GPUCC
An Open-Source GPGPU Compiler
A Preview

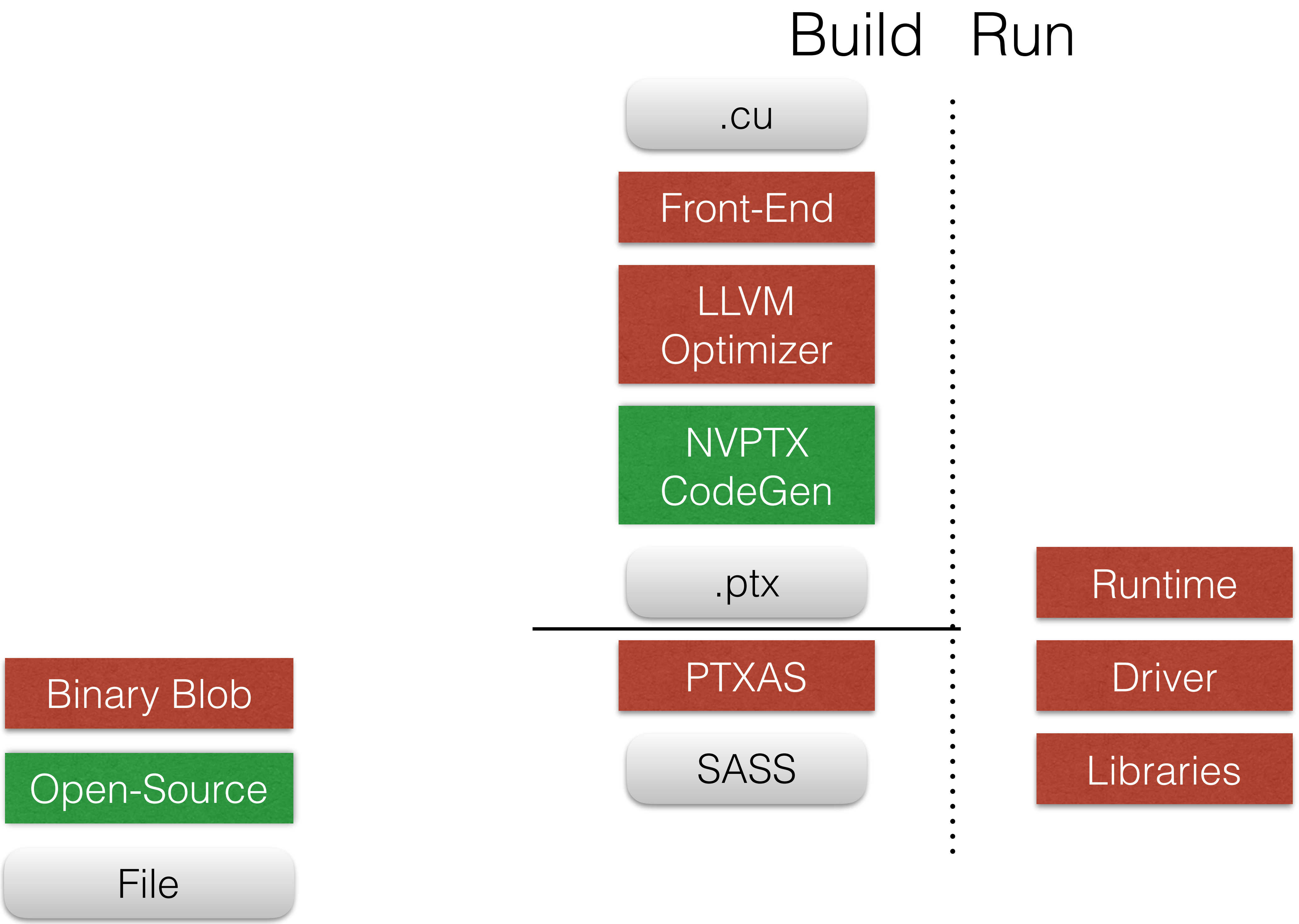
Eli Bendersky, Mark Heffernan, Chris Leary, Jacques Pienaar, Bjarke Roune, Rob Springer, Jingyue Wu, Xuetian Weng, Artem Belevich, **Robert Hundt (rhundt@google.com)**

Why Make an Open-Source Compiler?

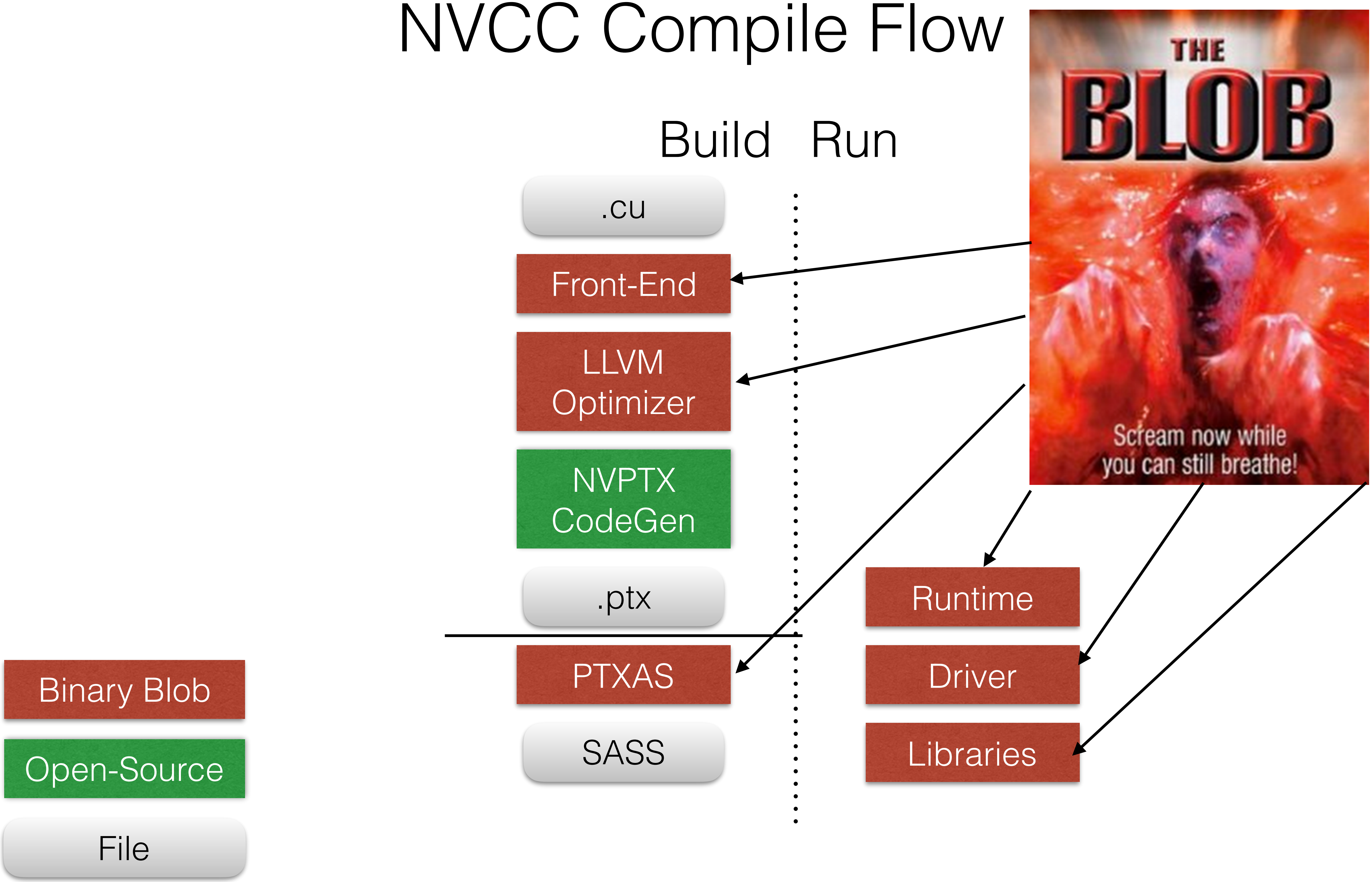
- **Security** - No binary blobs in the datacenter
- **Binary Dependencies** - Software updates become difficult
- **Performance** - We can always do better on our benchmarks
- **Bug Fix Time** - We can be faster than vendors
- **Language Features** - Incompatible development environments
- **Lock-In** - Nobody likes that
- **Philosophical** - We just want to do this ourselves

- ***Enable compiler research***
- ***Enable community contributions***
- ***Enable industry breakthroughs***

NVCC Compile Flow



NVCC Compile Flow



Challenge: Mixed-Mode Compilation

foo.cu

```
template <int N>
void host(float *x) {
  float *y;
  cudaMalloc(&y, 4*N);
  cudaMemcpy(y, x, ...);
  kernel<N><<<16, 128>>>(y);
  ...
}
```

```
template <int N>
__global__ void kernel(
    float *y) {
  ...
}
```

