Why the new V8 is so Damn Fast

One Demo at a Time

@thlorenz

NODESOURCE®

Why the new V8 is so Damn Fast

@thlorenz

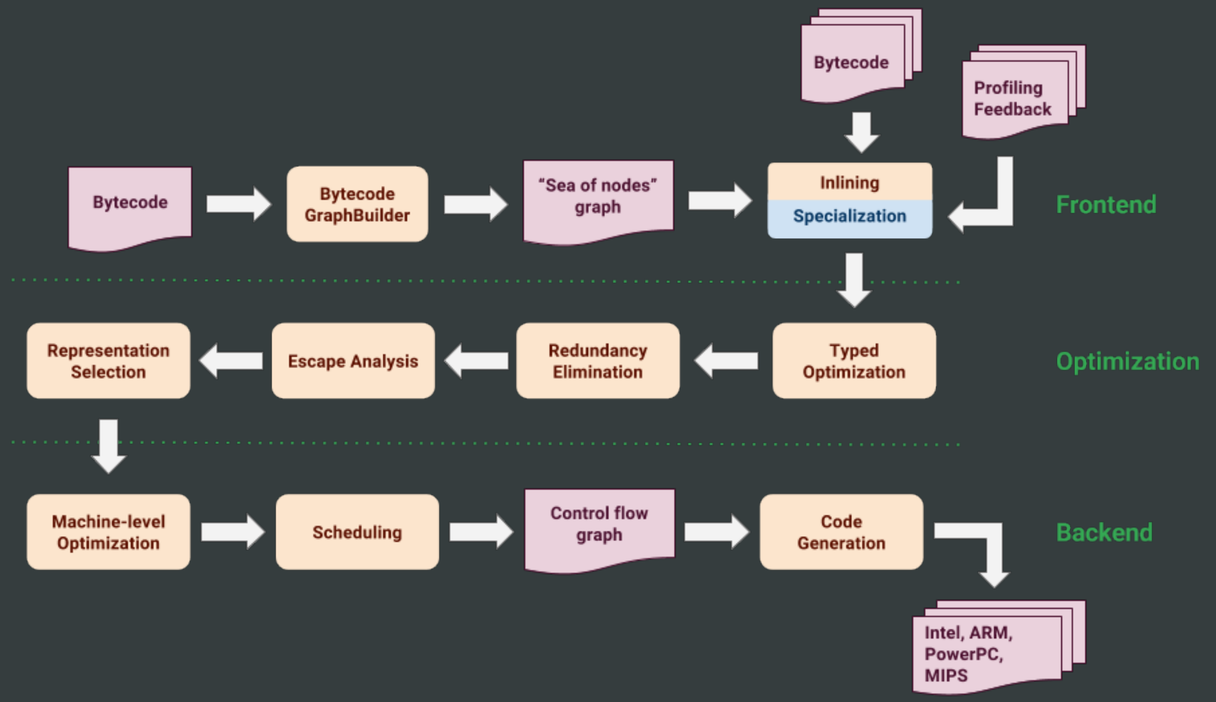
NODESOURCE®

Why the new V8 is so Damn Fast

- Overview of New Compiler Pipeline
- Why is it Faster?
- Which language features are supported?
- What does that mean for your JavaScript?

New V8 Compiler Pipeline

Cleanly split into *Frontend*, *Optimization Layer* and *Backend*



New V8 Compiler Pipeline

- Ignition Interpreter
- TurboFan optimizing Compiler

Ignition Interpreter

- runs highly optimized interpreter code
- *Inline Caches (aka ICs)* gather knowledge about types while program runs, stored in *FeedbackVector*

Performance Cliffs

Previous V8 implementations suffered from huge *performance cliffs*

Performance Cliffs

Ignition’s highly optimized and small interpreter code results in much smaller performance cliffs

TurboFan Optimizing Compiler

Crankshaft is Fast

Turbofan is *insanely* Fast

TurboFan Optimizing Compiler

- recompiles and optimizes *hot code* identified by the runtime profiler
- speculates that kinds of values seen in the past will be seen in the future and generates optimized code just for those cases

Which language features are supported?

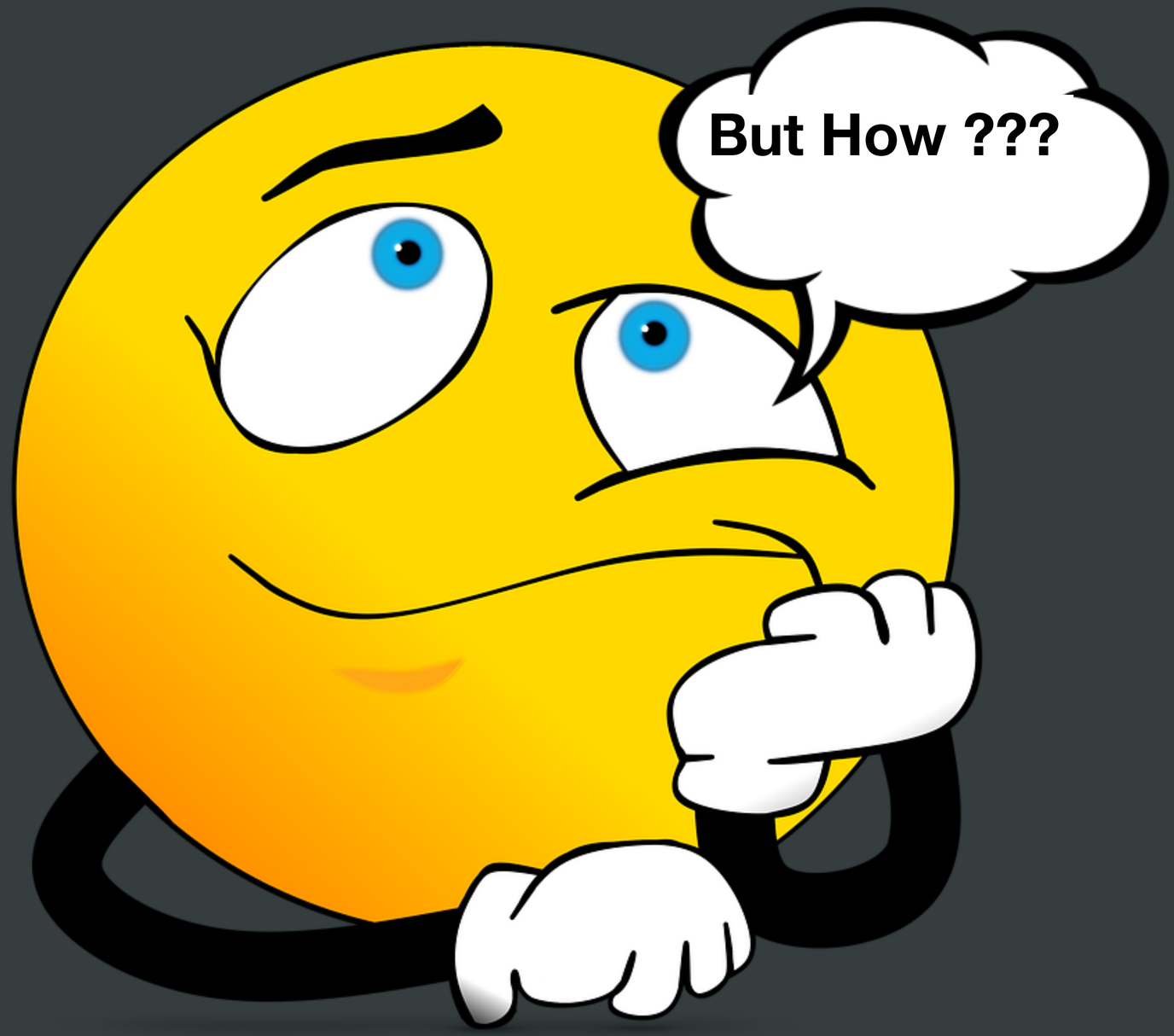
TurboFan is able to optimize *ALL* language features

