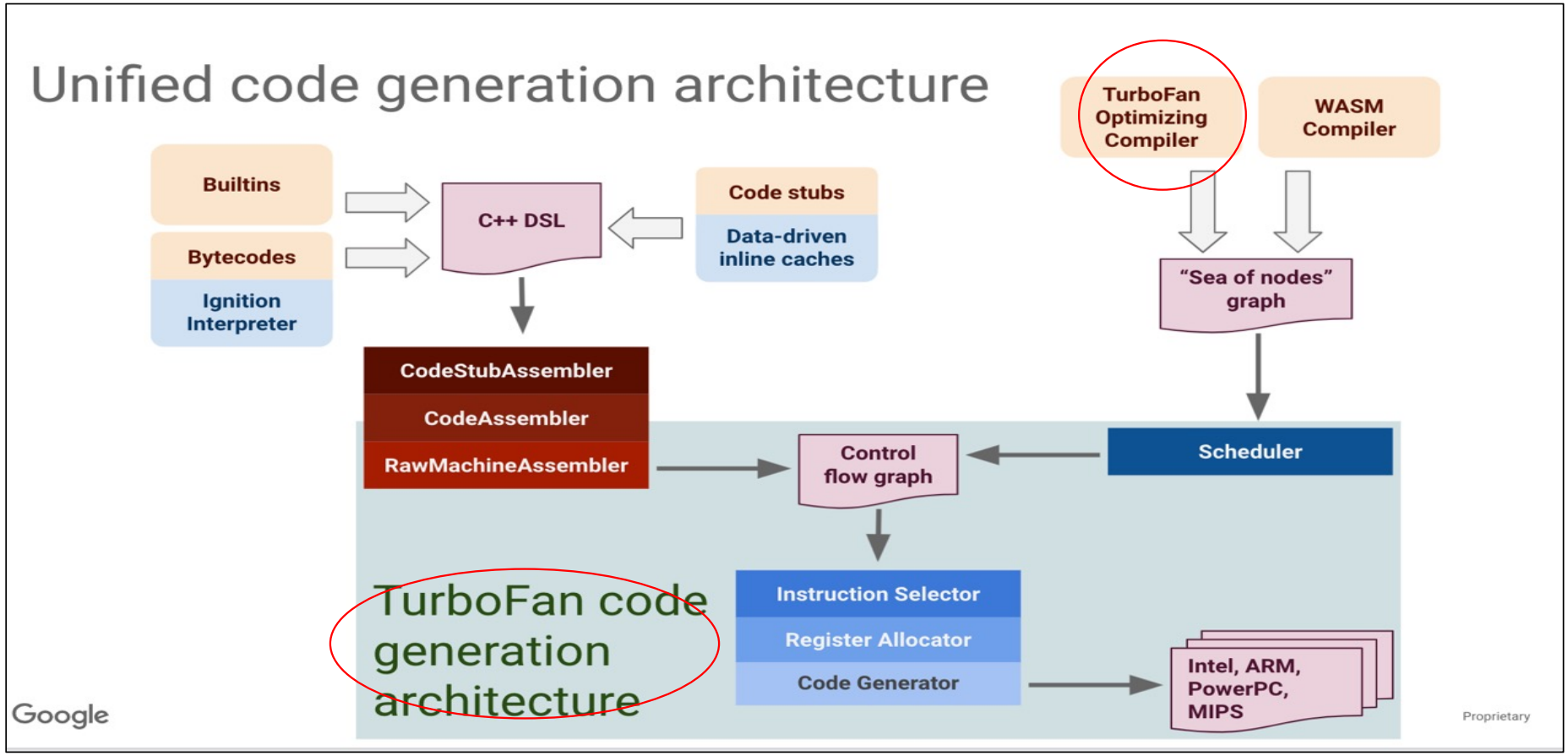
中国科学院软件研究所 Institute of Software Chinese Academy of Sciences | 智能软件研究中心 Intelligent Software Research Center

