

# DEMISTING THE NVDLA<sup>®</sup> COMPILER

OSDT2018-北京  
中科重德智能科技 邱吉

# 概览

- NVDLA 是什么
- NVDLA 的软件栈
- 基本版NVDLA compiler
- 高阶版本：优化
- 总结

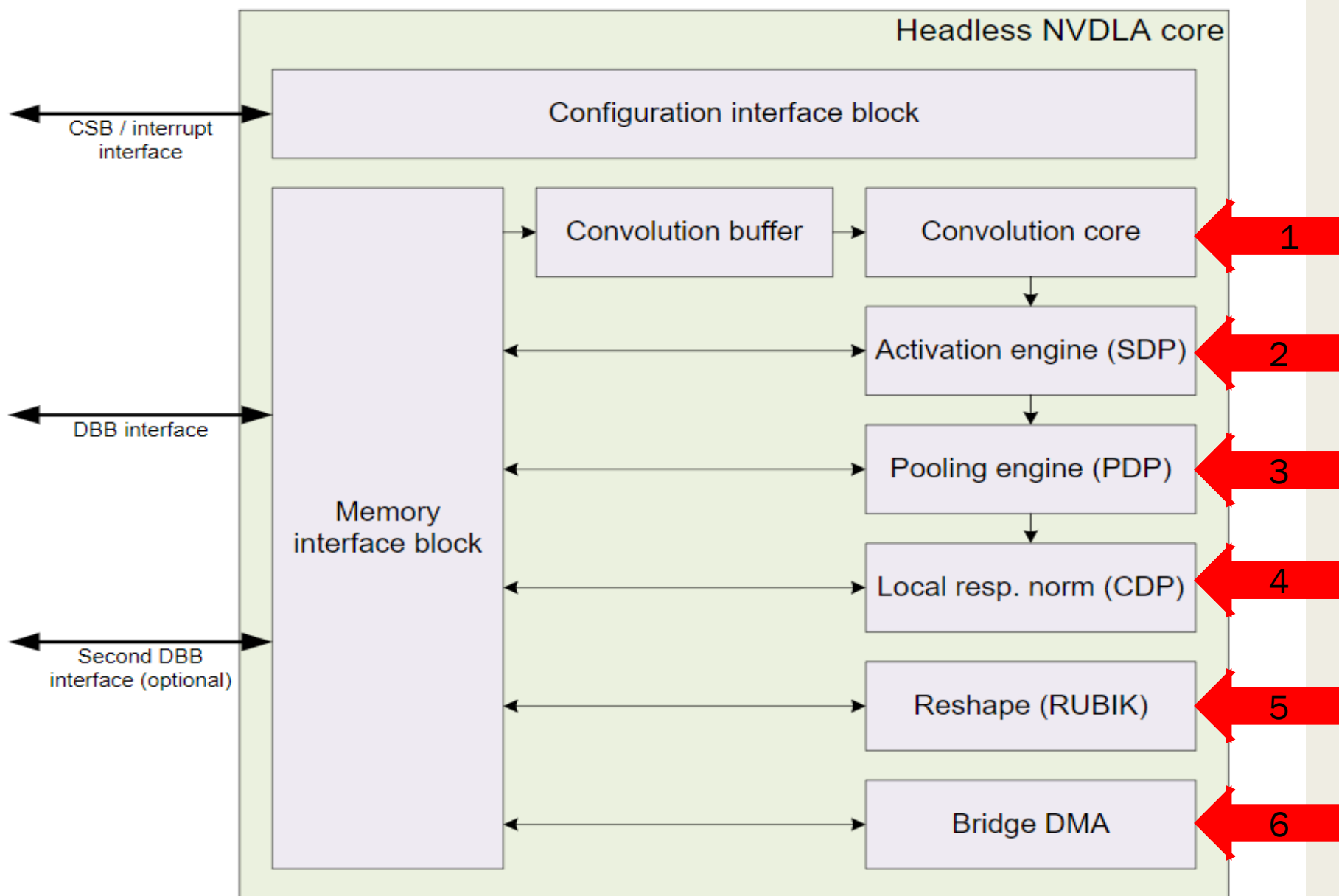
# 概览

- NVDLA 是什么
- NVDLA 的软件栈
- 基本版NVDLA compiler
- 高阶版本：优化
- 总结

# NVDLA是什么

- NVDLA是由NVIDIA公司推出的开源深度学习推理加速器
- 端侧、inference
- 项目状态

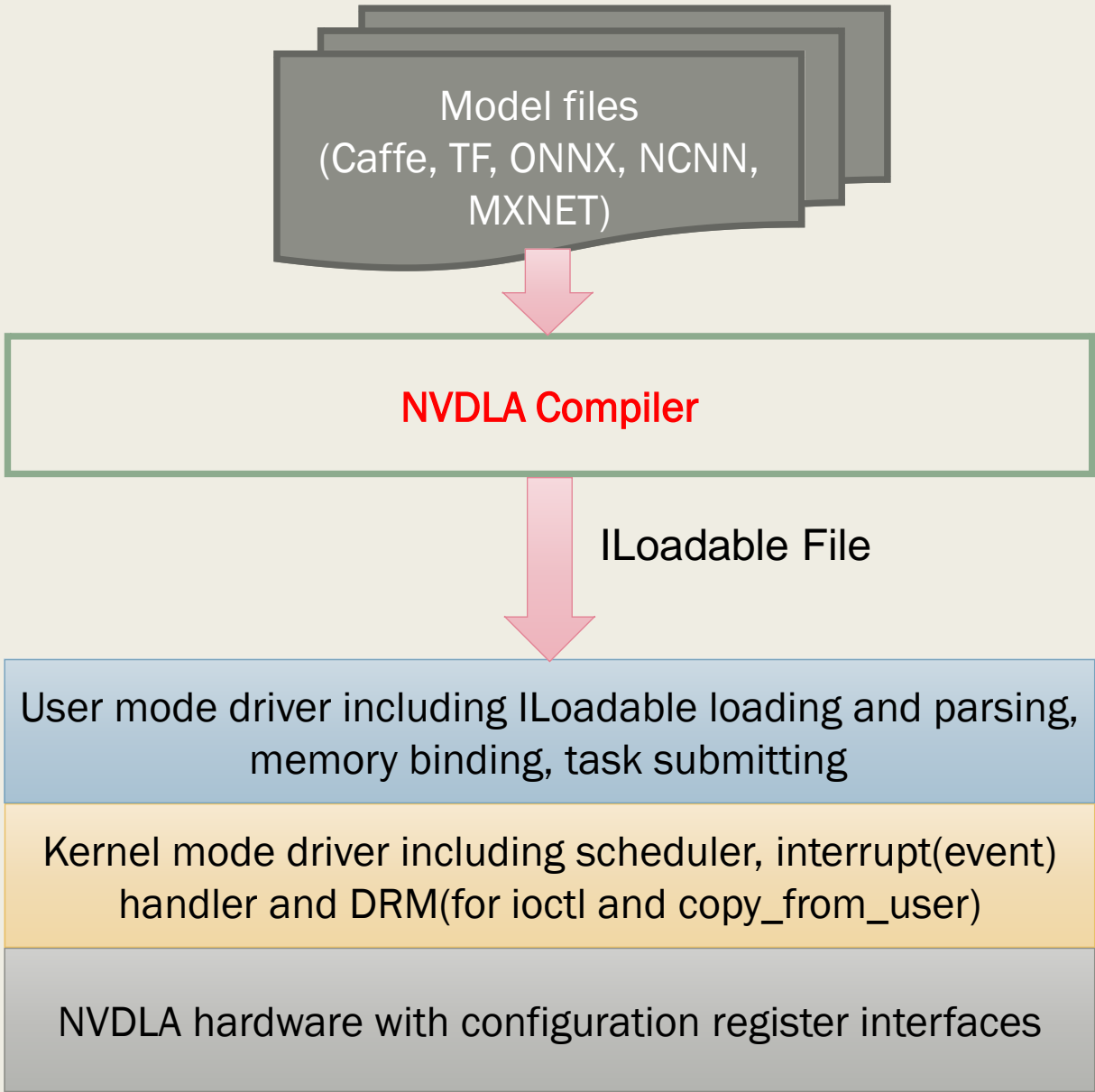
GitHub repo	Last push	Issues
<a href="https://github.com/nvdla/sw">https://github.com/nvdla/sw</a>	8/7	67
<a href="https://github.com/nvdla/hw">https://github.com/nvdla/hw</a>	4/19	113
<a href="https://github.com/nvdla/vp">https://github.com/nvdla/vp</a>	8/22	11
<a href="https://github.com/nvdla/doc">https://github.com/nvdla/doc</a>	10/10	7



