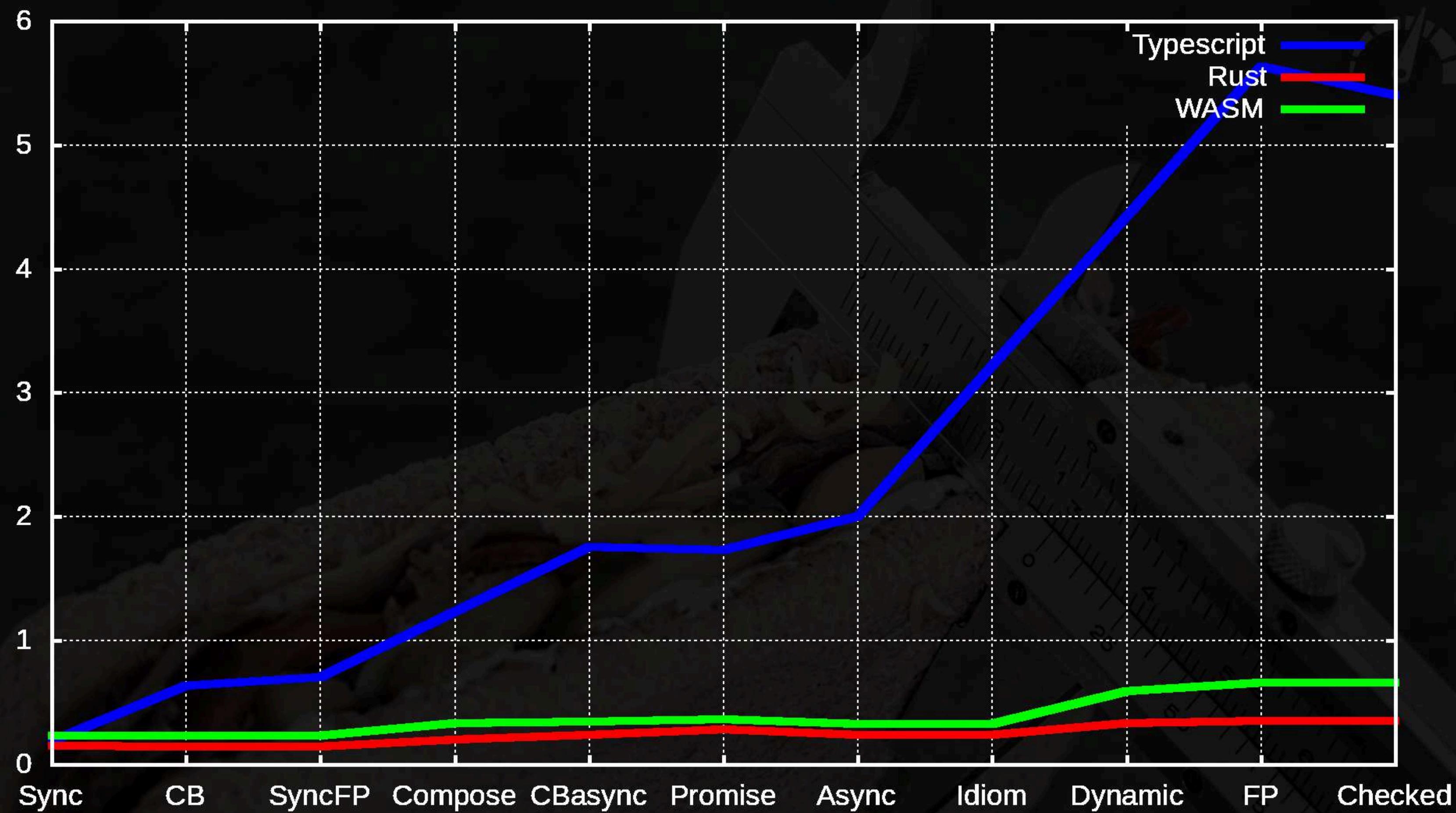


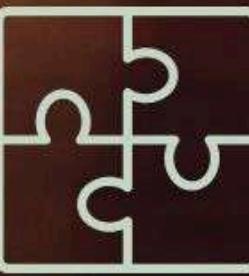
WHAT ABOUT THE WEB?



NEXT STEP

SAME RUST CODE RUNNING ON WEBASSEMBLY





THOUGHT CATALOG





THOUGHT CATALOG

› WE'RE FOCUSING ON BETTER SOFTWARE



- WE'RE FOCUSING ON BETTER SOFTWARE
- BUT WHAT DOES 'BETTER' SOFTWARE MEAN?



THOUGHT CATALOG

profit = revenue – cost



*profit = (revenue * time2market) - cost*

WHERE

$[0 >= time2market <= 1]$



$profit(t) = (revenue(t) * time2market(t)) - cost(t)$

WHERE

$[0 >= time2market <= 1]$



- › WE'RE FOCUSING ON BETTER SOFTWARE
- › BUT WHAT DOES 'BETTER' SOFTWARE MEAN?
- › " *better* " = $\max(\text{profit}(t))$



- > WE'RE FOCUSING ON BETTER SOFTWARE
- > BUT WHAT DOES 'BETTER' SOFTWARE MEAN?
- > " *better* " = $\max(\text{profit}(t))$
- > HOW DO WE DO THAT?