Function Bind

Due to crossing language boundaries C++/JS `*Function.prototype.bind*` and bound functions had performance issues

Function Bind

New approach to how bound function are implemented avoids crossing C++/JS boundaries entirely


But How ???

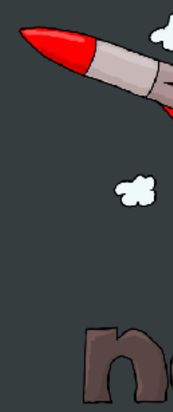
CodeStub Assembler

- defines a portable assembly language built on top of TurboFan's backend and adds a C++ based API to generate highly portable TurboFan machine-level IR directly
- can generate highly efficient code without crossing to C++ runtime

CodeStub Assembler

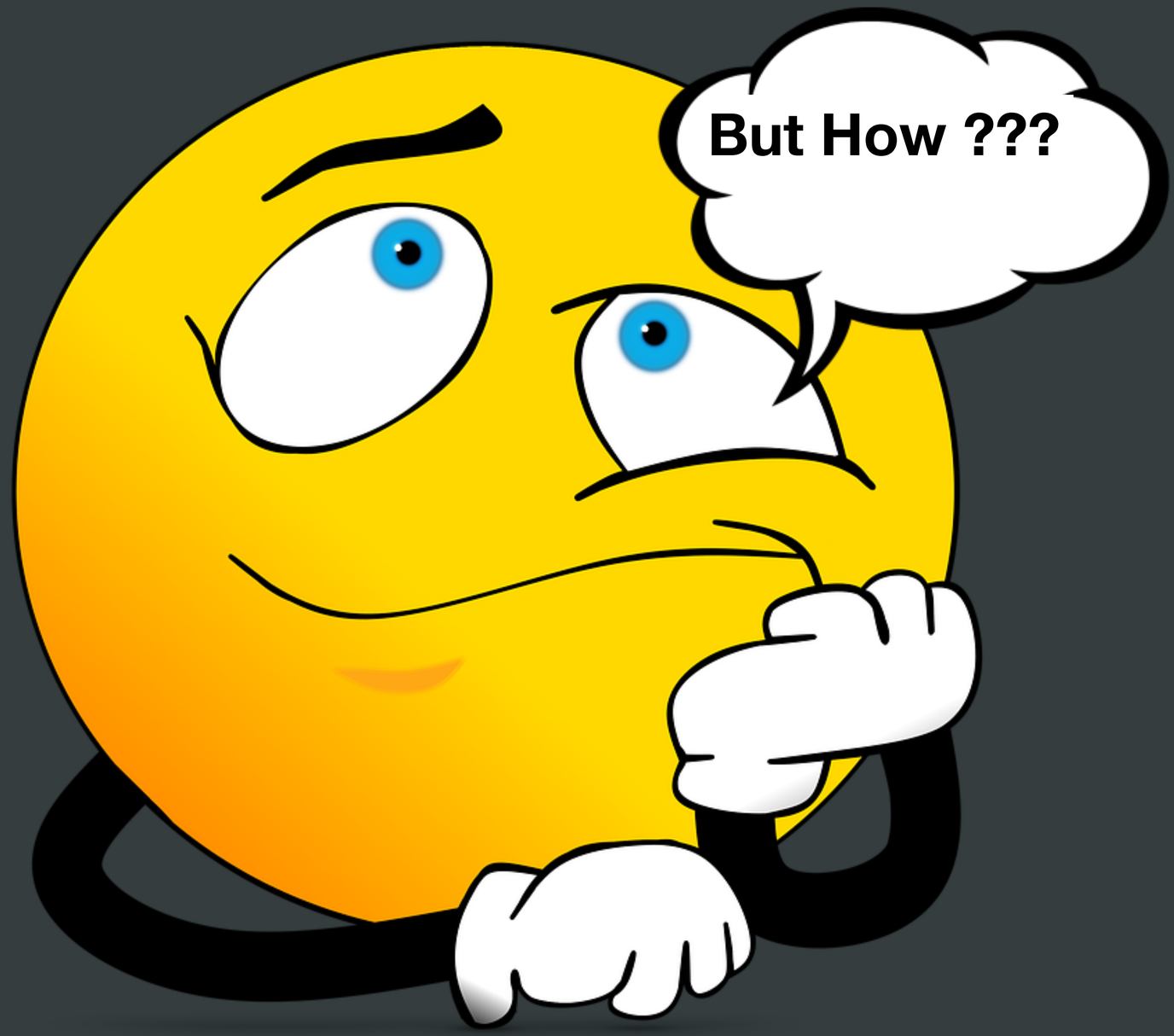
- includes features that make it faster and safer to iterate on new language features
- greatly reduces source for bugs
- CSA code is testable