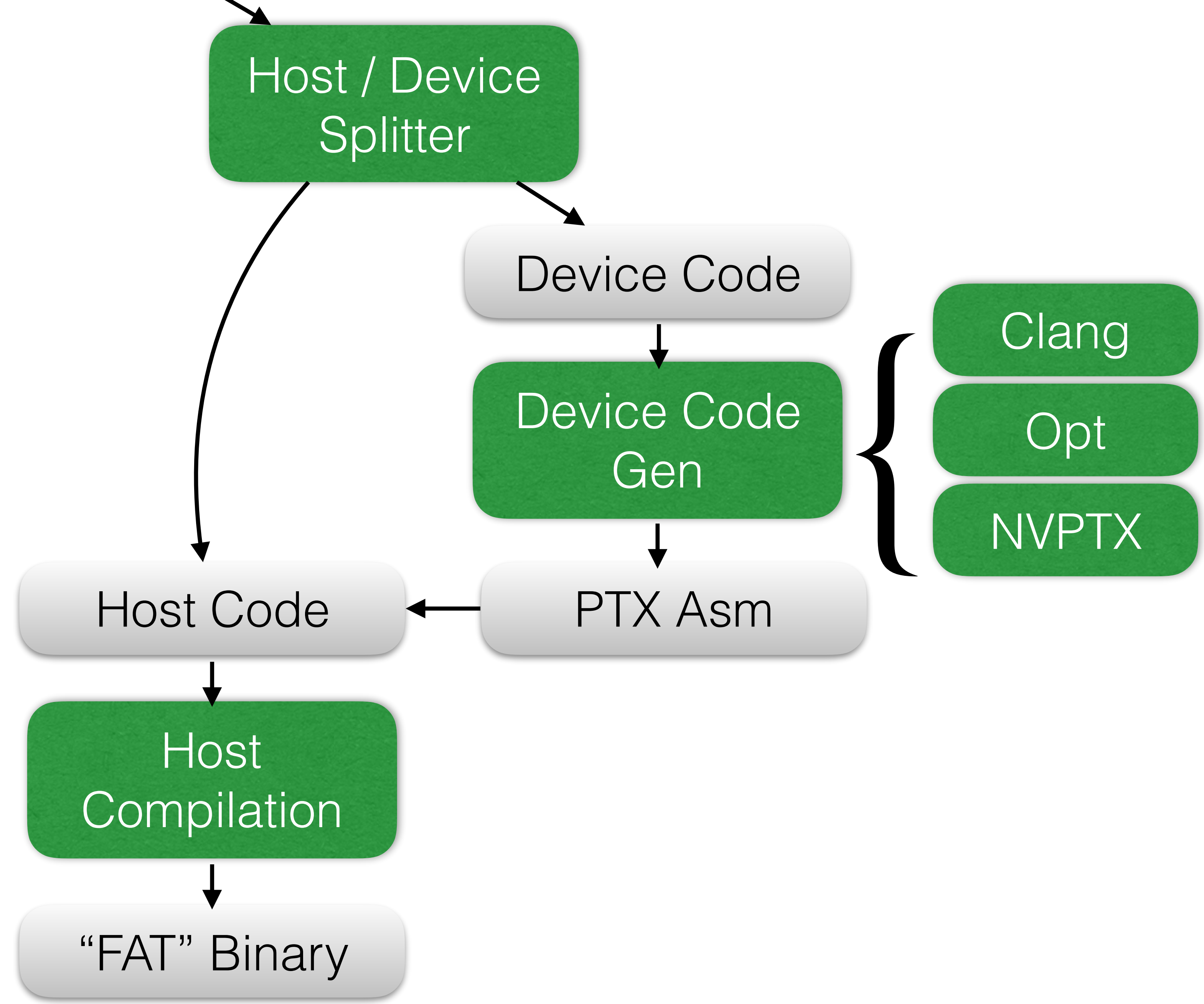
CPU/host

GPU/device

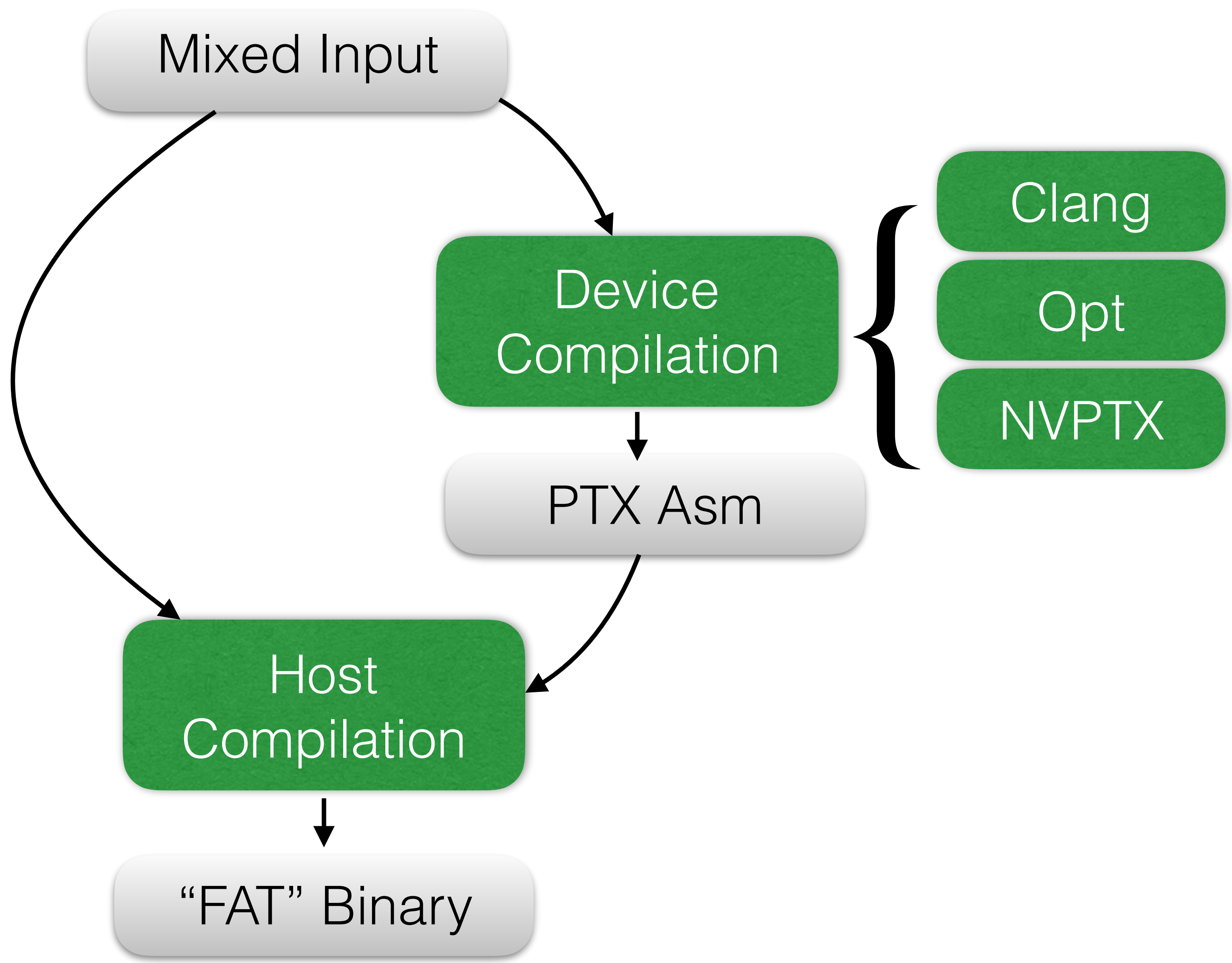
GPUCC Architecture - Current & Interim

```
template <int N>
__global__ void kernel(
    float *y) {
  ...
}

template <int N>
void host(float *x) {
  float *y;
  cudaMalloc(&y, 4*N);
  cudaMemcpy(y, x, ...);
  kernel<N><<<16, 128>>>(y);
  ...
}
```



GPUCC - Future Architecture (WIP)



- **Clang Driver instead of Code Splitting**
- **Faster Compile Time**
- **No Src-To-Src Translation**

Optimizations

- **Unrolling** (duh)
- **Inlining** (duh)
- **Straight-line scalar optimizations** (redundancies)
- **Inferring memory spaces** (use faster loads)
- **Memory space alias analysis** (it does help)
- **Speculative Execution** (divergence, predication)
- **Bypassing 64-bit divisions** (can be done in source, but…)
- **Heuristics changes** in other passes

See also:
Jingyue Wu, GPUCC, An Open-Source GPGPU Compiler
LLVM Dev Meeting, 2015