1P

34200

ONI

50000

2P

3600



Evil Ryu

K.O

82

Violent Ken



PERFORMANCE VS MAINTAINABILITY



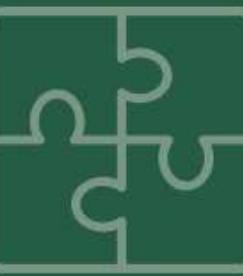


DESIGN ABSTRACTION

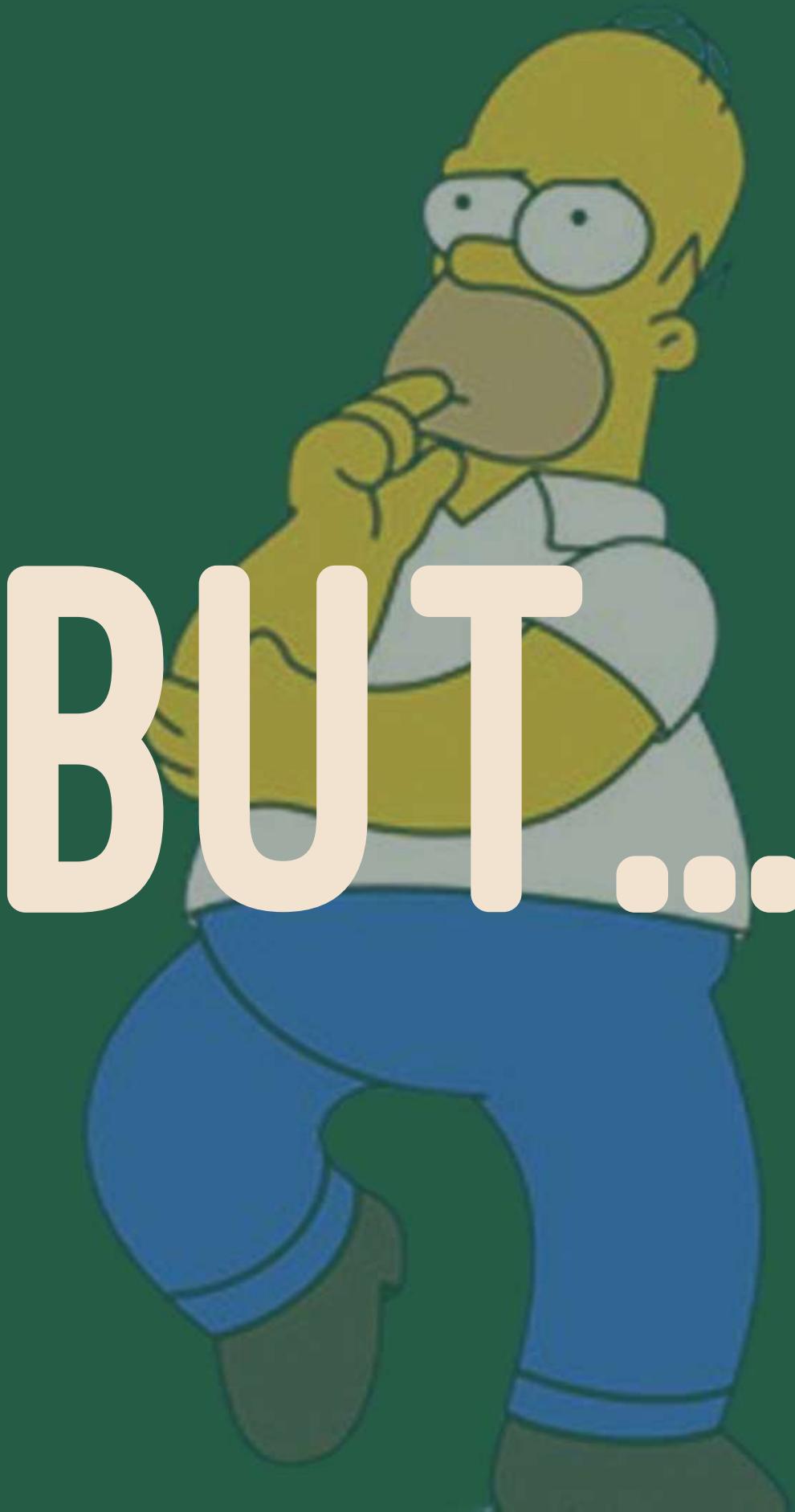
- PERFORMANCE
- MAINTAINABILITY

CODE OPTIMISATION ABSTRACTION

-  PERFORMANCE
-  MAINTAINABILITY



BUT...





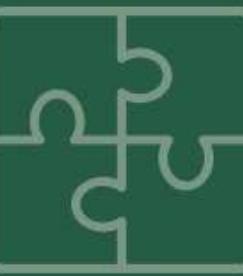
WE KNOW HOW TO ACHIEVE
NO PERFORMANCE COST
AND STILL HAVE MAINTAINABILITY...