# 概览

- NVDLA 是什么
- NVDLA 的软件栈
- 基本版NVDLA compiler
- 高阶版本：优化
- 总结

Model files  
(Caffe, TF, ONNX, NCNN,  
MXNET)



```
graph TD; A["Model files (Caffe, TF, ONNX, NCNN, MXNET)"] --> B["NVDLA Compiler"]; B -- "ILoadable File" --> C["User mode driver including ILoadable loading and parsing, memory binding, task submitting"]; C --> D["Kernel mode driver including scheduler, interrupt(event) handler and DRM(for ioctl and copy_from_user)"]; D --> E["NVDLA hardware with configuration register interfaces"];
```

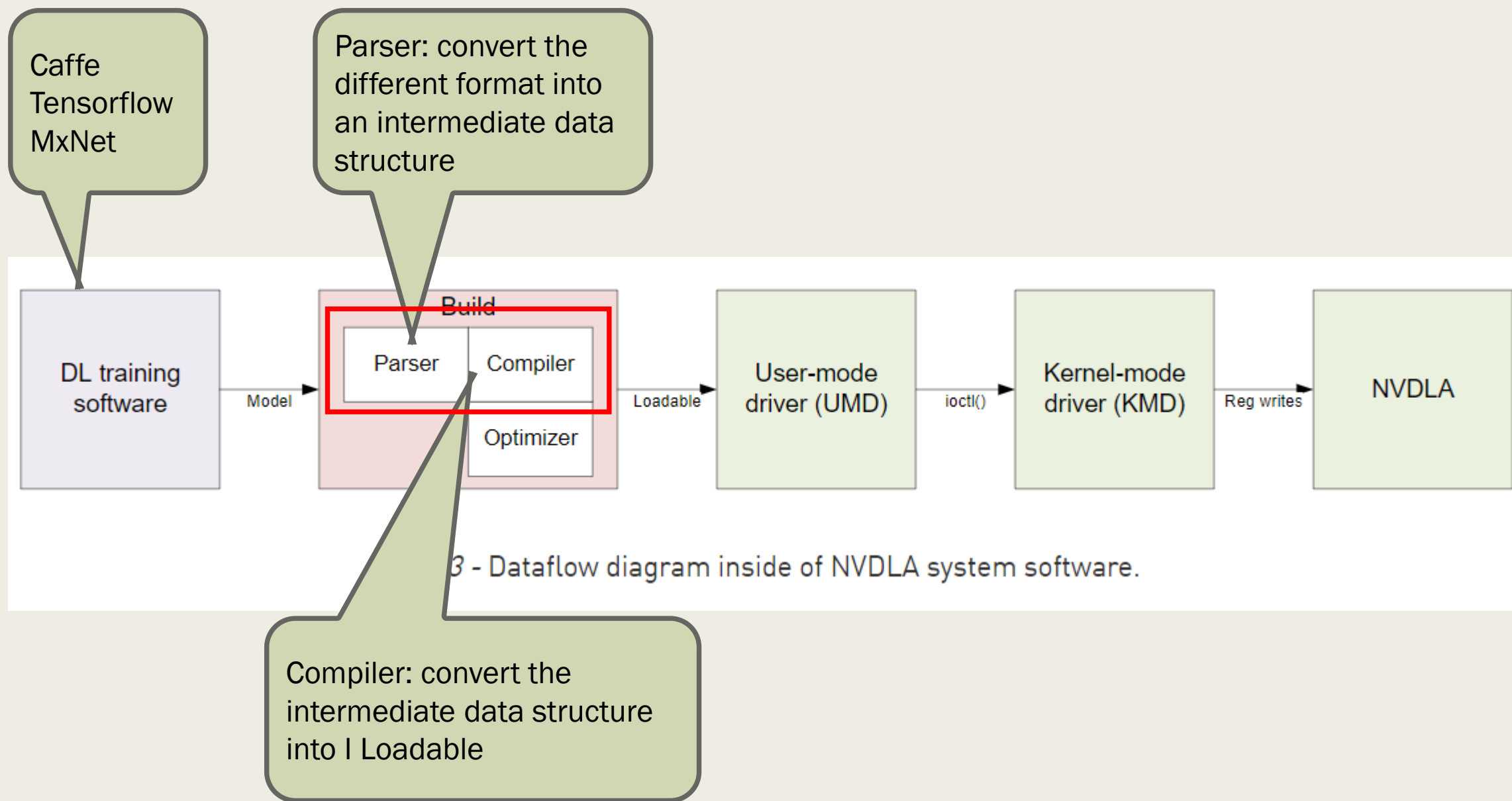
**NVDLA Compiler**

ILoadable File

User mode driver including ILoadable loading and parsing,  
memory binding, task submitting

Kernel mode driver including scheduler, interrupt(event)  
handler and DRM(for ioctl and copy\_from\_user)

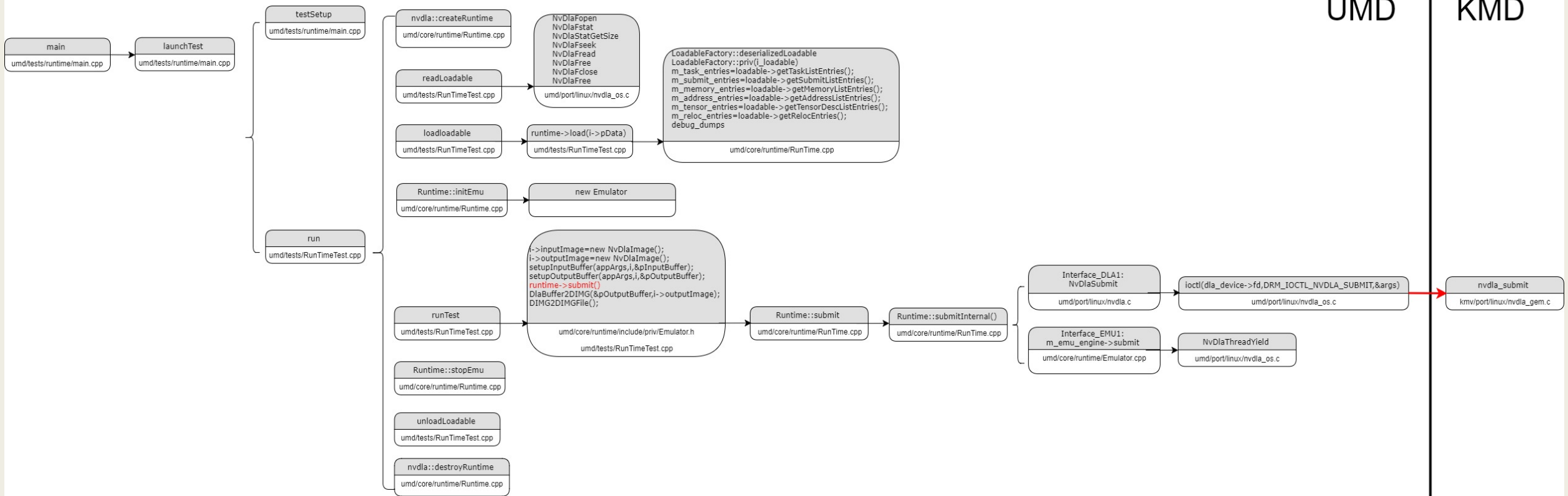
NVDLA hardware with configuration register interfaces