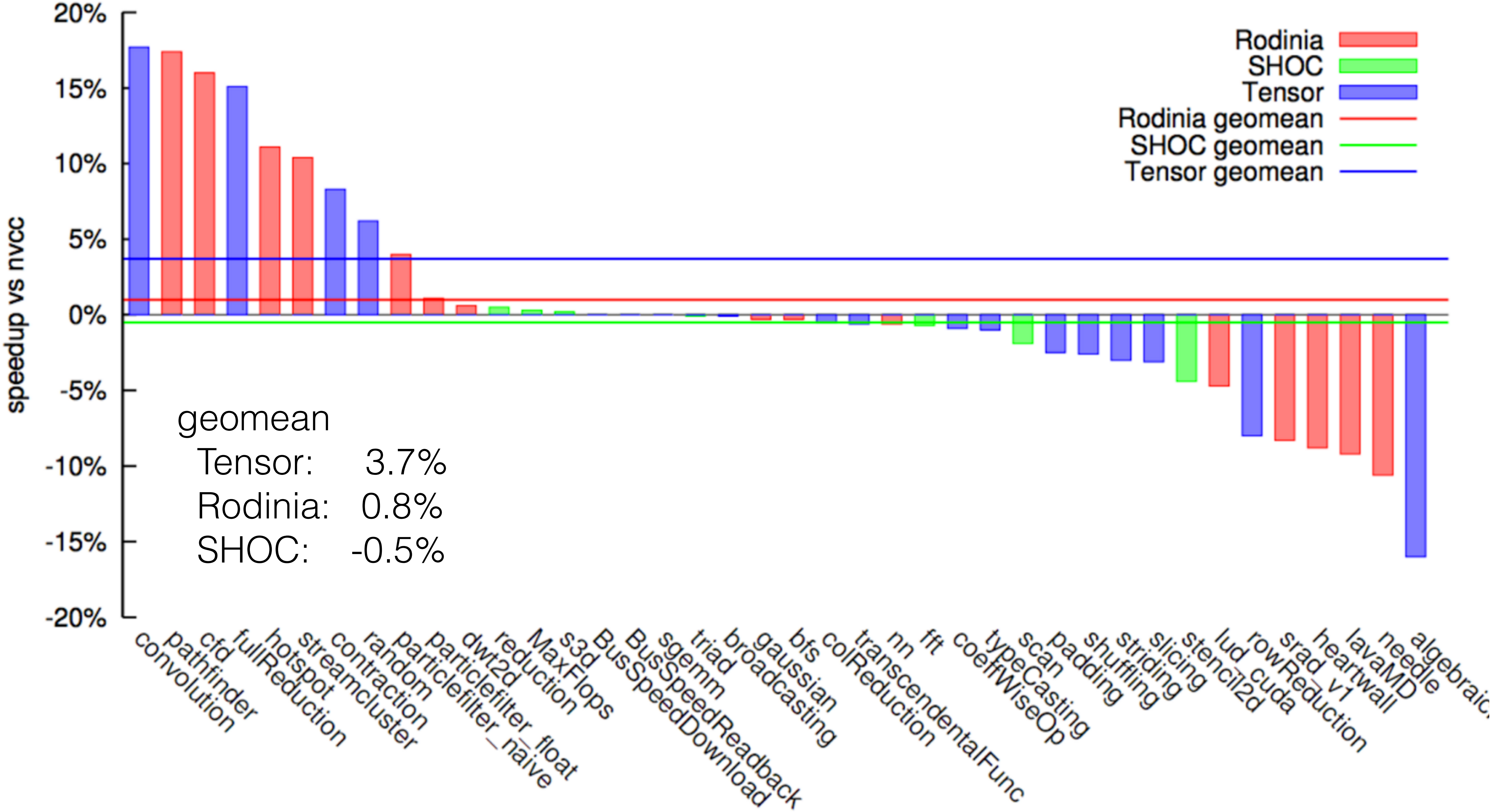
Runtime: StreamExecutor

- Compiler can an target CUDA runtime or StreamExecutor
- StreamExecutor: Thin abstraction around CUDA/OpenCL
- Advantages: C++, concise, type safe, better tooling, stable host code
- Open-Sourced with TensorFlow release

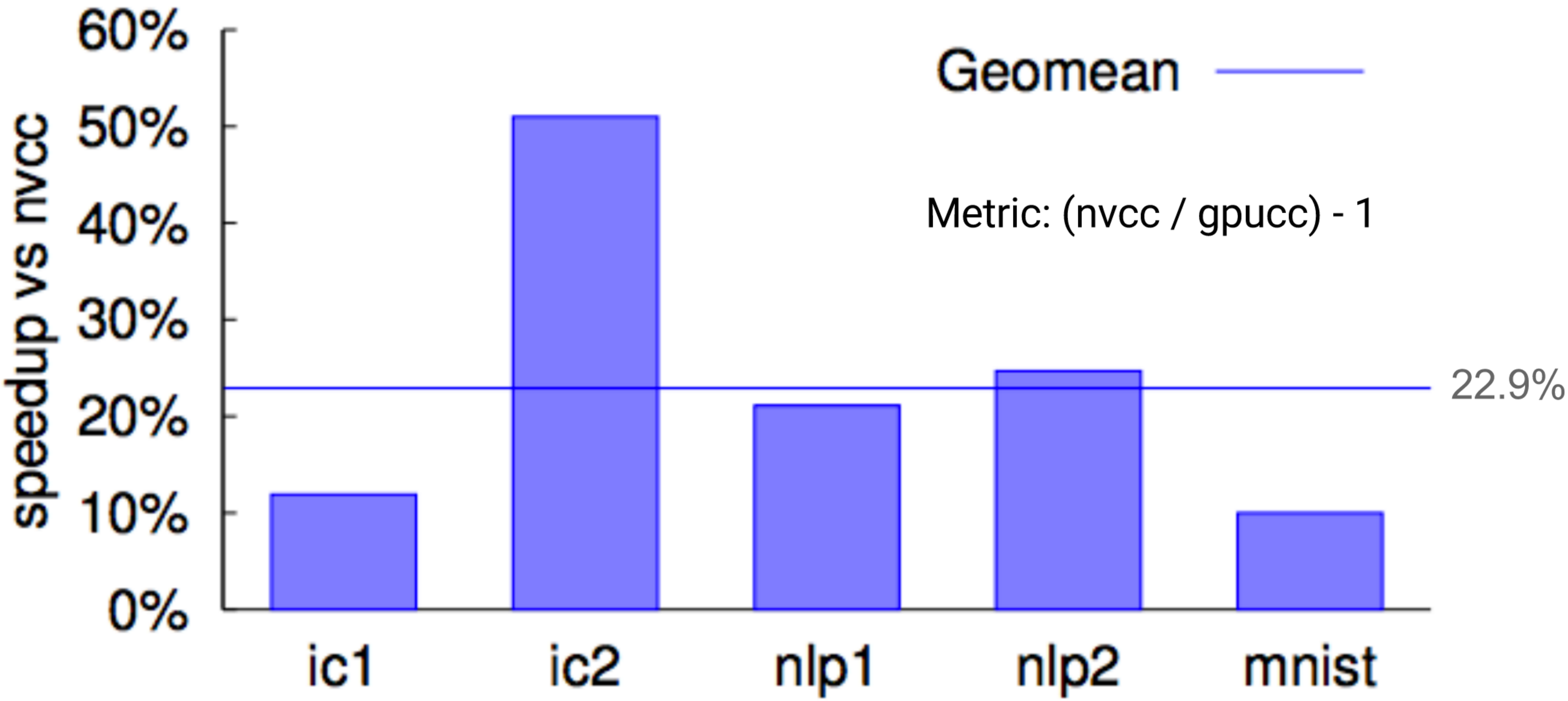
Evaluation

- End-to-End Benchmarks
  - **ic1**, **ic2**: Image Classification
  - **nlp1**, **nlp2**: Natural Language Processing
  - **mnist**: Handwritten Character Recognition
- Open-Source Benchmarks
  - **Rodinia**
  - **SHOC**
  - **Tensor**
- Machine Setup: **GPU NVidia Tesla K40c**
- Baseline: **NVCC v7.0**