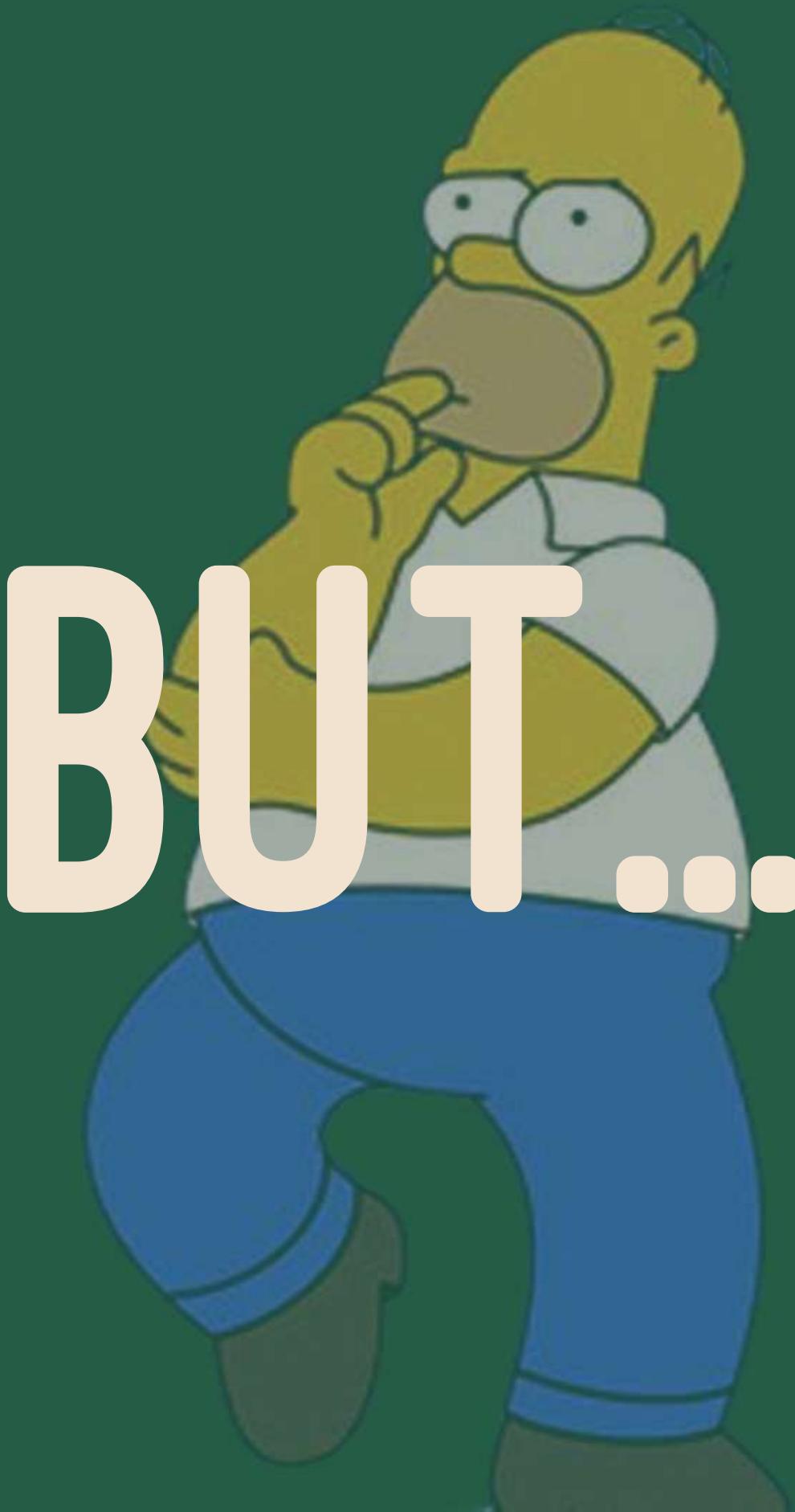


RUST IS THE ANSWER!

ZERO COST OVERHEAD ABSTRACTION

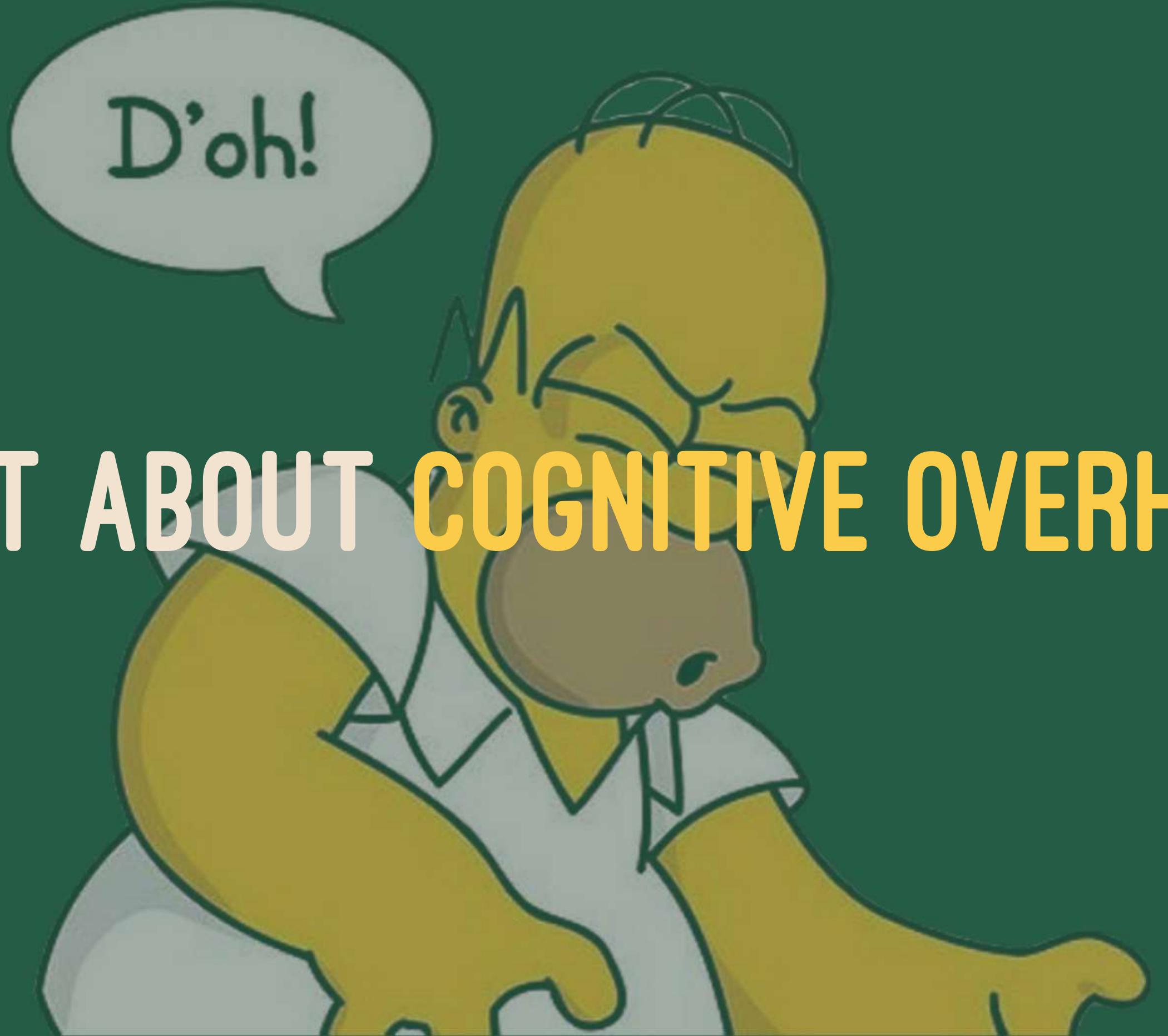


BUT...





WHAT ABOUT COGNITIVE OVERHEAD?



A close-up photograph of a sleek, dark green fountain pen lying diagonally across a white spiral-bound notebook. The notebook has 'FLIGHT LOG' printed in green capital letters at the top. The pen's cap is off, showing its internal mechanism and a small clip. The lighting is soft, creating a warm glow on the pen and the notebook.

WRAPPING UP

THERE ARE DIFFERENT KINDS OF ABSTRACTIONS

WRAPPING UP

THERE ARE DIFFERENT KINDS OF ABSTRACTIONS
AND THERE ARE DIFFERENT KINDS OF COSTS

WRAPPING UP

THERE ARE DIFFERENT KINDS OF ABSTRACTIONS
AND THERE ARE DIFFERENT KINDS OF COSTS
DIFFERENT ABSTRACTIONS INVOLVE DIFFERENT COSTS

WRAPPING UP

THERE ARE DIFFERENT KINDS OF ABSTRACTIONS
AND THERE ARE DIFFERENT KINDS OF COSTS
DIFFERENT ABSTRACTIONS INVOLVE DIFFERENT COSTS
THERE ARE NO ZERO COST ABSTRACTIONS. BUT...