V8 Turbofan 架构概览

PLCT实验室 邱吉
qiuji@iscas.ac.cn

2022/02/22

TurboFan是V8执行引擎的核心组成部分



https://benediktmeurer.de/2017/03/01/v8-behind-the-scenes-february-edition

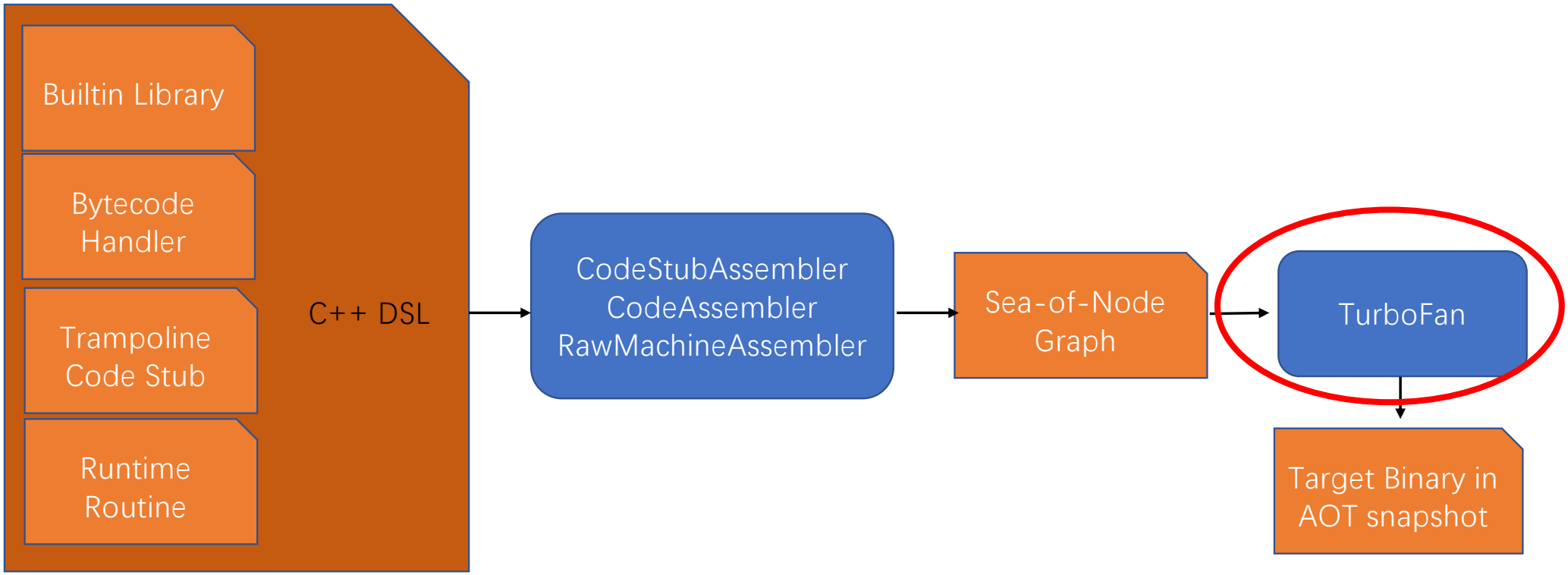
今天的内容： TurboFan 架构概览

- TurboFan在V8中的位置和所发挥的作用
  - AOT
  - JIT
- TurboFan的IR
  - SON
  - 层次化的Node
  - 层次化的IR图
- TurboFan的Pipeline结构
  - Phase
  - Pipeline的组织框架
  - Pipeline的数据载体
- TurboFan的pipeline的编译任务驱动