Demo

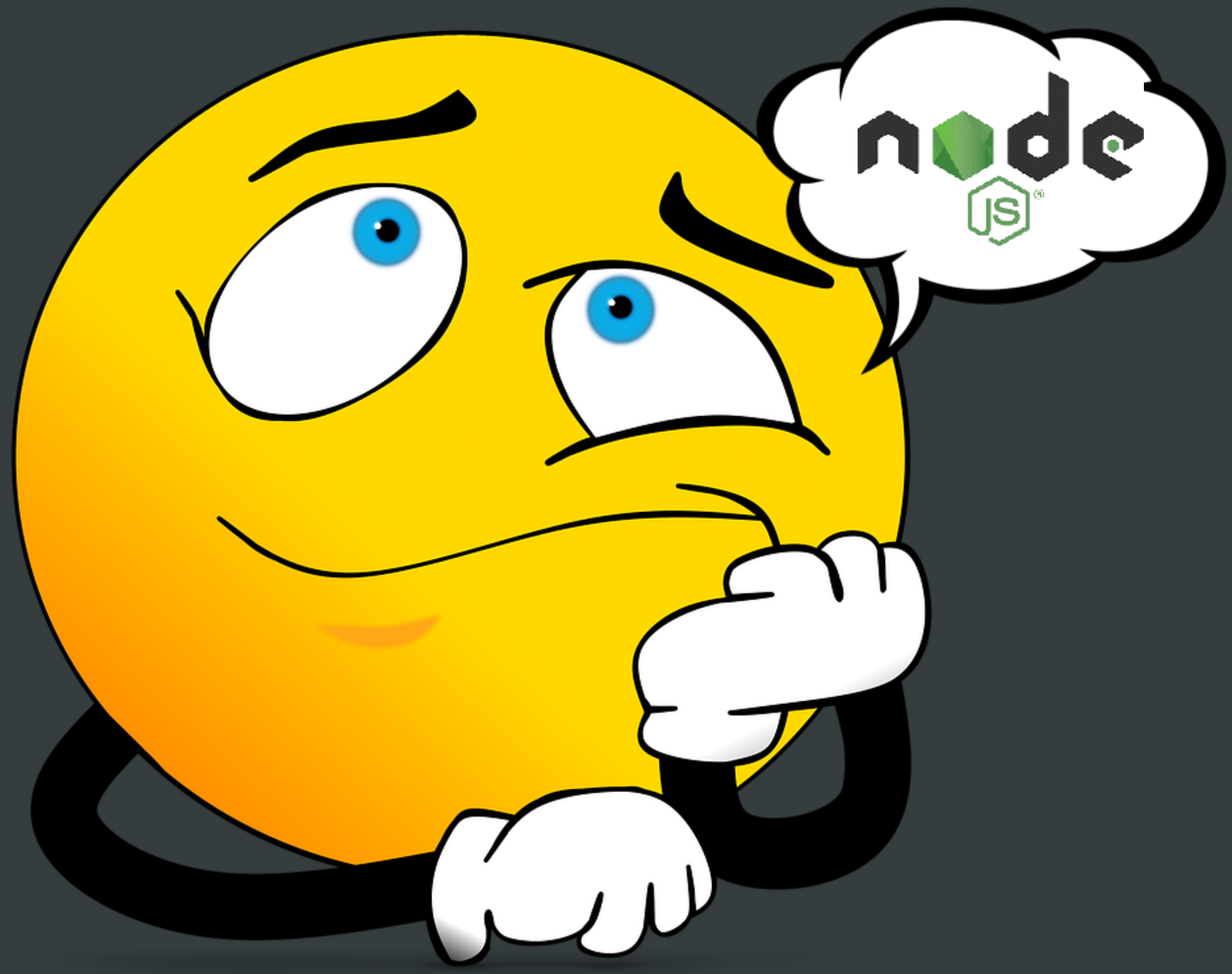


TurboFan

Maximizes the execution speed of your JavaScript by applying more optimizations than were possible in the past


But How ???