WRAPPING UP

THERE ARE DIFFERENT KINDS OF ABSTRACTIONS
AND THERE ARE DIFFERENT KINDS OF COSTS
DIFFERENT ABSTRACTIONS INVOLVE DIFFERENT COSTS
THERE ARE NO ZERO COST ABSTRACTIONS. BUT...
...WE CAN CHOOSE WHERE TO INCUR COSTS!

WRAPPING UP

THERE ARE DIFFERENT KINDS OF ABSTRACTIONS
AND THERE ARE DIFFERENT KINDS OF COSTS
DIFFERENT ABSTRACTIONS INVOLVE DIFFERENT COSTS
THERE ARE NO ZERO COST ABSTRACTIONS. BUT...
...WE CAN CHOOSE WHERE TO INCUR COSTS!
HOW DO WE CHOOSE?

ABSTRACTION SPACE

TypeScript

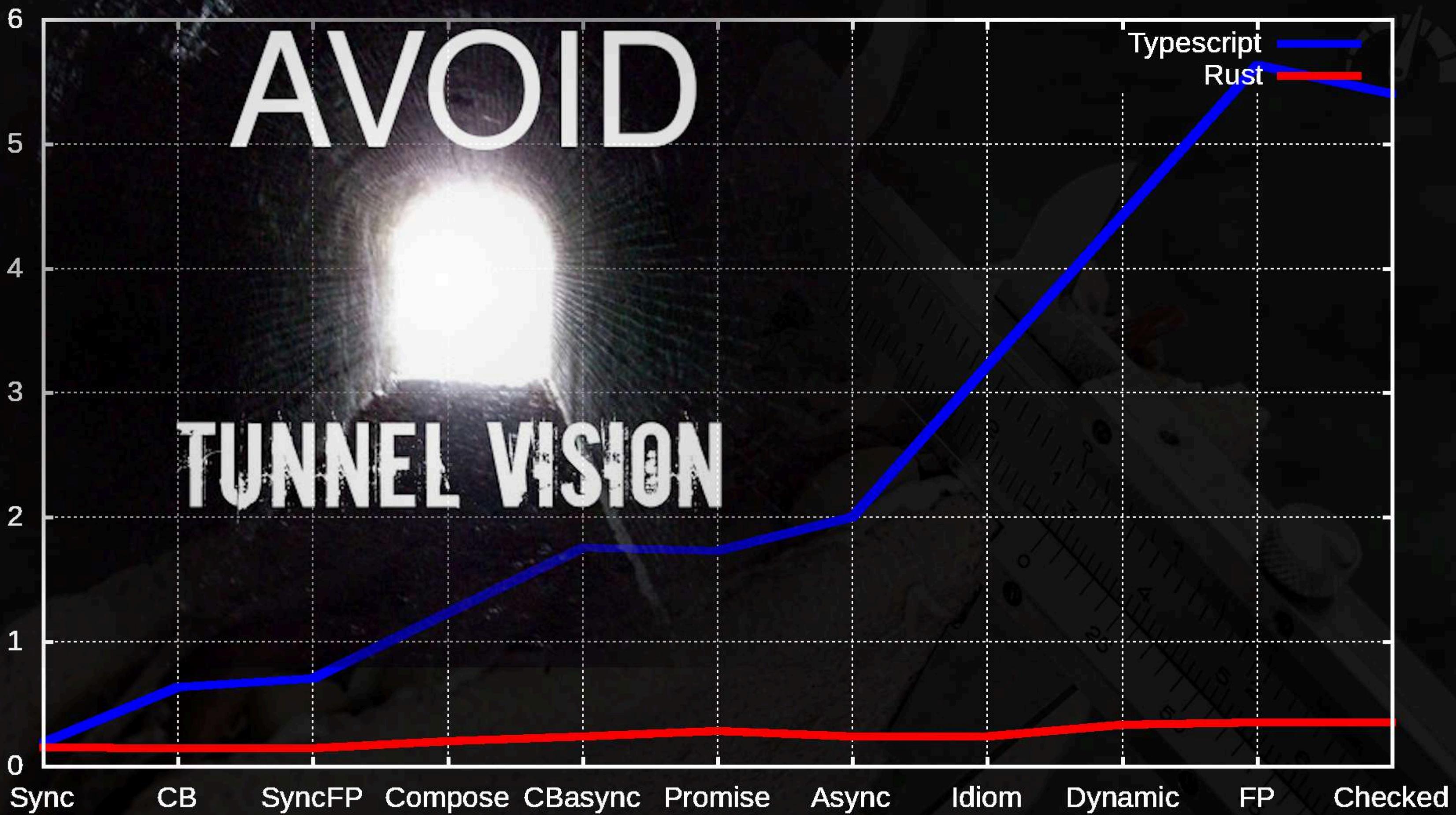
RD



AVOID

TUNNEL VISION

TypeScript
Rust



6

5

4

3

2

1

0

AVOID

TypeScript

TUNNEL VISION

LANGUAGE BIAS

Sync

CB

SyncFP

Compose

CBasync

Promise

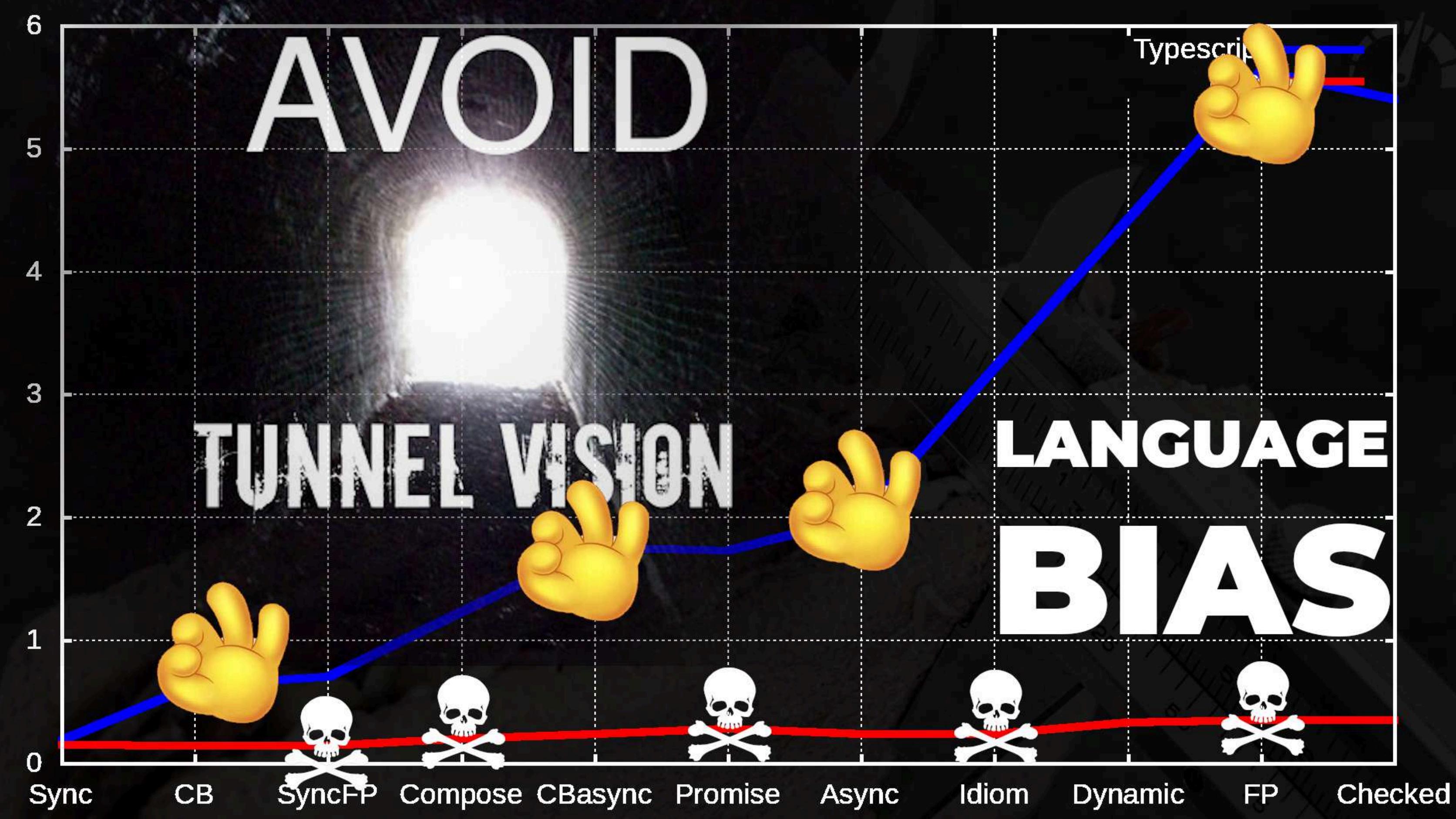
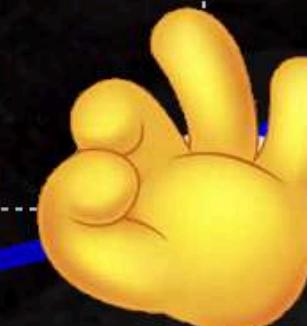
Async

Idiom

Dynamic

FP

Checked



6

5

4

3

2

1

0

AVOID

TUNNEL VISION

STYLE BIAS

Sync

CB

SyncFP

Compose

CBasync

Promise

Async

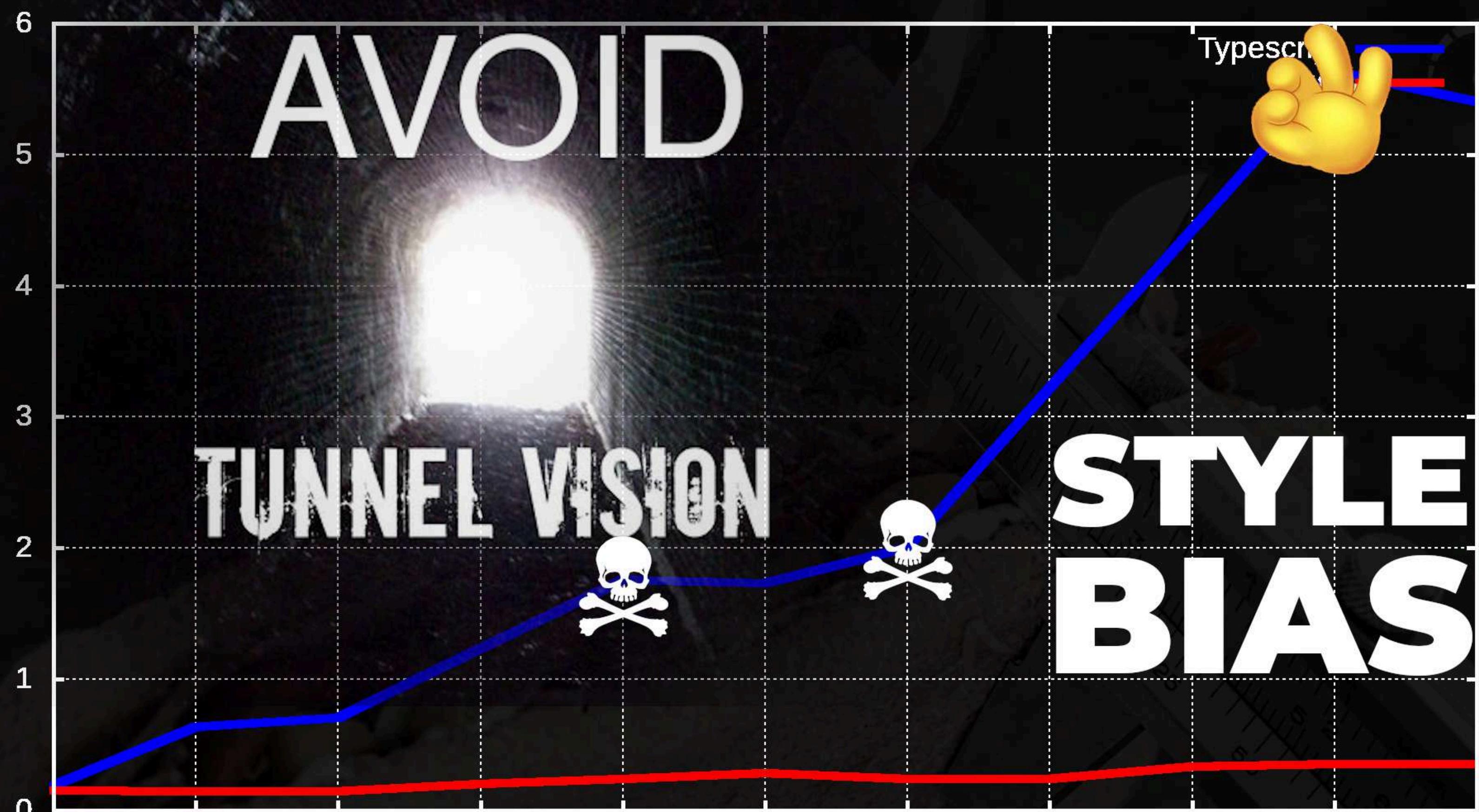
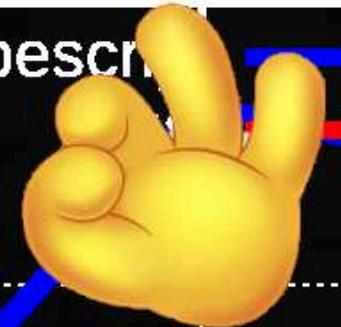
Idiom