Open-Source Benchmarks

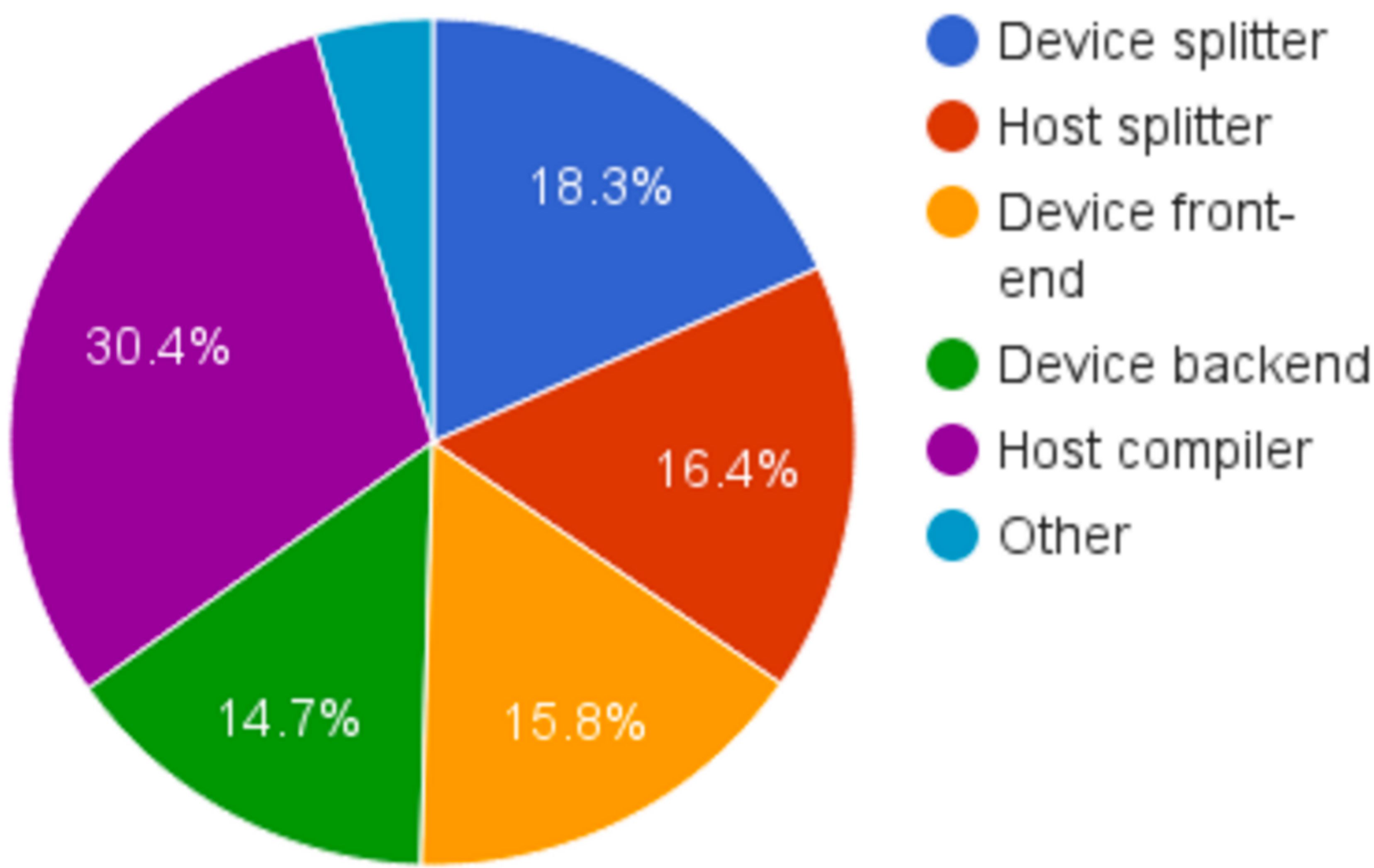


End-To-End Benchmarks



Compile Time

- **8%** faster than nvcc on average (per unit)
- **2.4x** faster for pathological compiles (eg., 109 secs vs 263 secs)
- Will be even faster after Clang integration



Libraries: FFT (geomean: 49%)

Routine	Speedup
1D C2C	39%
2D C2C	51%
3D C2C	66%
1D Batched C2C	18%
2D Batched C2C	33%
3D Batched C2C	40%
1D R2C	52%
2D R2C	37%
3D R3C	57%
1D Batched R2C	65%
2D Batched R2C	64%
3D Batched R2C	74%

Average Speedup, K40c, vs cuFFT 6.5

Libraries: Blas 1 (geomean: 21%)

Function	Speedup
ASUM	15.1%
AXPY	9.6%
COPY	14.6%
DOT	15.7%
IAMIN/IAMAX	17.2%
NRM2	25.8%
ROT	3.5%
ROTM	141.6%
SCAL	9.6%
SWAP	0.3%

Average Speedup, K40c, vs cuBLAS 6.5

Libraries: Blas 2 (geomean: 92%)

Function	Speedup
GEMV	8.3%
GBMV	136.5%
SYMV	51.2%
SBMV	368.9%
SPMV	99.4%
TRMV	177.8%
TBMV	160.6%
TPMV	165.1%
GER	1.3%
SYR	30.1%
SPR	62.1%
SYR2	20.1%
SPR2	51.5%
TRSV	2.1%
TBSV	334.2%
TPSV	191.7%

Average Speedup, K40c, vs cuBLAS 6.5

Libraries: Blas 3 (geomean: -20%)

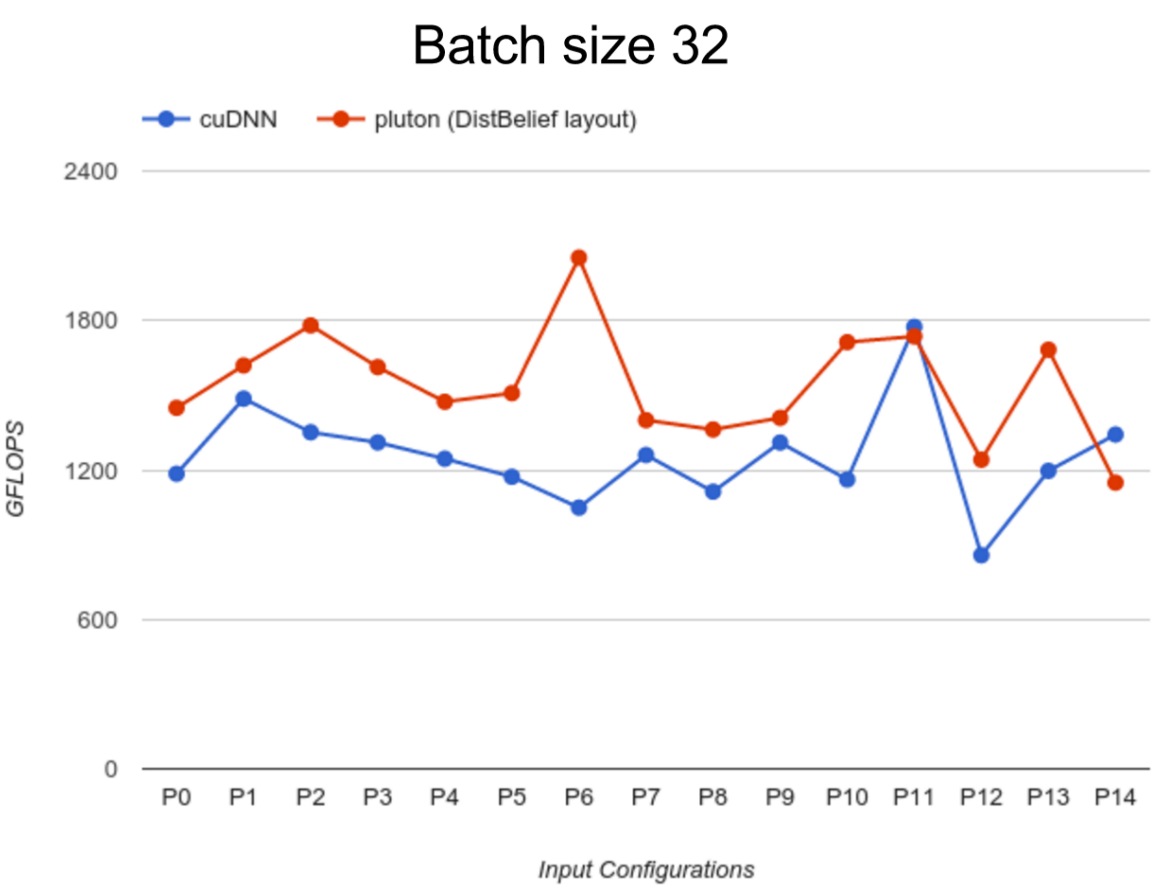
Function	Speedup
GEMM	-33.0%
TRMM	80.5%
SYMM	-11.8%
SYRK	-44.1%
SYR2K	-43.4%