UMD

KMD



# 为什么要探究nvdla compiler

- 在GitHub的Nvdla\_compiler只是个二进制和语焉不详的介绍
- 玩主跳票发布更新的compiler和开源
- 67 sw issues, 27 compiler titled issues, still 11 opened

## 2018-H1

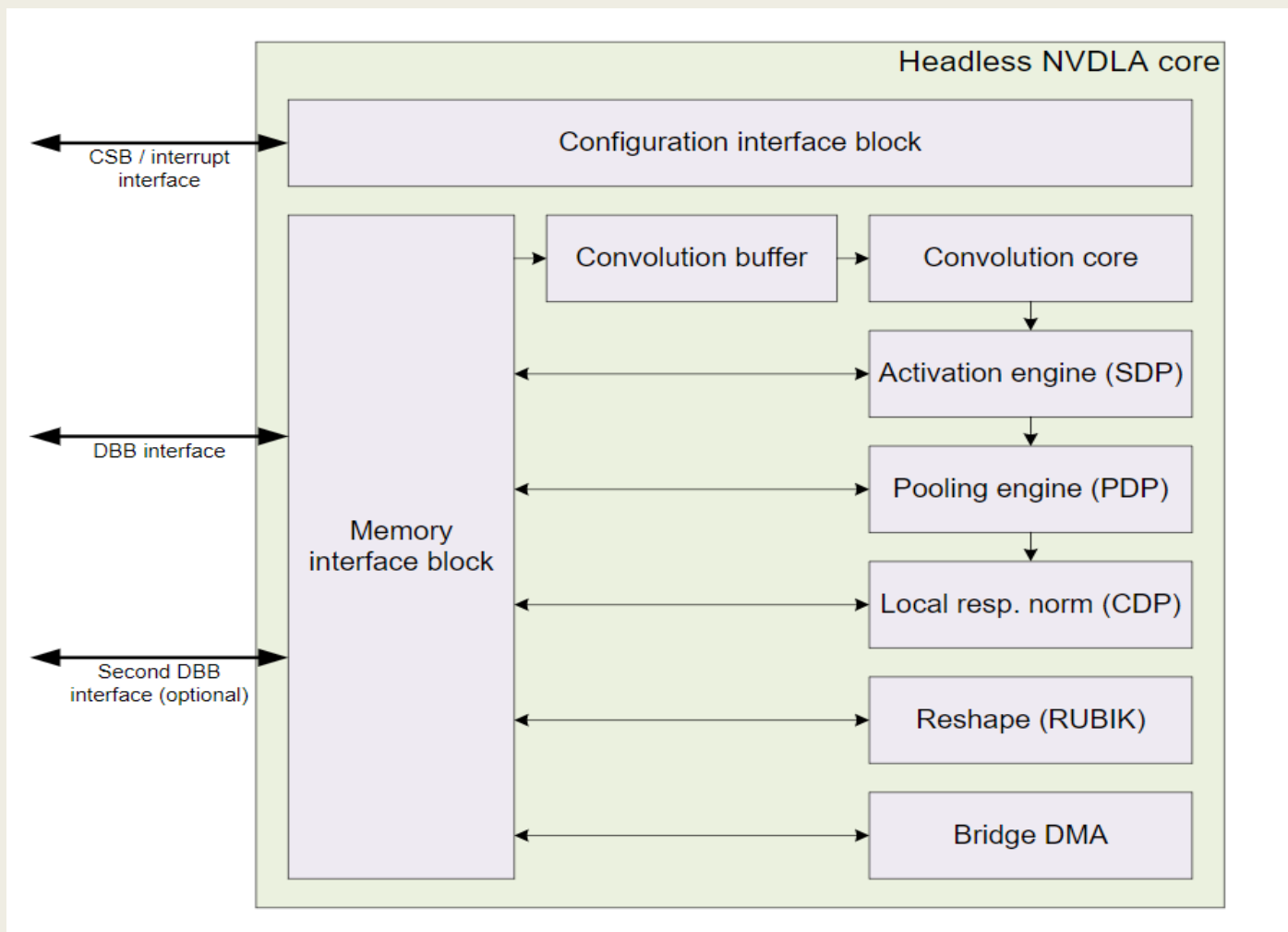
The third set of deliverables will see scalable solutions: the user can set parameters to create the exact configuration that best match his or her needs. A test bench, test suite, and FPGA support are added to allow for rigorous testing needed for high-confidence tape outs.