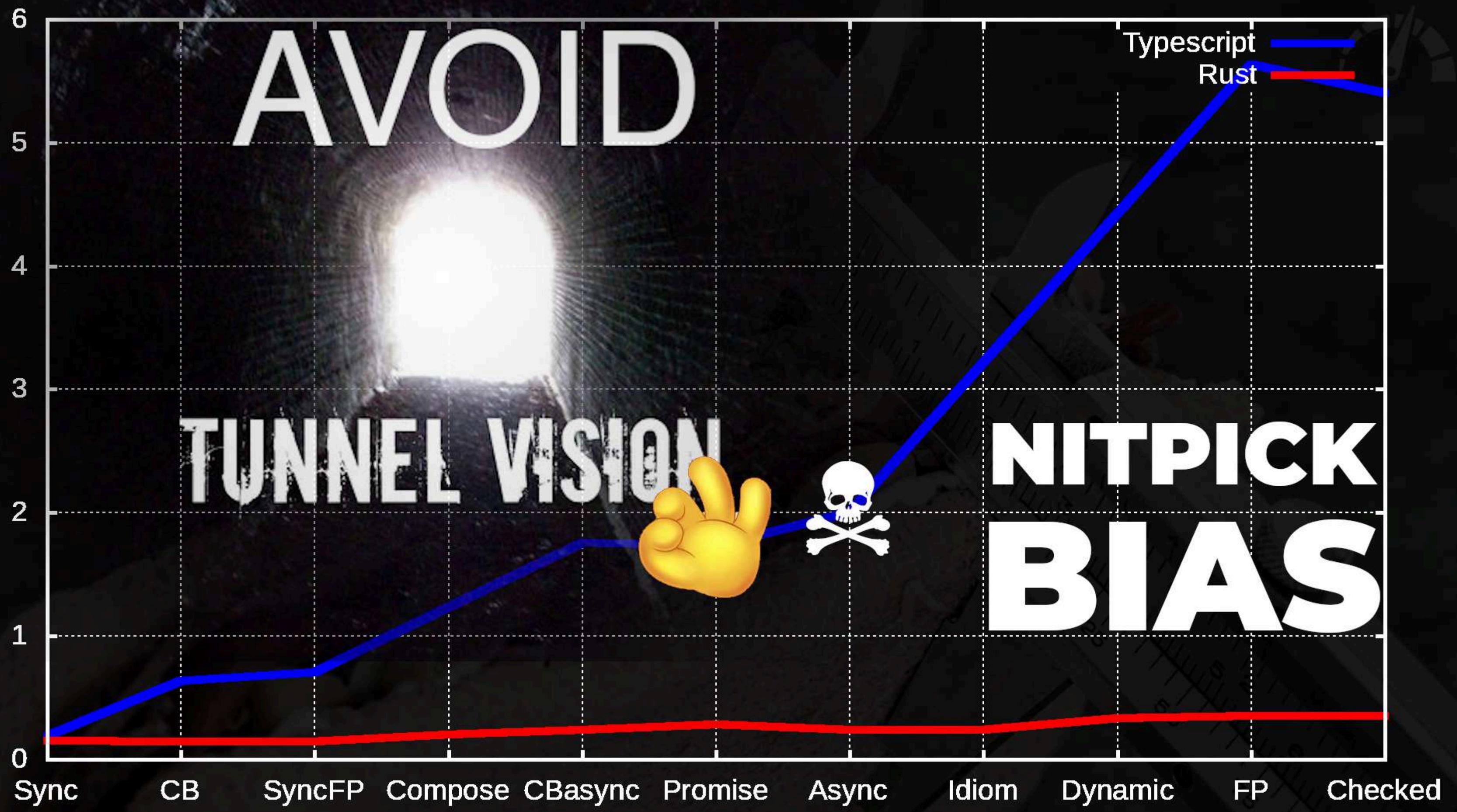
Dynamic

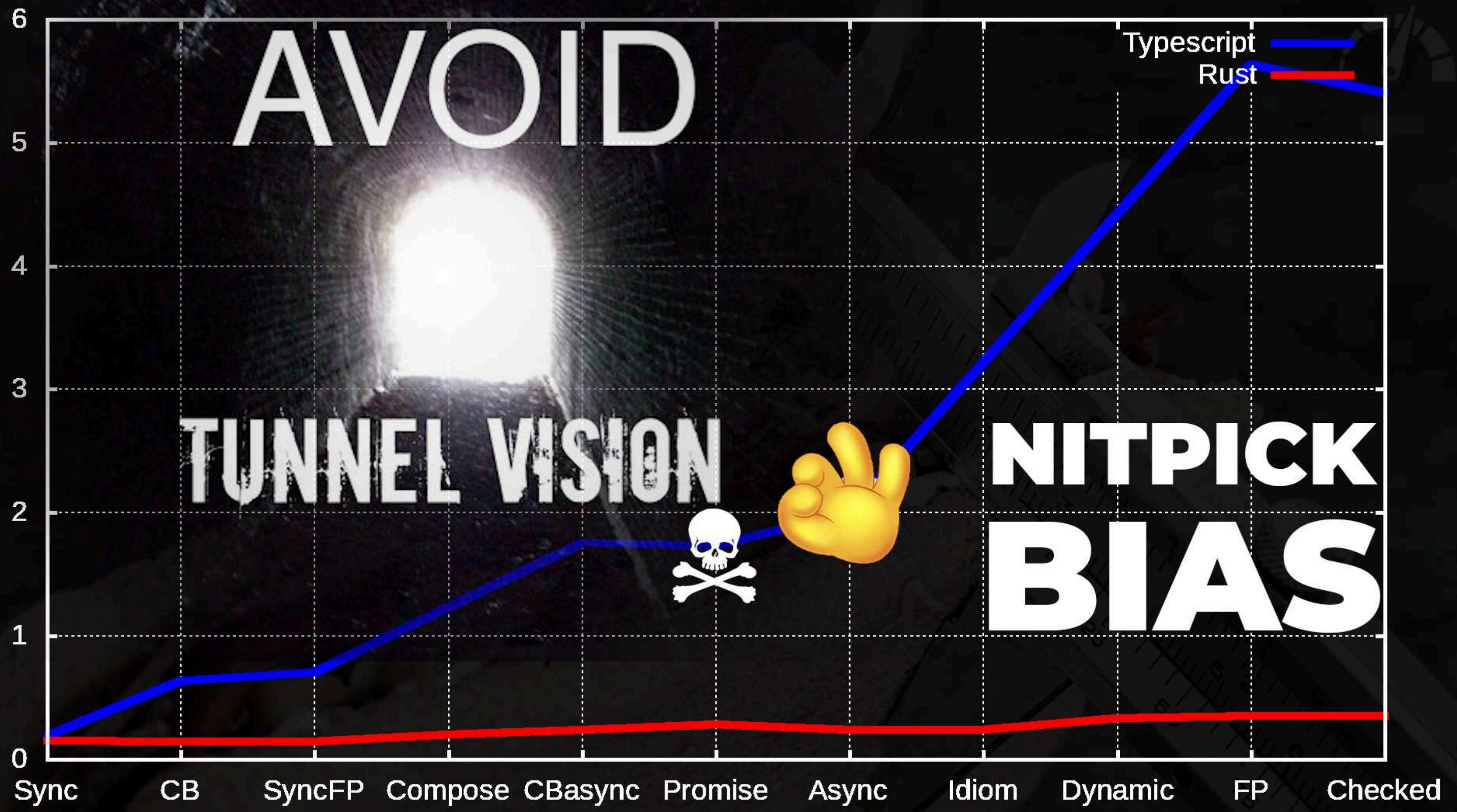
FP

Checked

Typescript

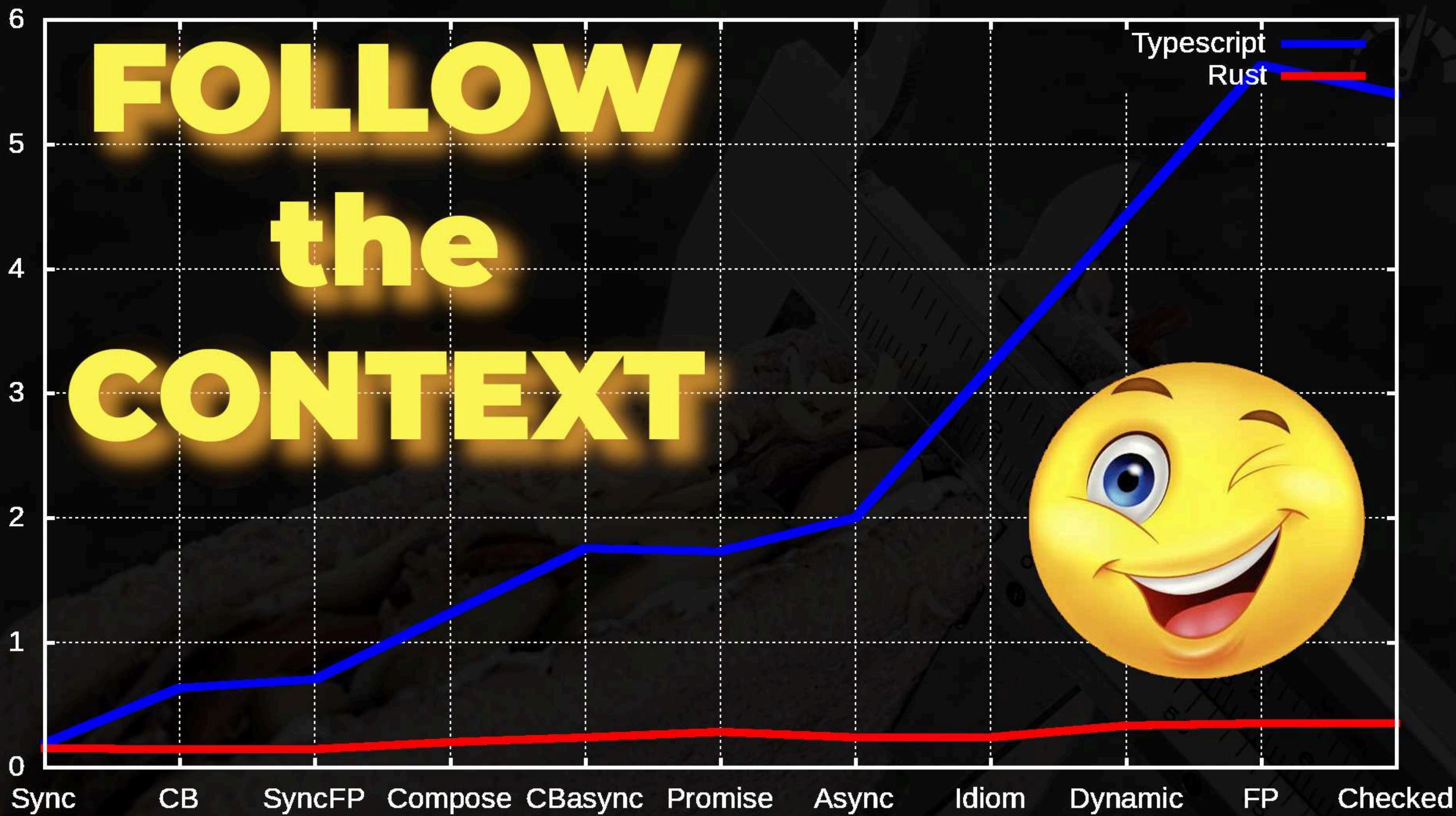






FOLLOW the CONTEXT

TypeScript
Rust



TypeScript

**FOLLOW
the
CONTEXT**

6

5

4

3

2

1

0

Sync

CB

SyncFP

Compose

CBasync

Promise