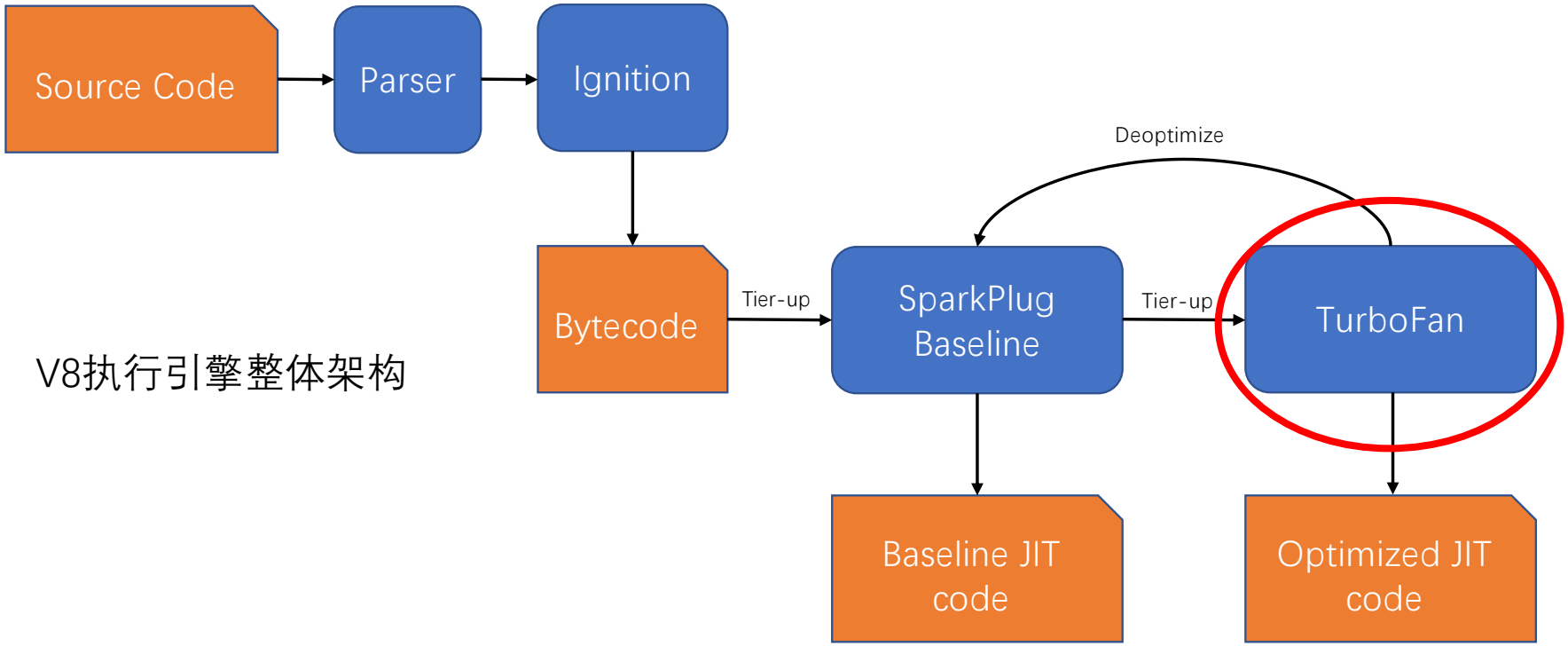
今天的内容： TurboFan 架构概览

- TurboFan在V8中的位置和所发挥的作用
  - AOT
  - JIT
- TurboFan的IR
  - SON
  - 层次化的Node
  - 层次化的IR图
- TurboFan的Pipeline结构
  - Phase
  - Pipeline的组织框架
  - Pipeline的数据载体
- TurboFan的pipeline的编译任务驱动

V8的构建流程中TurboFan的位置：发挥AOT作用



TurboFan在V8执行引擎中的位置：发挥JIT作用



V8执行引擎整体架构

今天的内容： TurboFan 架构概览

- TurboFan在V8中的位置和所发挥的作用
  - AOT
  - JIT
- TurboFan的IR
  - SON
  - 层次化的Node
  - 层次化的IR图
- TurboFan的Pipeline结构
  - Phase
  - Pipeline的组织框架
  - Pipeline的数据载体
- TurboFan的pipeline的编译任务驱动

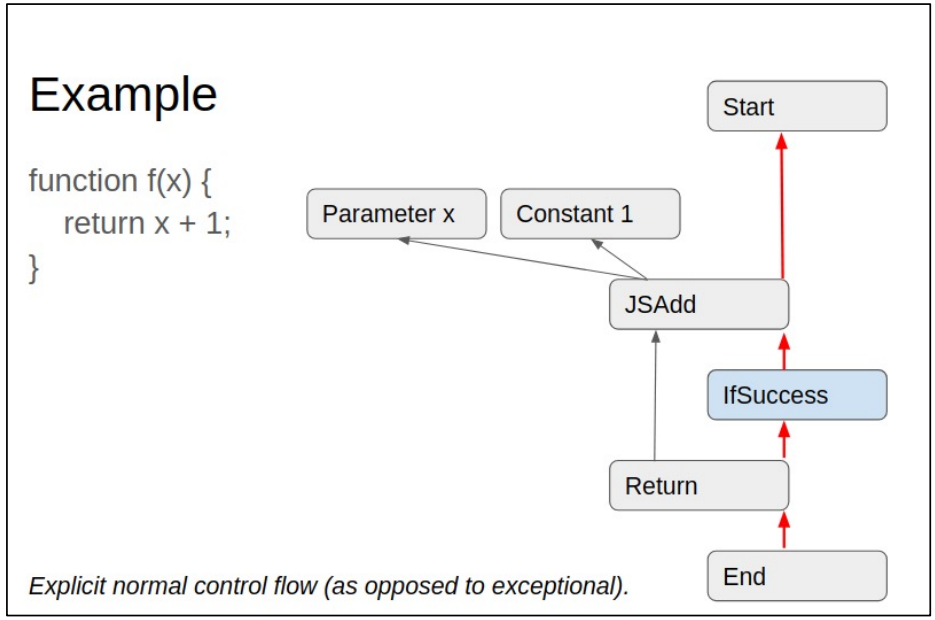
TurboFan的IR

- TurboFan 的 Sea-of-Node IR

> - Graph based IR
>   - Nodes for operations.
>   - Edges for value flow, *control flow* and dependencies.
>   - No distinction between basic blocks and statements.
>   - Single-static assignment.