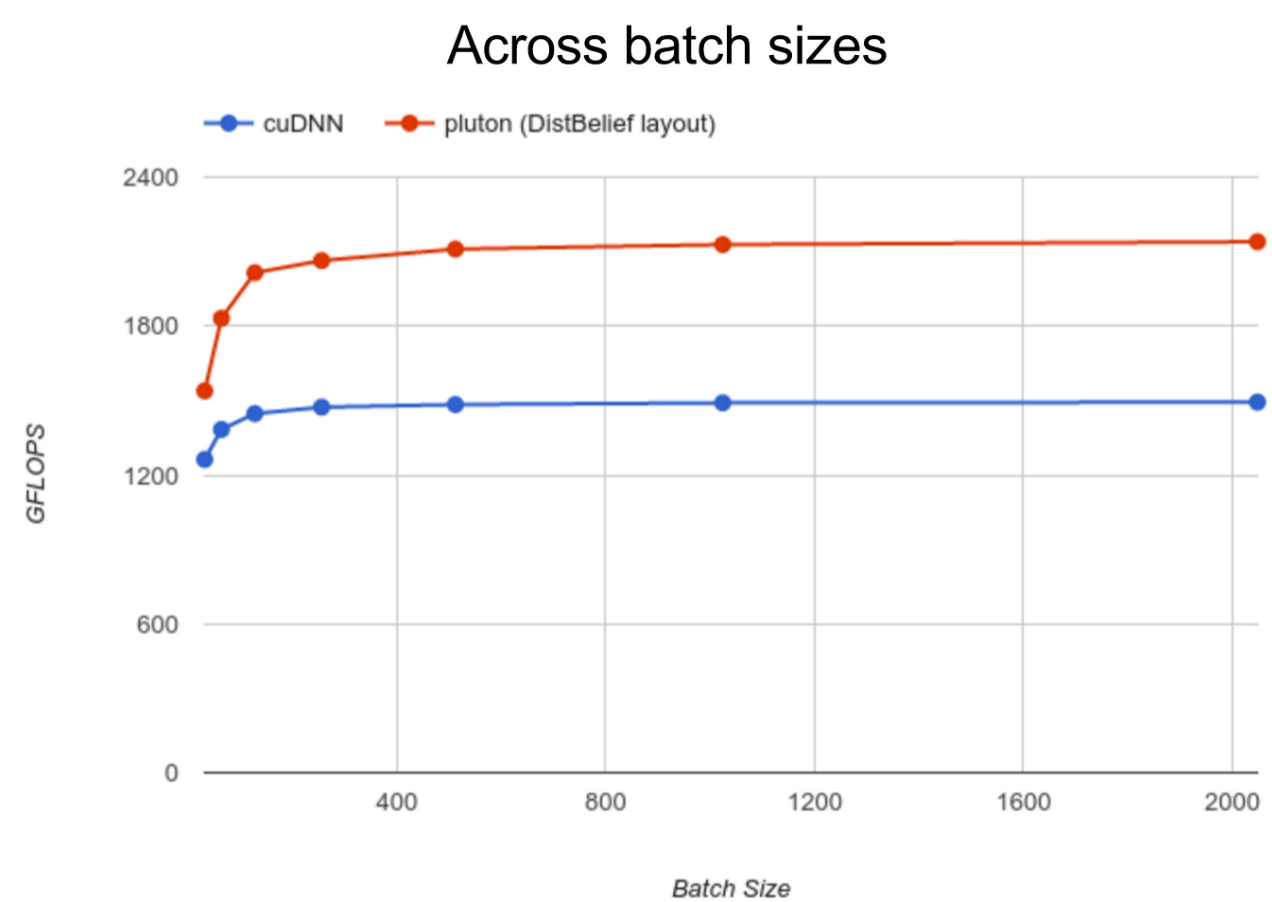
Lack of SASS-level Optimizations

Average Speedup, K40c, vs cuBLAS 6.5

Libraries: DNN, Forward Convolution (WIP)

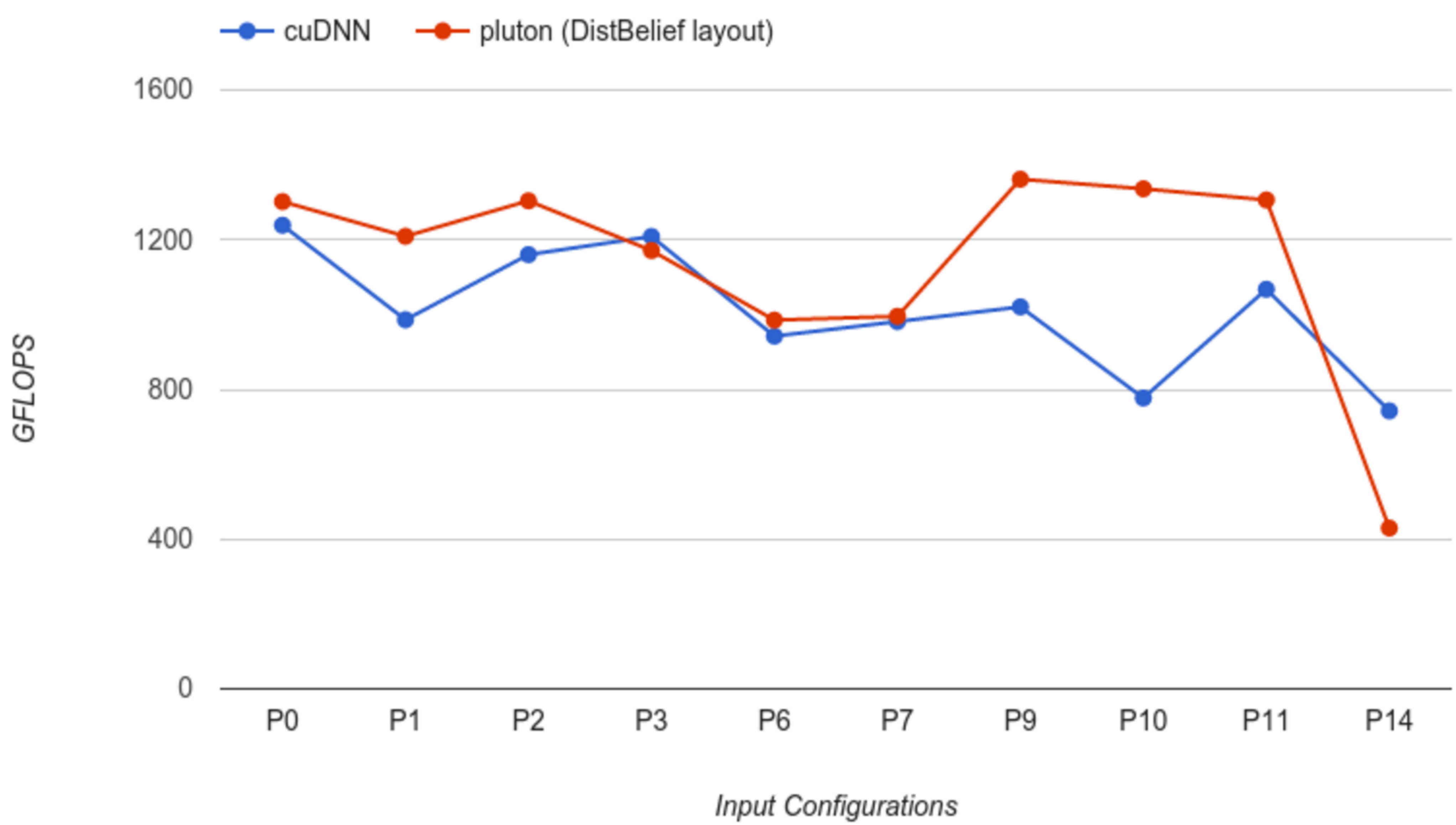
- **23%** better on batch size 32
- Up to **43%** better on larger batch sizes