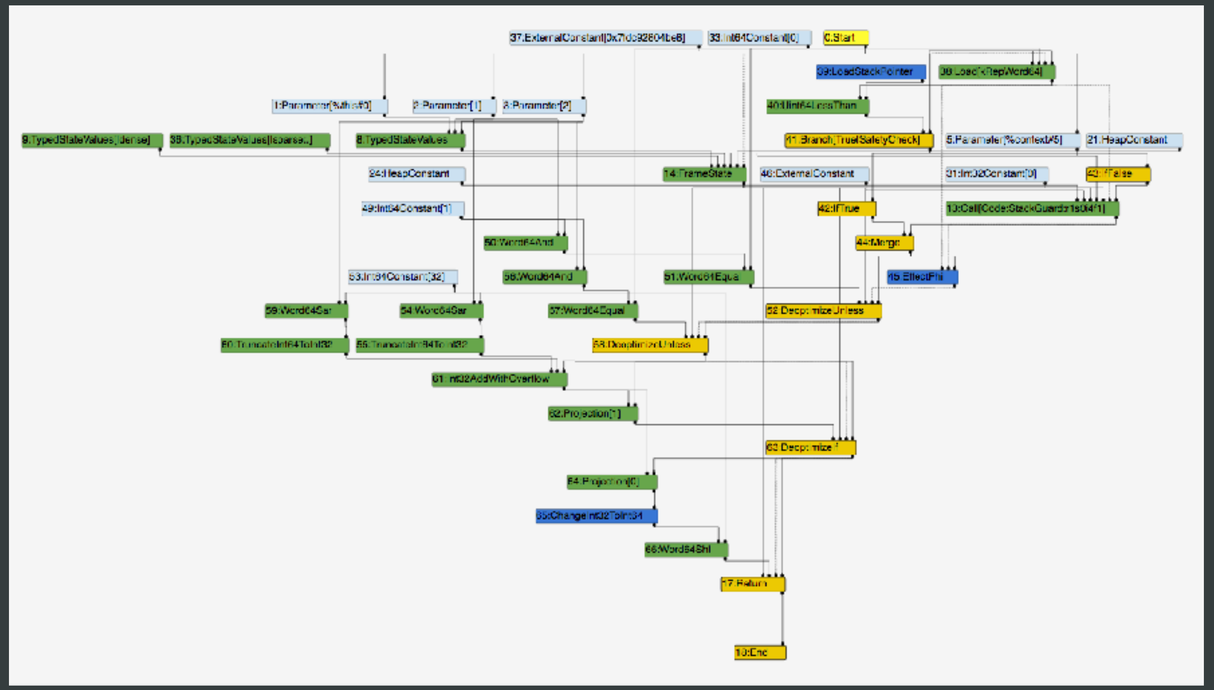
Sea Of Nodes



Sea Of Nodes

Sea Of Nodes

- *control dependencies* between operations instead of program order
- expresses many legal orderings



Sea Of Nodes

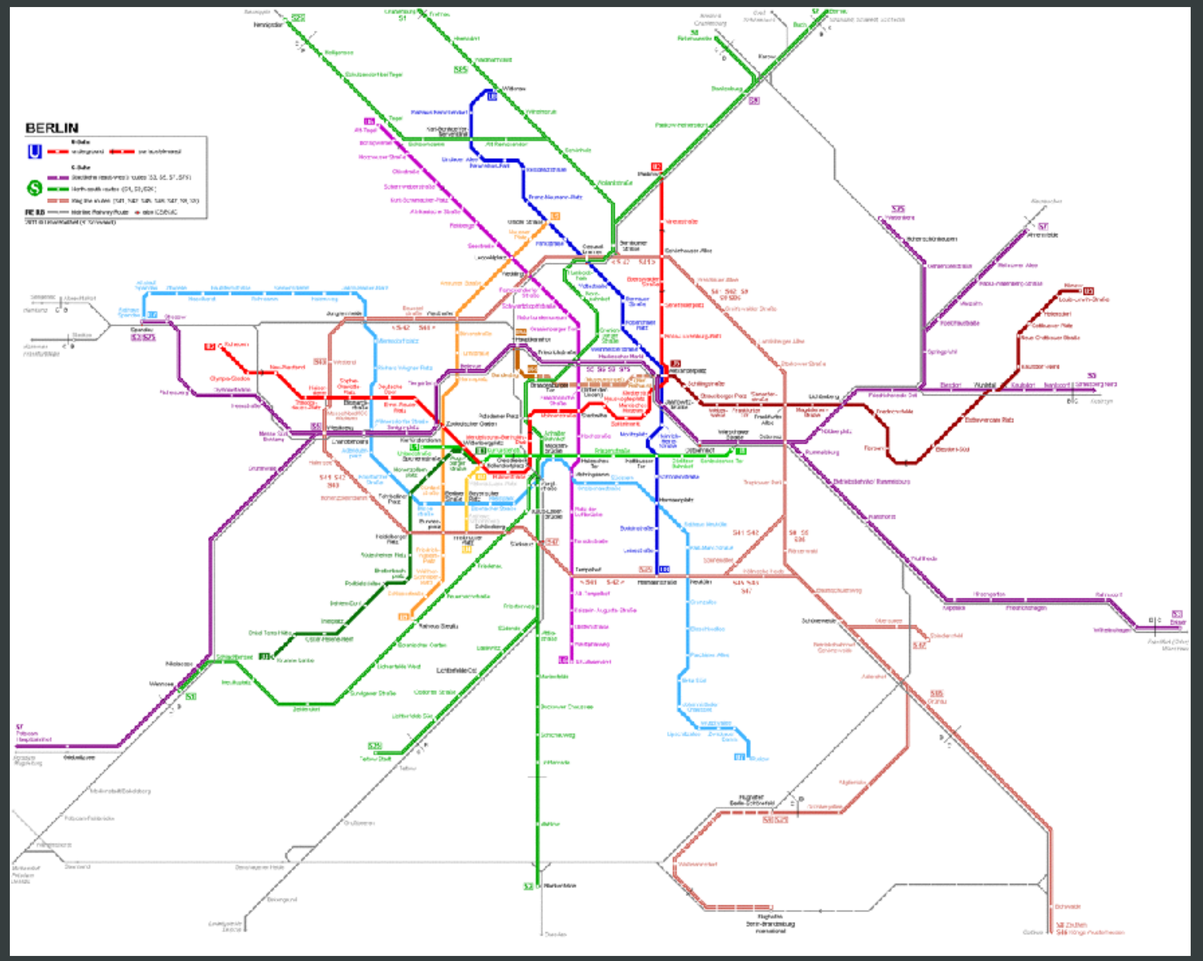
Sea Of Nodes

Enables many Optimizations

- better redundant code elimination due to more code motion
- loop peeling
- control flow elimination
- many more ...