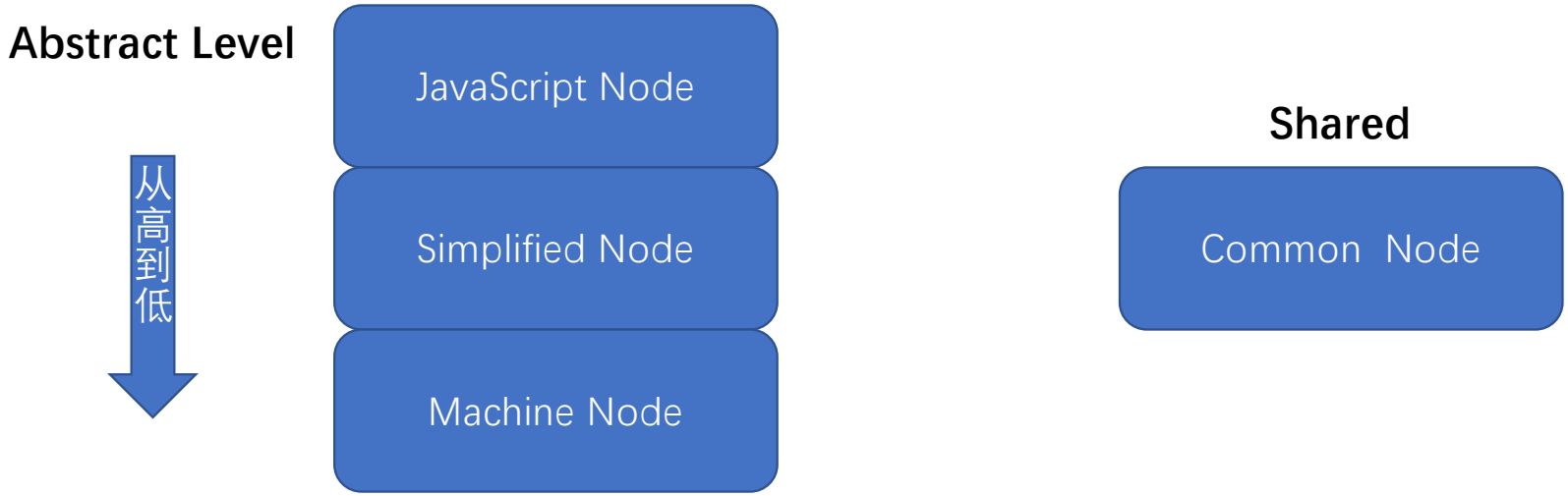
-- “Turbofan IR，Jaroslav Sevcik”

- 一个简单的例子：右图->
  - 灰色边：反向后的数据流
  - 红色边：反向后的控制流

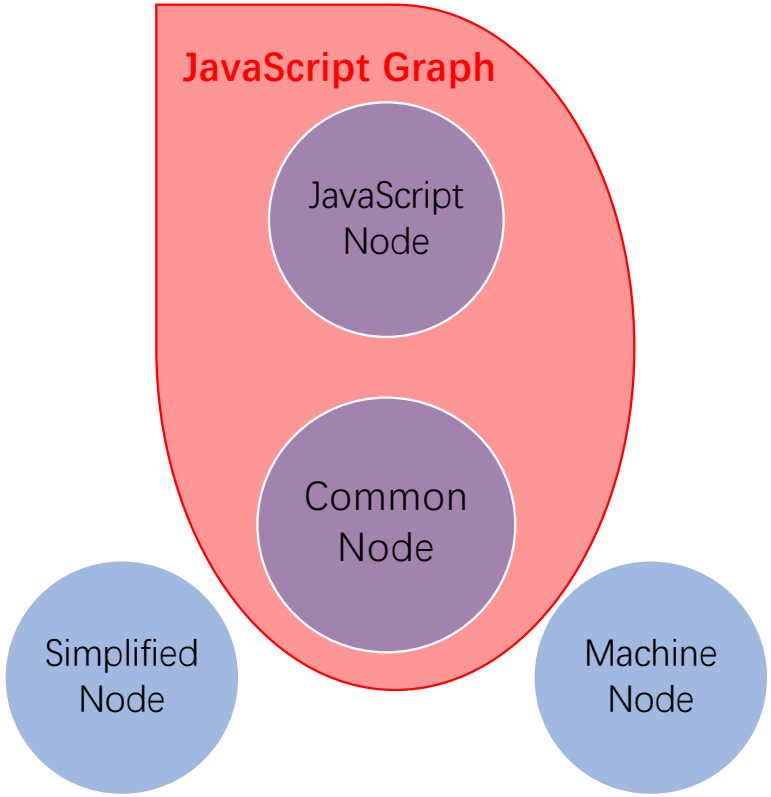


https://docs.google.com/presentation/d/1Z9ilHojKDrXvZ27gRX51UxHD-bKf1QcPzSijntpMJBM/edit#slide=id.p