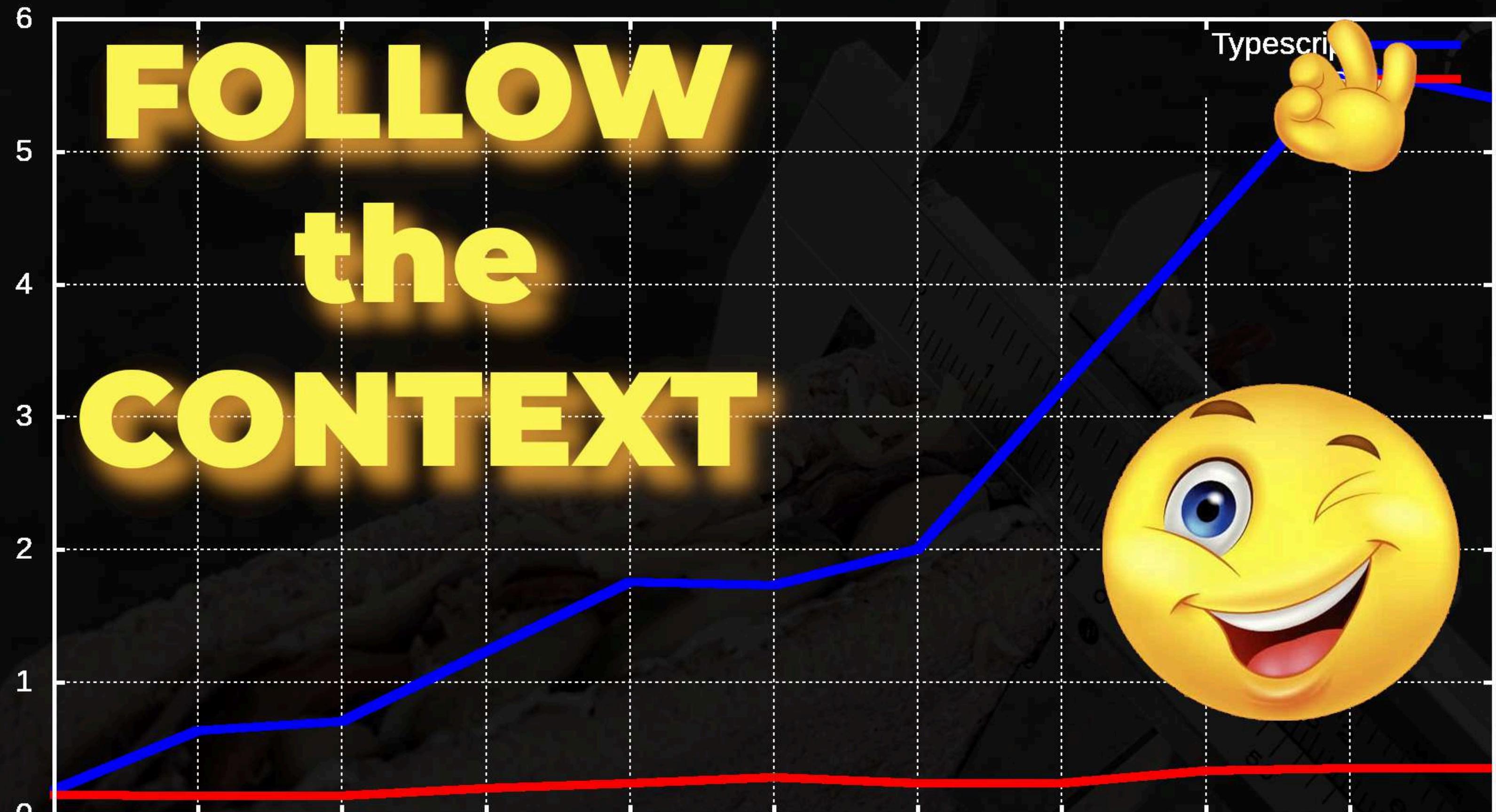
Async

Idiom

Dynamic

FP

Checked



TypeScript

FOLLOW the CONTEXT

6

5

4

3

2

1

0

Sync

CB

SyncFP

Compose

CBasync

Promise

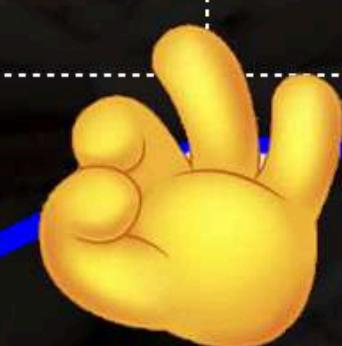
Async

Idiom

Dynamic

FP

Checked



TypeScript

FOLLOW the CONTEXT

6

5

4

3

2

1