Sea Of Nodes

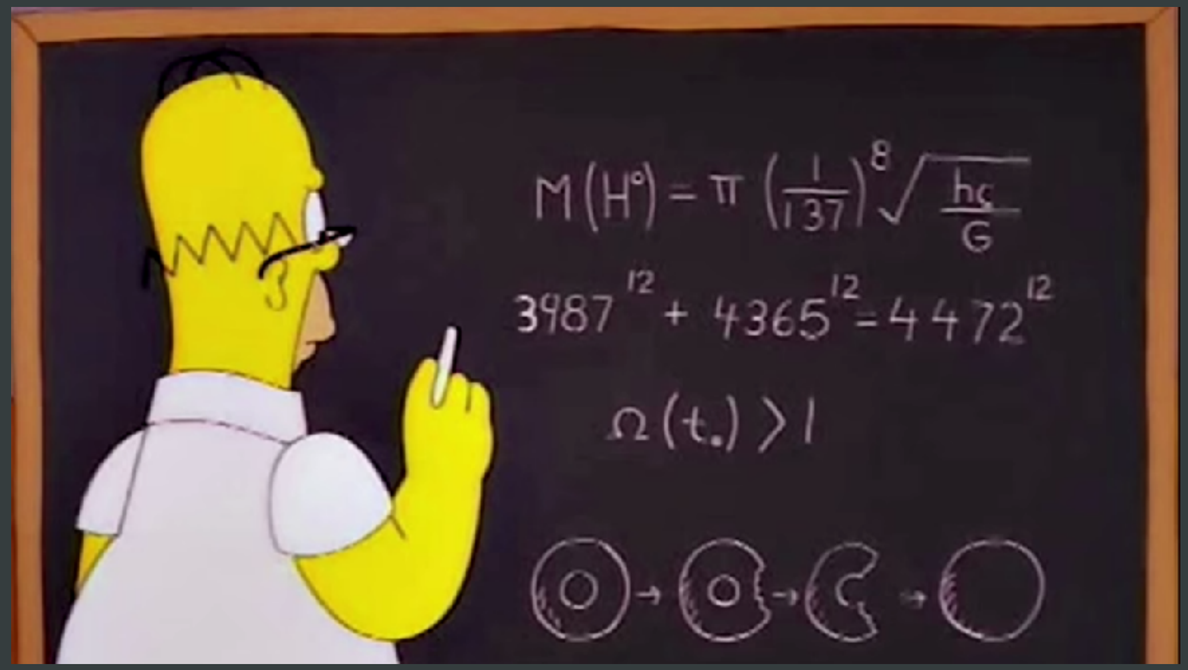
The *CFG* (control flow graph) is built after applying optimizations



Will Any JS do?

Write idiomatic, declarative as in *easy to read* JavaScript with good data structures and algorithms.

All language features will execute with predictable, good performance.



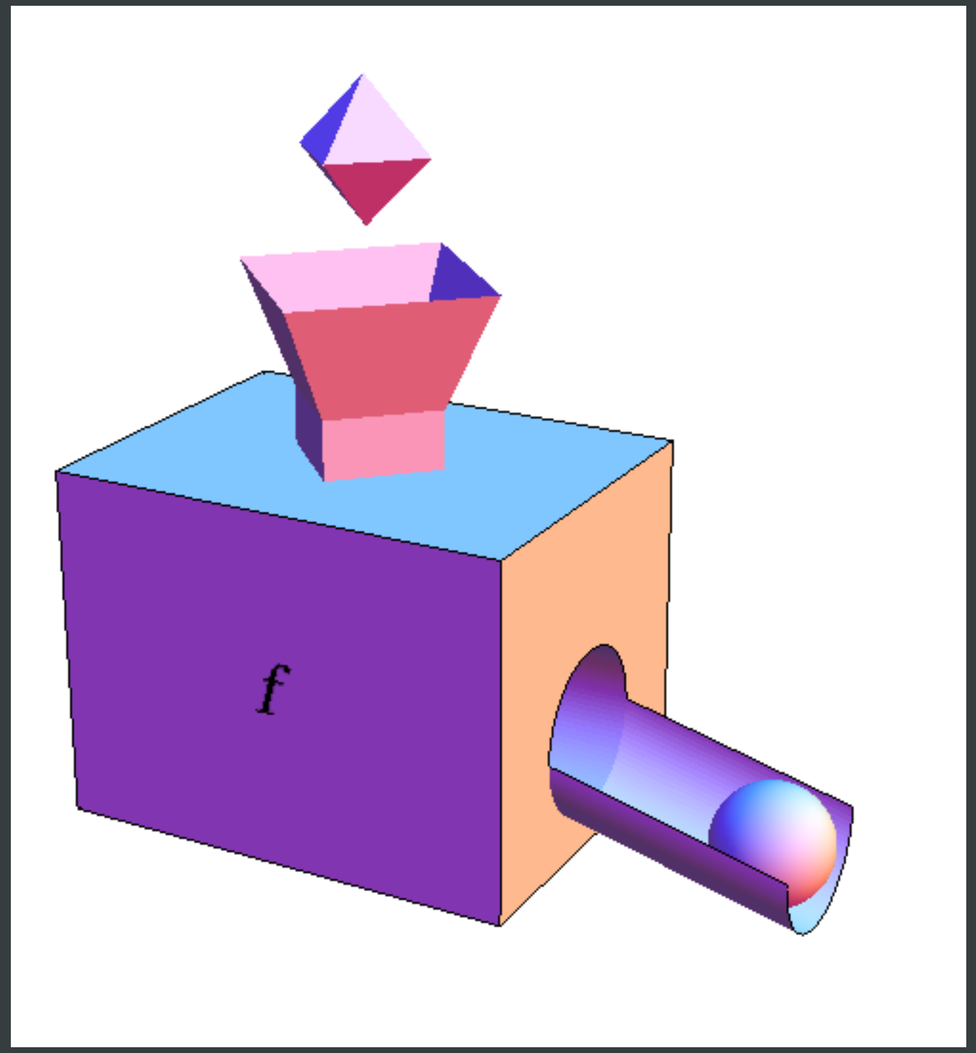
Will Any JS do?

Optimized code for fewer cases is smaller and executes at peak speed.