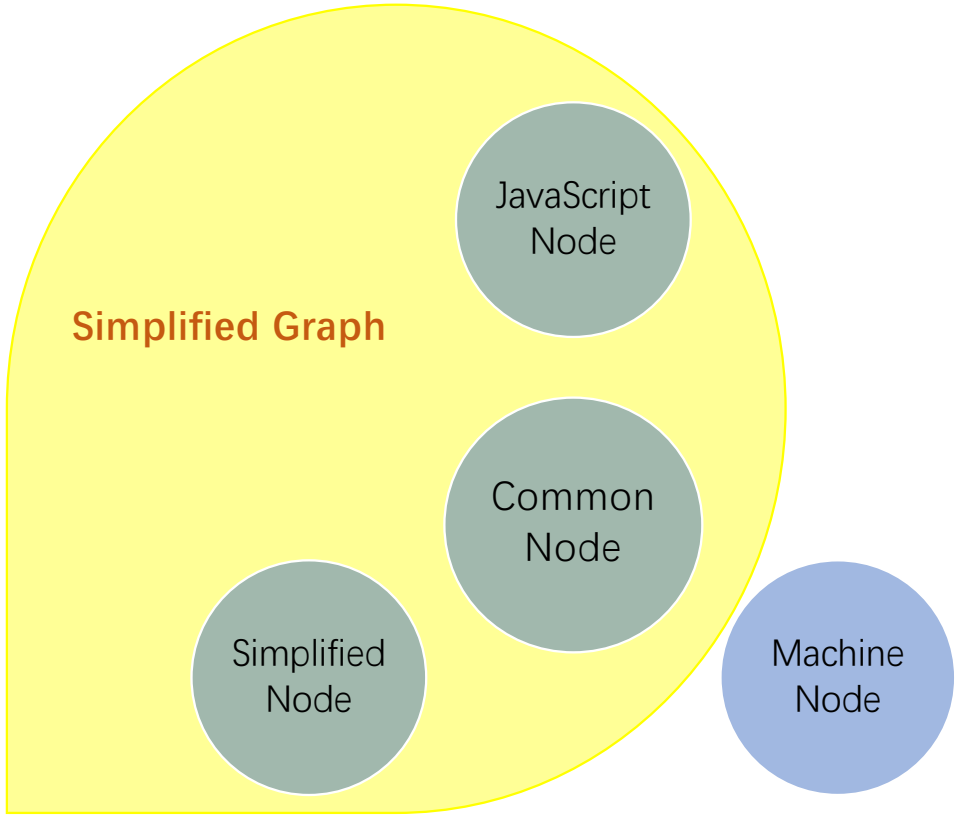
TurboFan IR的层次化节点类型



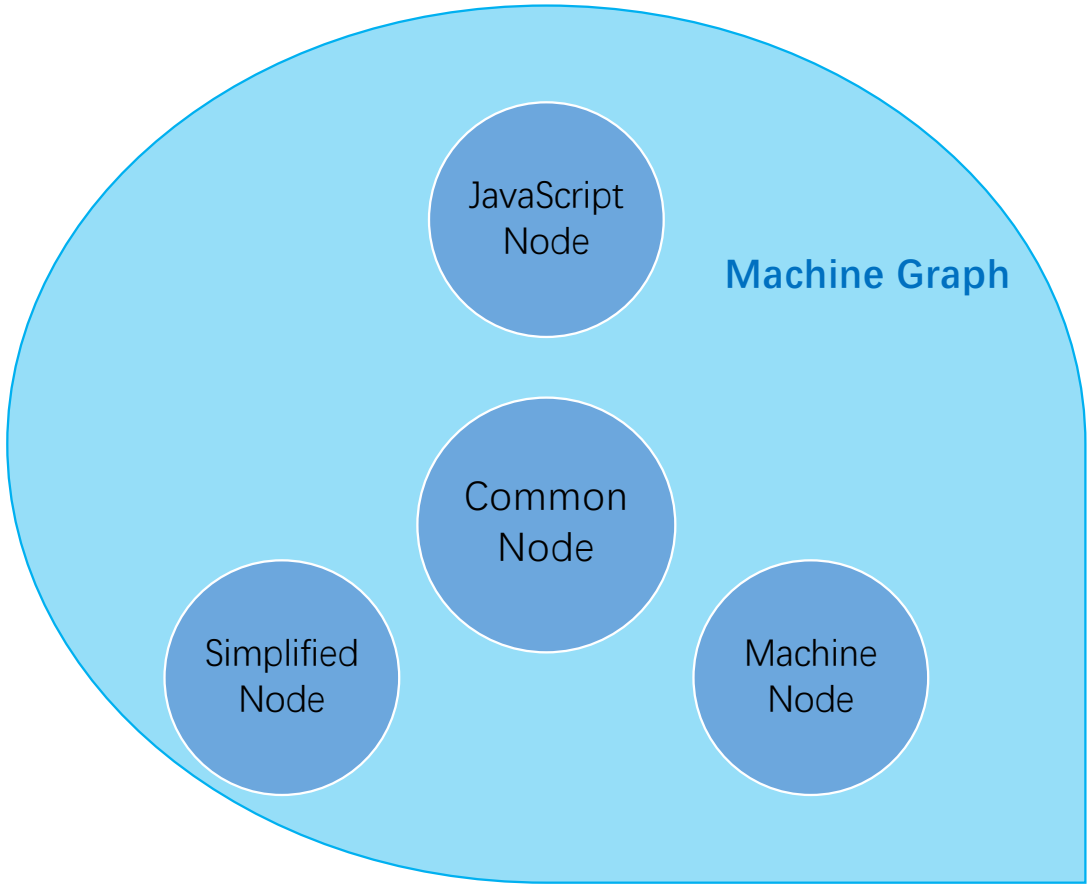
TurboFan SON图的不同层次-最高级别的图JavaScript Graph