0

Sync

CB

SyncFP

Compose

CBasync

Promise

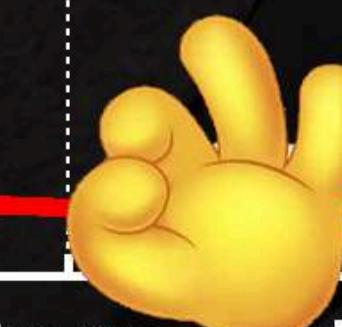
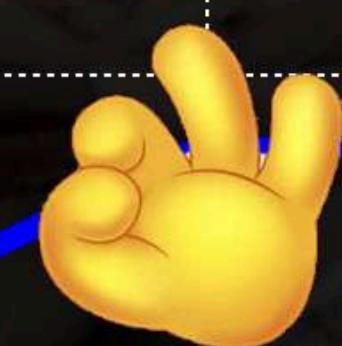
Async

Idiom

Dynamic

FP

Checked



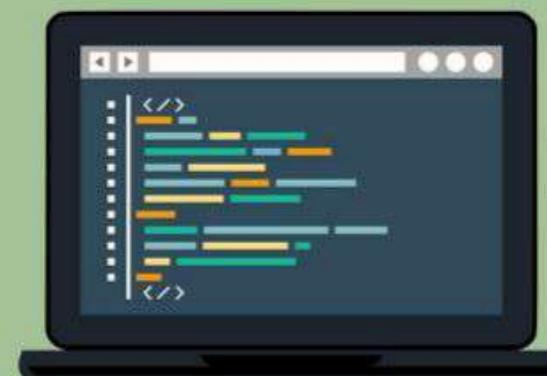
Thank
you!



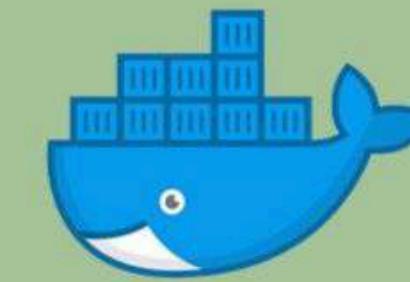
QUESTIONS?



slides



code



environment