Smaller is Better

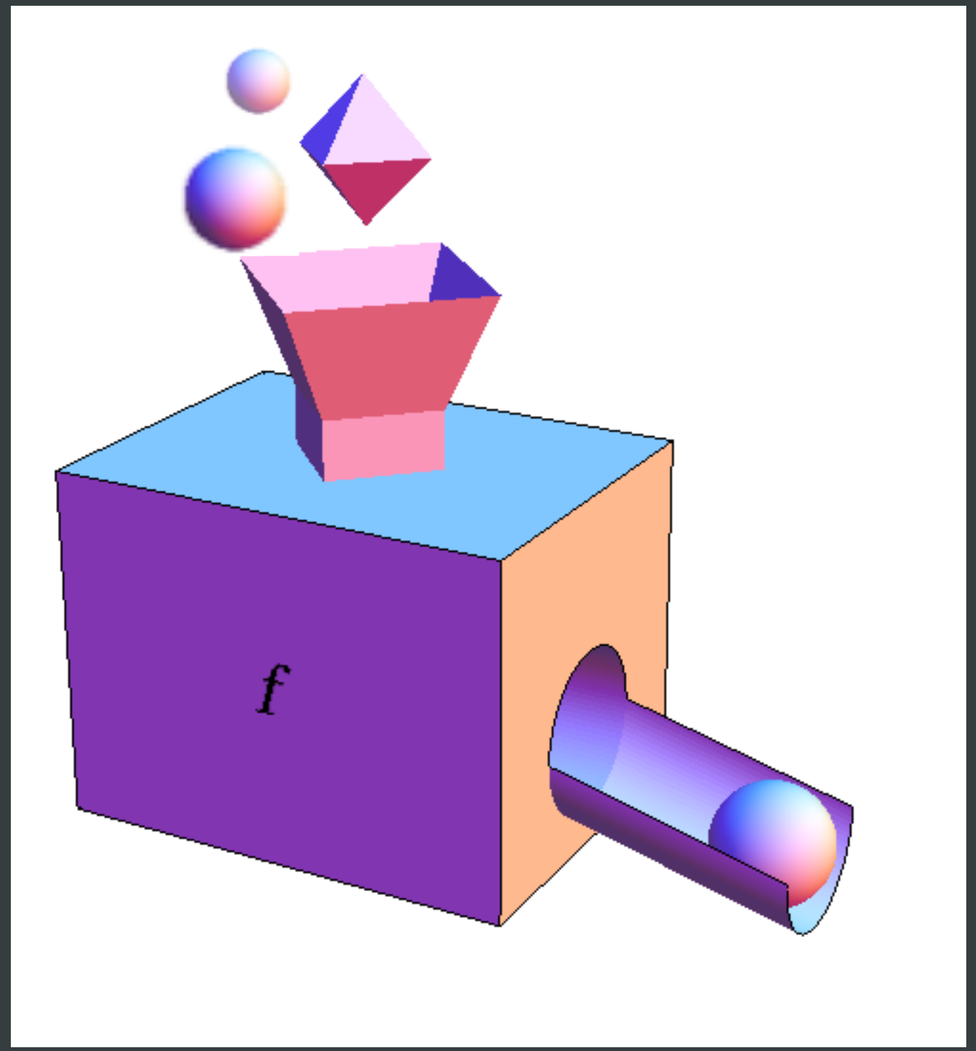
Mono-Poly-Megamorphism

Monomorphic === One Input Type



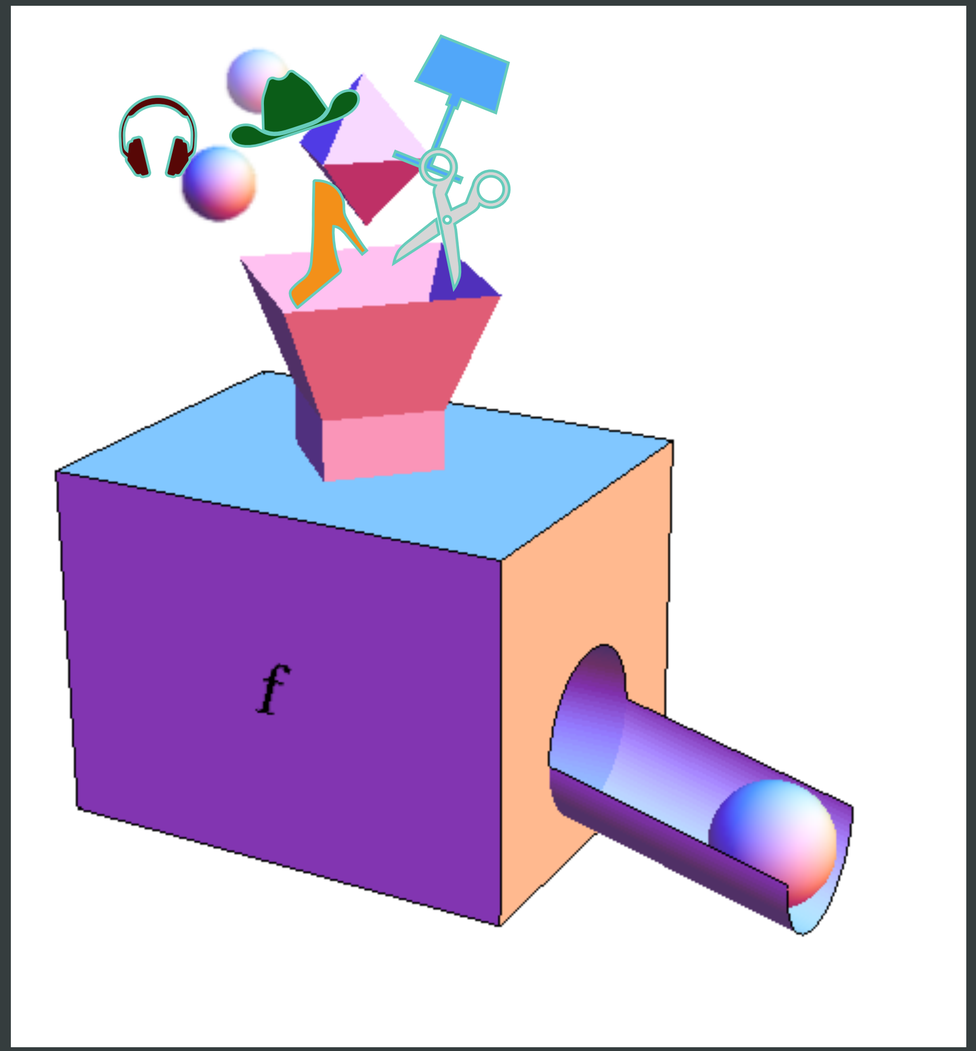
Mono-Poly-Megamorphism

Polymorhic === Two - Four Input Types



Mono-Poly-Megamorphism

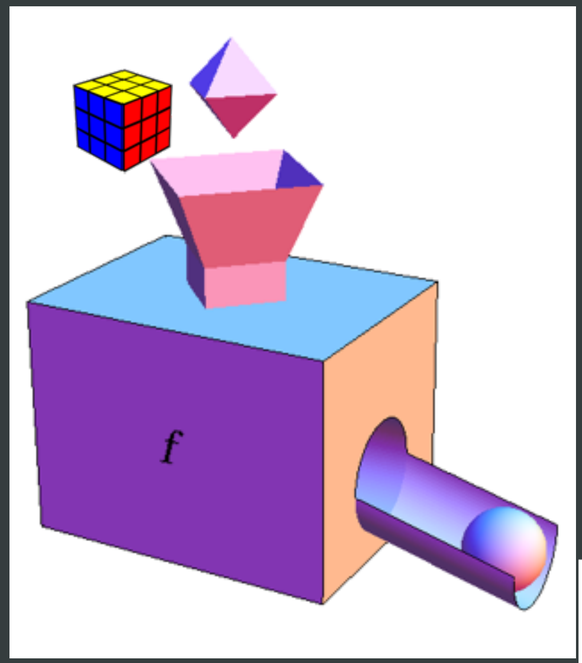
Megamorphic === Five+ Input Types

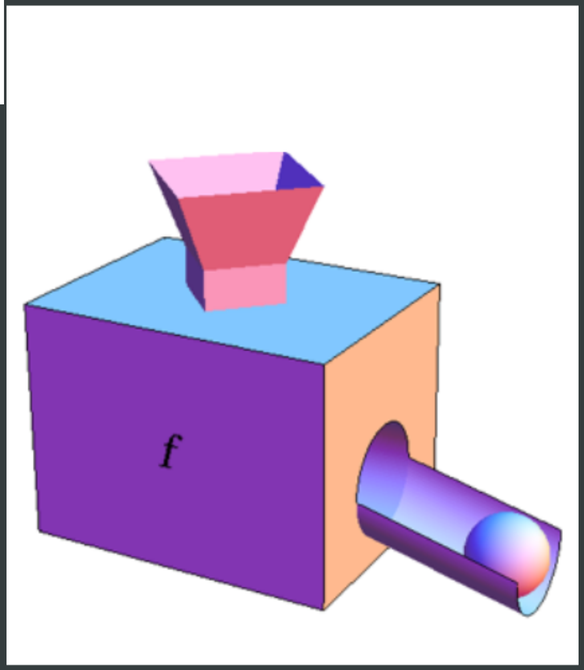