TurboFan SON图的不同层次-中间级别的图Simplified Graph

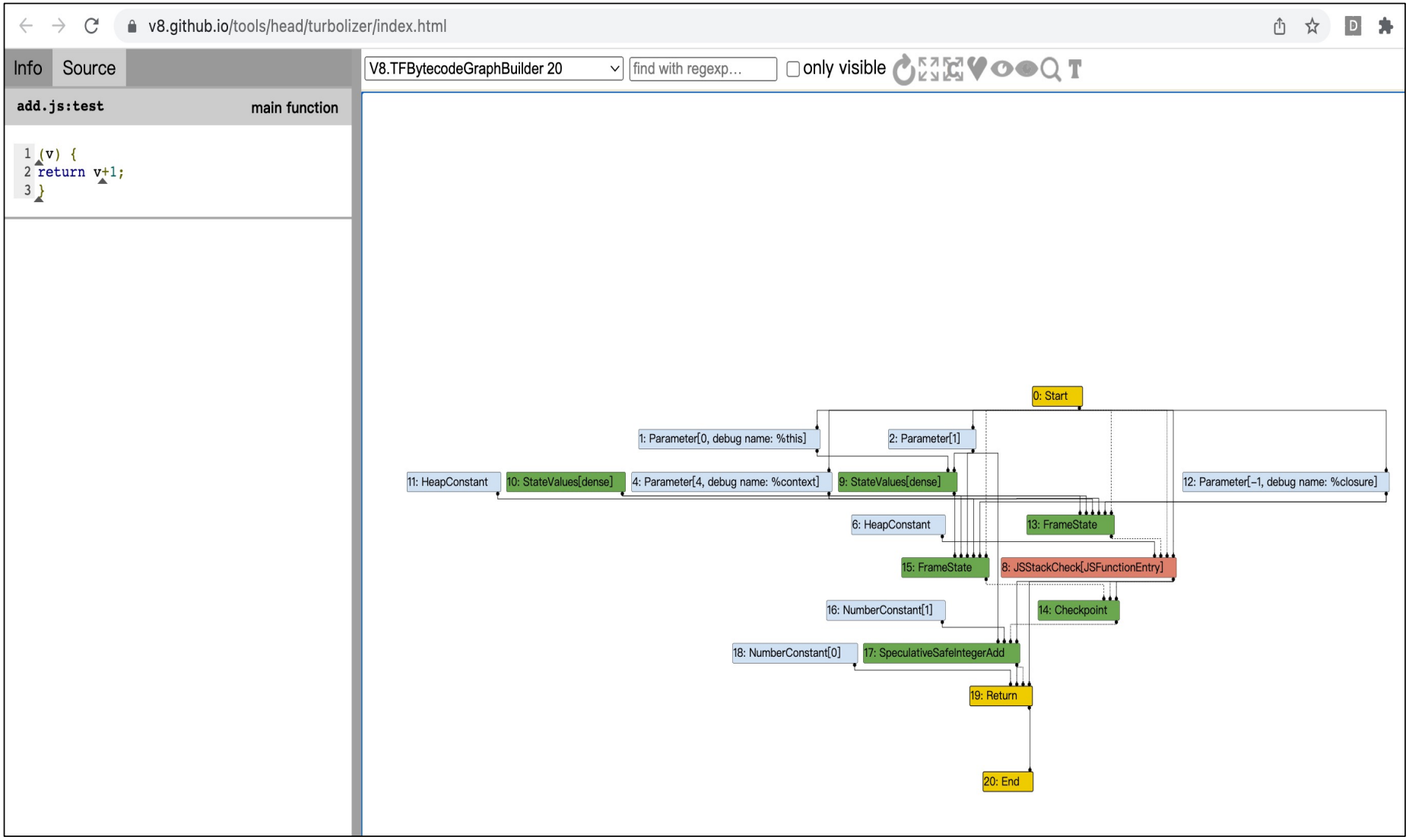


TurboFan SON图的不同层次-最低级别的图Machine Graph



TurboFan IR的查看方式

- d8命令行参数
  - -trace-turbo-graph: 文本形式输出到终端
  - --trace-turbo：输出执行目录下的json文件和后缀为.cfg的文本文件中
    - json文件的命名规则是turbo-<函数名>-<编号>.json，它保存了每一个phase对应的IR
    - test.cfg文件保存的是SON图经过SON Scheduler后具有Basic Block和Control flow的IR图
    - json文件可以通过V8提供的工具Turbolizer来进行可视化的展示
    - https://v8.github.io/tools/head/turbolizer/index.html