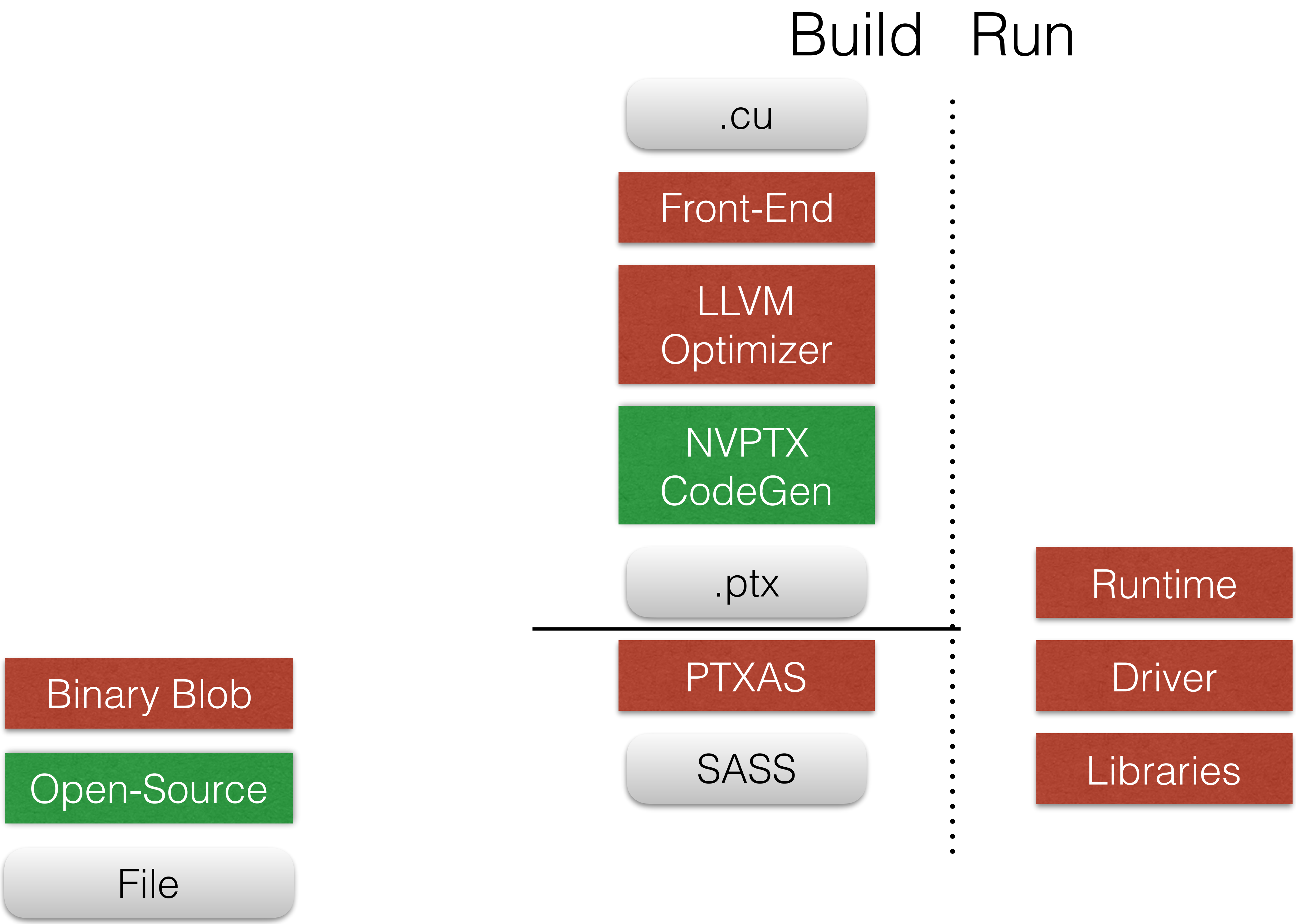


Libraries: DNN, Backward Convolution (WIP)