Mono-Poly-Megamorphism

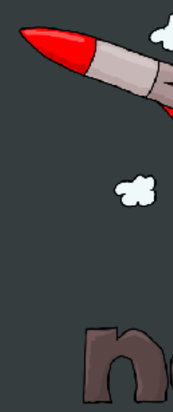
Monomorphic operations are easier optimized

Prefer multiple monomorphic functions over one polymorphic function where performance matters





Demo

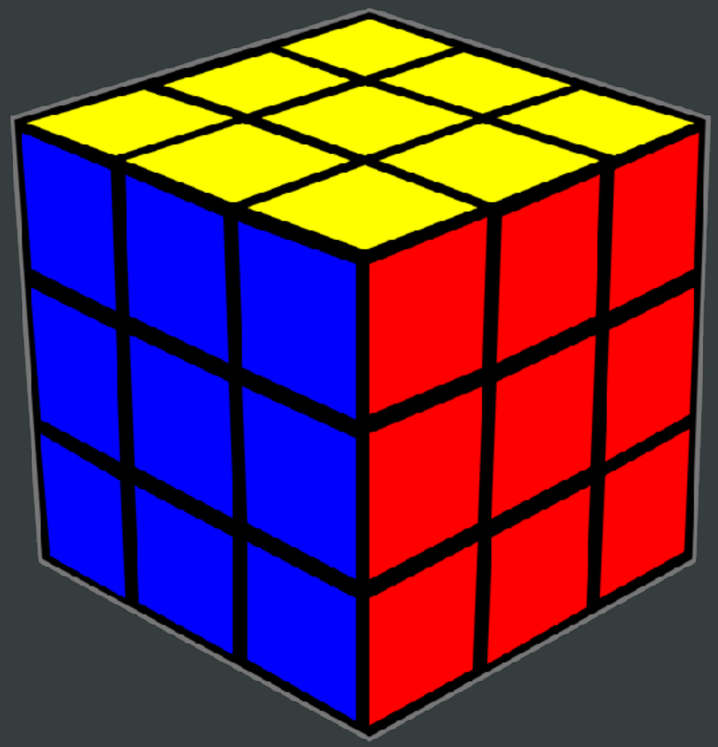


Mono-Poly-Megamorphism

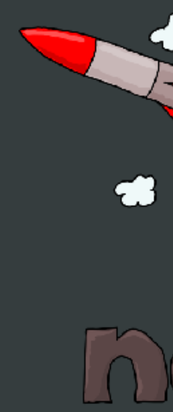
V8 creates hidden classes for your JavaScript objects *aka* Maps