v8.github.io/tools/head/turbolizer/index.html
Info
Source
add.js:test
main function
1 (v) {
2 return v+1;
3 }
V8.TFBytecodeGraphBuilder 20
find with regexp...
only visible
0: Start
1: Parameter[0, debug name: %this]
2: Parameter[1]
11: HeapConstant
10: StateValues[dense]
4: Parameter[4, debug name: %context]
9: StateValues[dense]
12: Parameter[-1, debug name: %closure]
6: HeapConstant
13: FrameState
15: FrameState
8: JSStackCheck[JSFunctionEntry]
16: NumberConstant[1]
14: Checkpoint
18: NumberConstant[0]
17: SpeculativeSafeIntegerAdd
19: Return
20: End

今天的内容： TurboFan 架构概览

- TurboFan在V8中的位置和所发挥的作用
  - AOT
  - JIT
- TurboFan的IR
  - SON
  - 层次化的Node
  - 层次化的IR图
- TurboFan的Pipeline结构
  - Phase
  - Pipeline的组织框架
  - Pipeline的数据载体
- TurboFan的pipeline的编译任务驱动

TurboFan的pipeline结构概览

- 类似于LLVM/GCC/Hotspot, 具有multi-pass和phase的结构
- 由phase作为基本单元，对不同的编译任务构成不同的pipeline结构
- Class PipelineImpl负责为不同的编译任务（job）组织不同的phase形成不同Pipeline
- Class PilelineData是编译流水线数据载体

TurboFan的phase结构

- TurboFan的Phase的实现在v8/src/comiler/pipeline.cc文件中，该文件一共以Struct的形式定义了56个Phase。
- 以GraphBuilderPhase为例说明Phase的定义过程：

```
struct GraphBuilderPhase {
  DECL_PIPELINE_PHASE_CONSTANTS(BytecodeGraphBuilder) //通过宏定义了phase_name函数
  void Run(PipelineData* data, Zone* temp_zone) { //phase的主体部分
    BytecodeGraphBuilderFlags flags;
    if (data->info()->analyze_environment_liveness()) {
      flags |= BytecodeGraphBuilderFlag::kAnalyzeEnvironmentLiveness;
    }
    if (data->info()->bailout_on_uninitialized()) {
      flags |= BytecodeGraphBuilderFlag::kBailoutOnUninitialized;
    }

    JSFunctionRef closure = MakeRef(data->broker(), data->info()->closure());
    CallFrequency frequency(1.0f);
    BuildGraphFromBytecode(
        data->broker(), temp_zone, closure.shared(),
        closure.raw_feedback_cell(data->dependencies()),
        data->info()->osr_offset(), data->jsgraph(), frequency,
        data->source_positions(), SourcePosition::kNotInlined,
        data->info()->code_kind(), flags, &data->info()->tick_counter(),
        ObserveNodeInfo{data->observe_node_manager(),
                        data->info()->node_observer()});
  }
};
```

TurboFan的pipeline的框架-PipelineImpl类

- 通过PipelineImpl类来组织并运行各个phase，PipelineImpl是编译流水线的最终实现者
- 两种调用Phase的方法：
  - 直接代理Phase结构体的Run

```
template <typename Phase, typename... Args>
void PipelineImpl::Run(Args&&... args) {
  PipelineRunScope scope(this->data_, Phase::phase_name());
  Phase phase;
  phase.Run(this->data_, scope.zone(), std::forward<Args>(args)...);
}
```

  - 通过函数包装若干Phase，提供给上层的编译任务（CompilationJob）使用
    - InitializeHeapBroker，CreateGraph，OptimizeGraph，OptimizeGraphForMidTier，ComputeScheduledGraph，SelectInstructions，AssembleCode，FinalizeCode，，SelectInstructionsAndAssemble，GenerateCode

TurboFan的pipeline的数据载体-PipelineData类

- PipelineData是TurboFan compilation pipeline数据和操作句柄载体
- PipelineData是连接每一个编译任务和TurboFan的Pipeline静态结构的纽带，PipelineData包含的动态数据，就像水流一样，通过PipelineImpl对象的实例化，流入了的Pipeline的管道中
- 在TurboFan中，不同类型的编译任务，会产生不同的水流，流入不同的管道。不同的编译任务来自于V8构建或运行的不同阶段，或者V8接受的不同类型执行命令或脚本

```
class PipelineData {
  OptimizedCompilationInfo* const info_; //保存编译优化的信息，如osr
  std::unique_ptr<char[]> debug_name_; //保存编译单元的名字
  ZoneStats* const zone_stats_; // 内存分配Zone相关统计信息
  PipelineStatistics* pipeline_statistics_ = nullptr; //流水线的统计数据
  base::Optional<OsrHelper> osr_helper_;  //栈上替换优化的helper
  MaybeHandle<Code> code_; //生成代码的handler

  SimplifiedOperatorBuilder* simplified_ = nullptr;//Simplified 操作的Builder句柄
  MachineOperatorBuilder* machine_ = nullptr; //Machine操作的Builder句柄
  CommonOperatorBuilder* common_ = nullptr; //Common操作的Builder句柄

  ……

};
```