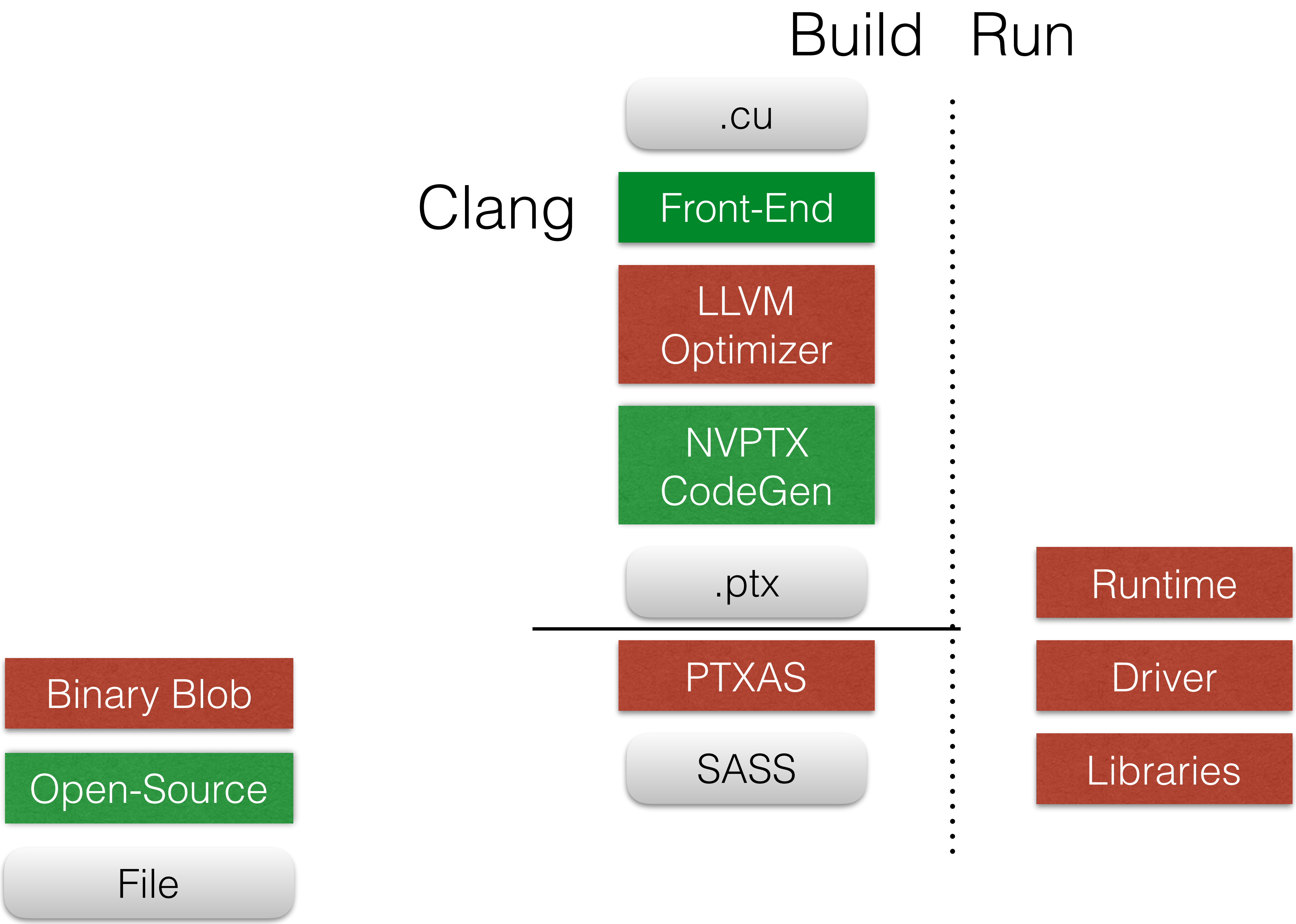
- **9%** better on batch size 32



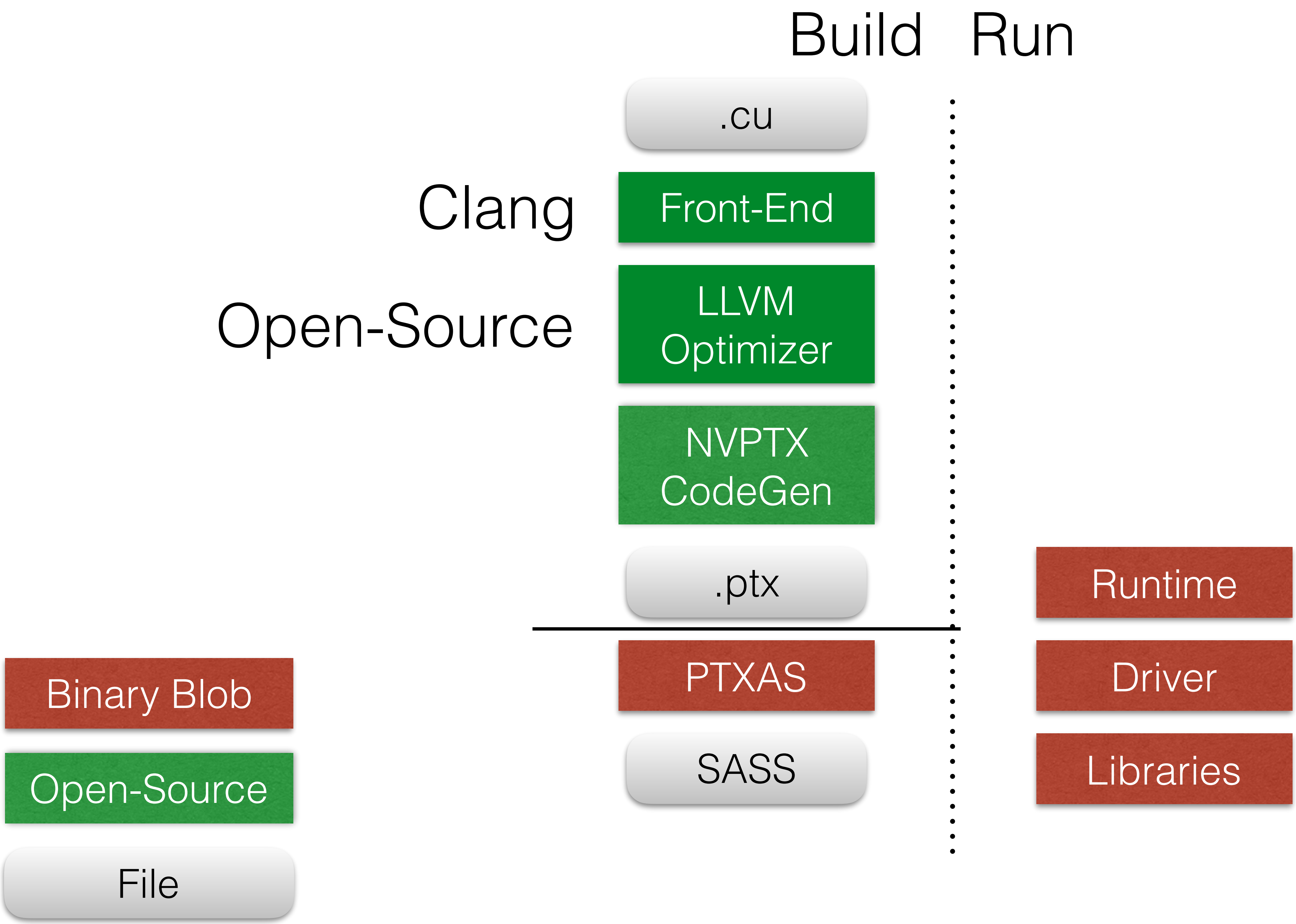
Recap: NVCC Compile Flow



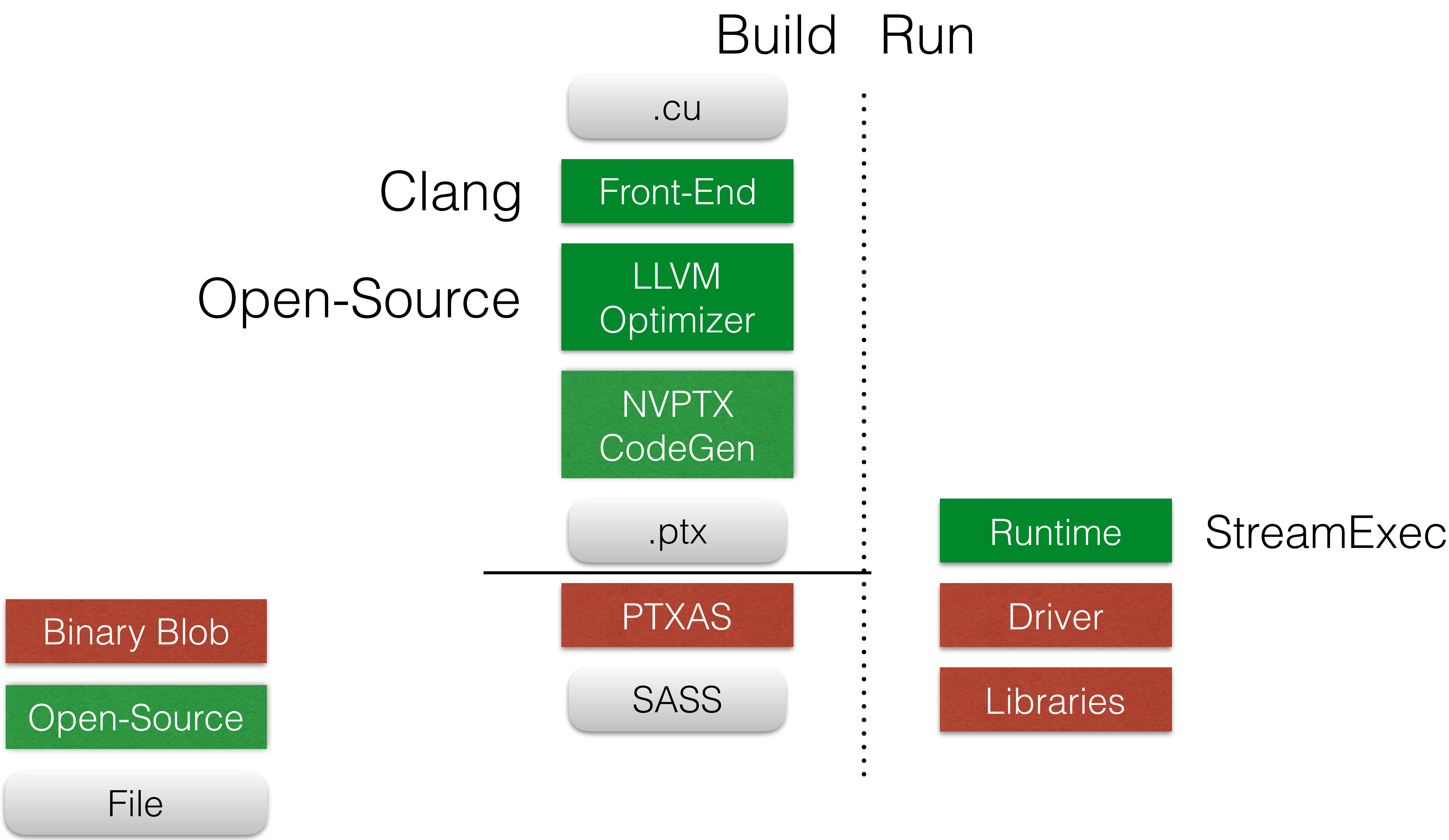
GPUCC Compile Flow



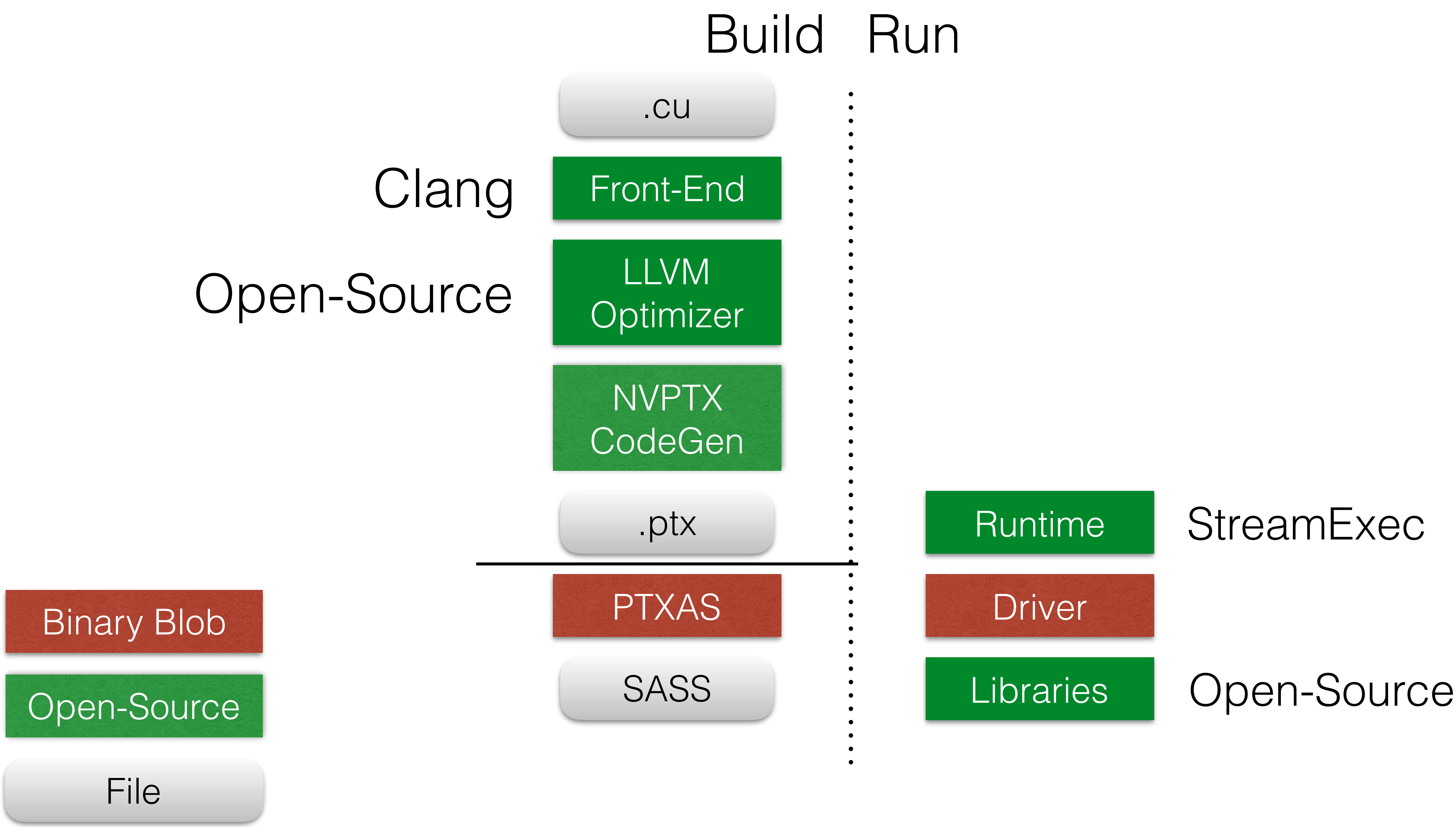
GPUCC Compile Flow



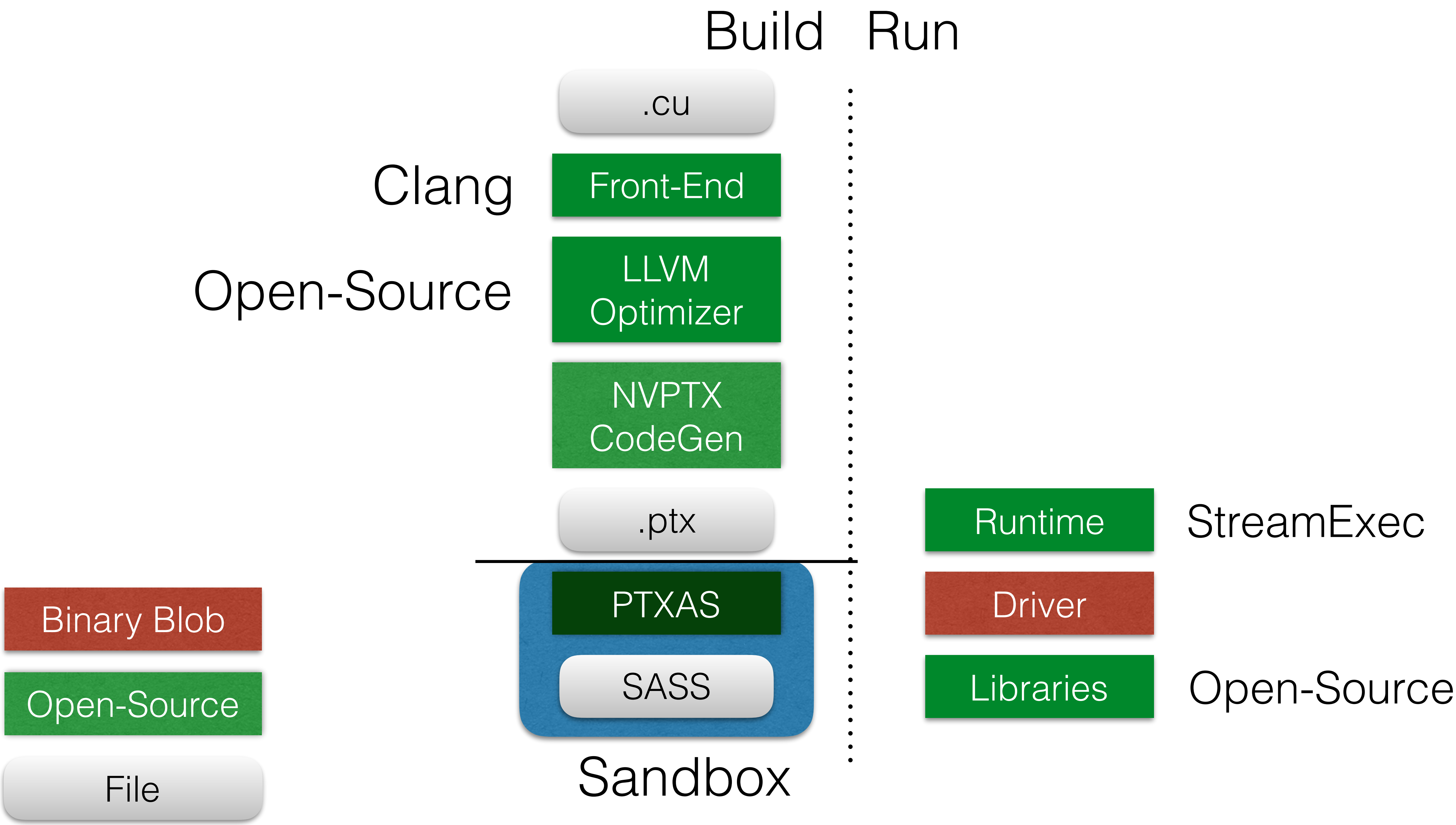
GPUCC Compile Flow



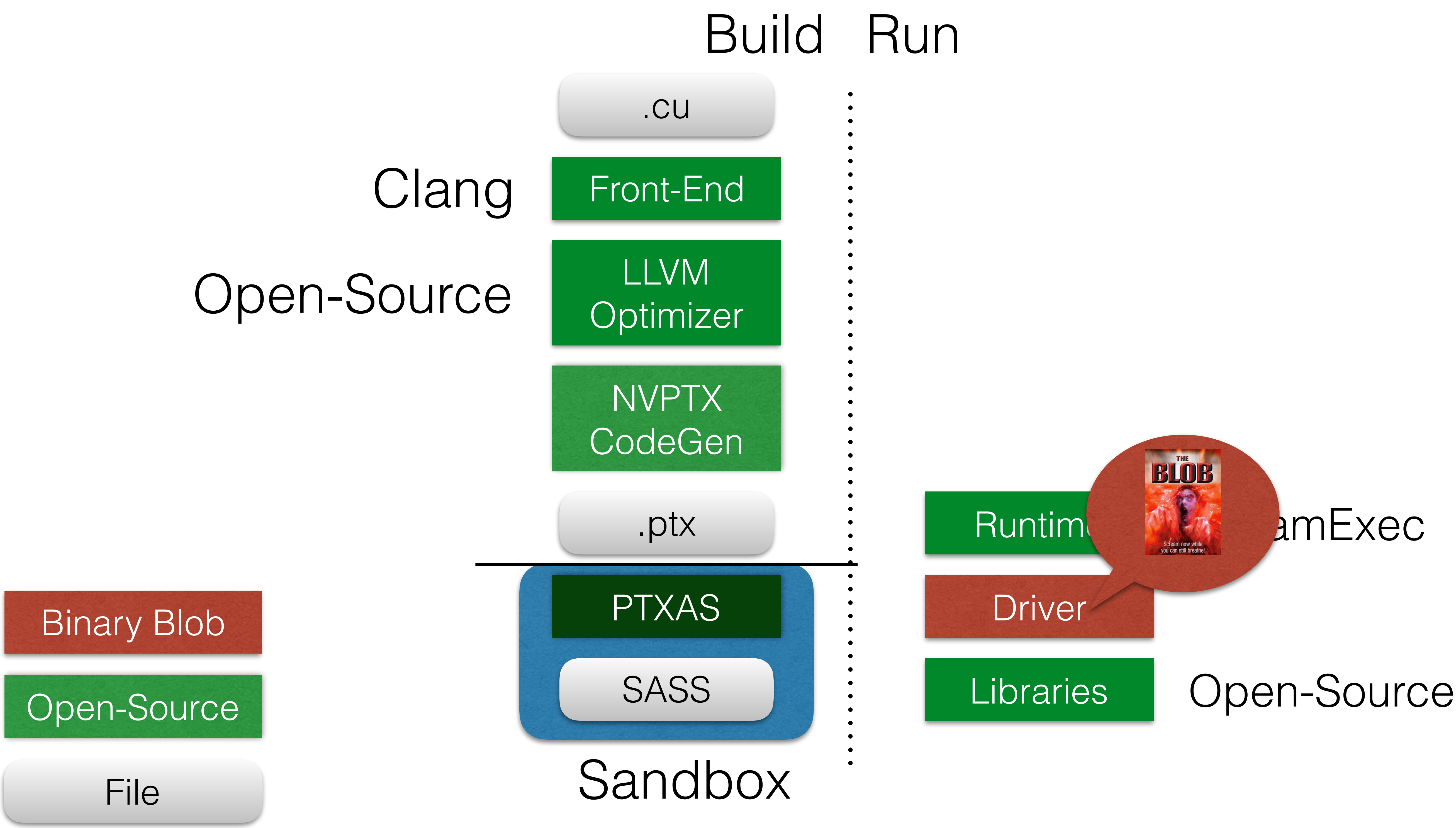
GPUCC Compile Flow



GPUCC Compile Flow



GPUCC Compile Flow



Summary

- Open-Source GPGPU Compiler (targets CUDA)
- Compilation to PTX, no SASS
- Performance on par for several benchmarks
- Compile time on par
- Supports modern language features
- High-performance libraries (FFT, BLAS, DNN soon)

- **Plan for release: March 2016**
- **Call for Participation!**