Mono-Poly-Megamorphism

Maps



Demo

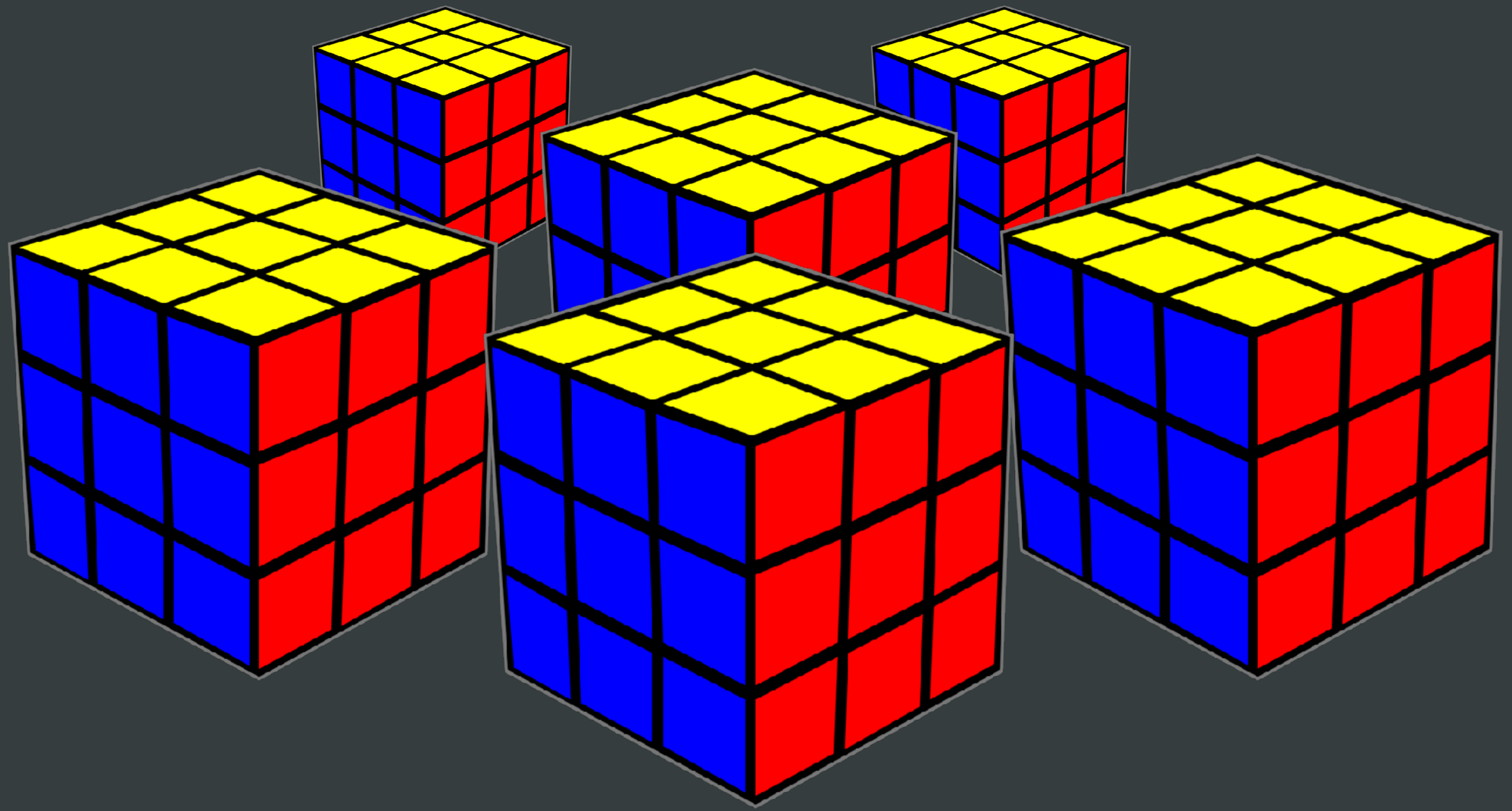


Deoptigate Case Study

- xml2js initially
- xml2js with improvements

Mono-Poly-Megamorphism

Ensure same Maps

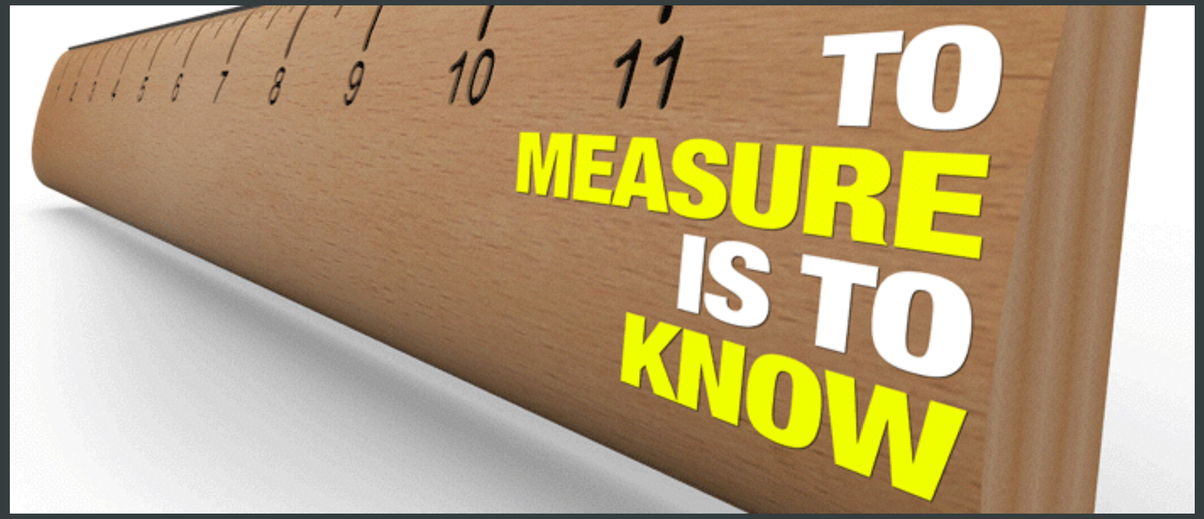


Recommendations

- write idiomatic, declarative JavaScript and focus on your application design
- use appropriate data structures and collections, i.e. *Maps, Sets*
- avoid engine specific workarounds *aka CrankshaftScript*
- prefer *monomorphic* functions in hot code paths

Recommendations

Measure *before* and *after* each optimization you apply to your code



Resources V8

- **V8-perf resources**
  - V8 compiler
  - Language Features
- V8 code search

Resources Tools

- deoptigate
- turbolizer
- map-processor
- V8 tools

Resources Blog Posts

- deoptigate
- Why the New V8 is so Damn Fast

Thank you.

Thorsten Lorenz

@thlorenz