今天的内容： TurboFan 架构概览

- TurboFan在V8中的位置和所发挥的作用
  - AOT
  - JIT
- TurboFan的IR
  - SON
  - 层次化的Node
  - 层次化的IR图
- TurboFan的Pipeline结构
  - Phase
  - Pipeline的组织框架
  - Pipeline的数据载体
- TurboFan的pipeline的编译任务驱动

TurboFan的pipeline的编译任务驱动

- TurboFan具有5种不同的pipeline 任务，对应不同的驱动流程（phase orgnization）
  - PipelineCompilationJob
  - WasmHeapStubCompilationJob
  - GenerateCodeForStubCode
  - GenerateCodeForWasmFunction
  - GenerateCodeForWasmNativeStub

PipelineCompilationJob

- 一个PipelineCompilationJob对象，对应一个JavaScript function的JIT编译任务
- PipelineCompilationJob接收Bytecode并产生SON Graph，编译完成从JavaScript Graph到Simplified Graph，最后到MachineGraph的下降，最后进入后端生成目标平台的二进制代码
- PipelineCompilationJob有TopTier和MidTier之分

	TopTier	MidTier
前端	CreateGraph	CreateGraph
中端	OptimizeGraph	OptimizeGraphFor MidTier
后端	AssembleCode FinalizeCode	AssembleCode FinalizeCode

- TopTier Job = 狭义意义上的TurboFan

WasmHeapStubCompilationJob

- WasmHeapStubCompilationJob类用于对Wasm的Heap Stub Object进行JIT编译
- Heap指的是该V8 isolate中能被垃圾回收器管理的managed heap
- Wasm在V8中运行需要大量JavaScript 和Wasm，以及C++ native函数之间的wrapper来辅助，这些wrapper通常以目标二进制形式的code stub存在，其中heap stub是跟运行实例相关的，无法在各个V8 isolate之间共享的code stub
- WasmHeapStubCompilationJob接收的是Machine Graph，因此pipeline只涉及简单的Machine Graph上的MemoryOptimization
- pipeline的配置如下：

```
MemoryOptimizationPhase
ComputeScheduledGraph()
SelectInstructionsAndAssemble()
FinalizeCode
```

GenerateCodeForStubCode

- GenerateCodeForStubCode函数用于V8的AOT编译，主要任务是将Builtin编译成本地代码
- GenerateCodeForStubCode接收的也是Machine Graph，其pipeline配置如下：

```
CsaEarlyOptimizationPhase
MemoryOptimizationPhase
CsaOptimizationPhase
DecompressionOptimizationPhase
VerifyGraphPhase
ComputeScheduledGraph()
SelectInstructionsAndAssemble()
FinalizeCode
```

GenerateCodeForWasmFunction

- GenerateCodeForWasmFunction函数用于对Wasm Function生成JIT code，是Wasm执行过程中的优化JIT
- 由于Wasm指令集已经比较接近机器代码，因此Wasm指令将直接被转换成Machine Graph作为TurboFan的输入
- GenerateCodeForWasmFunction的pipeline配置如下。

```
WasmLoopUnrollingPhase
WasmInliningPhase
WasmOptimizationPhase
WasmBaseOptimizationPhase
MemoryOptimizationPhase
ComputeScheduledGraph()
SelectInstructions()
```

GenerateCodeForWasmNativeStub

- GenerateCodeForWasmNativeStub函数用于对Wasm的native stub object进行JIT编译
- Native stub是跟运行实例不相关，可以在各个V8 isolate之间共享的code stub，它们位于V8进程的native heap而不是isolate实例的managed heap。
- 与WasmHeapStubCompilationJob类似，GenerateCodeForWasmNativeStub函数接收的也是Machine Graph，其pipeline的配置如下：

```
MemoryOptimizationPhase
ComputeScheduledGraph()
SelectInstructions();
AssembleCode()
```

总结今天的内容： TurboFan 架构概览

- TurboFan在V8中的位置和所发挥的作用： AOT/JIT
- TurboFan的IR：层次化的节点定义和SON图
- TurboFan的Pipeline结构： Phase/PipelineImpl/PipelineData
- TurboFan的pipeline的5类不同的编译任务驱动

形成整体的
认识和了解

预告：

TurboFan如何一步一步编译一个JavaScript函数：

- Graph Build
- Optimize
- CodeGen

谢 谢

欢迎交流合作