Documentation	Hardware	Software
Software architecture document	Small config Verilog RTL released	FPGA support for accelerating software development
	RTL support for fine grained configuration control	Customizable compiler
	UVM testbench validation of custom configurations	NVDLA compliance test suite
		TensorRT and all supported frameworks

# 怎么探究nvdla compiler

- 第一阶段
  - RDTFSC
  - Logging
  - Logging
- 第二阶段：发现参照系
  - [https://github.com/icubecorp/nvdla\\_compiler](https://github.com/icubecorp/nvdla_compiler)
  - 来自icubecorp（深圳中微电科技公司）
  - 9/17-10/15日提交代码后至今再也没有更新
  - issues : 4

# 需要了解的NVDLA硬件细节



- 6个功能单元覆盖了CNN的常见计算类型
- 在不存在数据依赖的情况下6个功能单元可以并行工作
- 每个功能单元都具有两个独立的运算部件和缓存，可以实现一个部件计算，另外一个部件等待访存的双缓冲调度
- SDP层可以跟Conv层fuse

# Convolution (Conv)

- Weight and input have different memory format
- Compressed weight has difference memory format
- HW/SW interface for size programming (software split convolution)
- Winograd
- Full connection
- Reference:
  - <http://nvdla.org/hw/format.html>

# Single Data Point Processor (SDP)

- Linear or non-linear individual data points
- LUT-programming
- Relu, Sigmoid, BN, bias add, elu, tanh

# Planar Data Processor (PDP)

- Spatial data
- maximum-pooling, minimum-pooling, and average-pooling

# Cross-channel Data Processor (CDP)

- LRN
- operates on channel dimensions

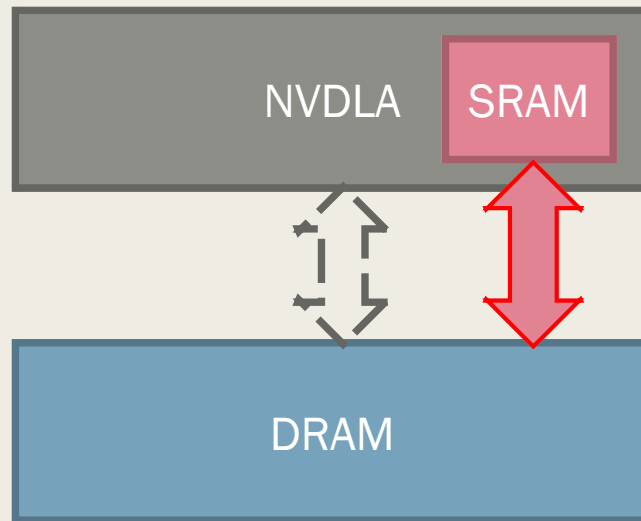


# Data Reshape Engine (RUBIK)

- Rubik operations
- Transpose

# Bridge DMA (BDMA)

- Move data from external DRAM into on-chip SRAM



# 概览

- NVDLA 是什么
- NVDLA 的软件栈
- 基本版NVDLA compiler: 3件事
- 高阶NVDLA compiler: 再加1件事
- 总结

# Flow of the Icube's NVDLA compiler



# Icube's NVDLA compiler—警

- 17K+行C++代码
- 目录结构
  - UMD: 直接从nvdla/sw/umd拷贝过来
  - tools:
    - Caffe2fb: NCNN network to NVDLA flatbuf
      - caffe\_layer: parse and transform input
      - nvdla\_layer: format target layers
      - list\_parser: build target data structure

# LoadNetwork

- 读取网络结构文件和权重文件
  - *NCNN format*
  - *.param and .bin*
- 生成内部数据结构
  - 移植了NCNN的src/layer目录下，不同类型layer param和bin文件的解析函数，生成caffe\_net（实际上是NCNN标准的内部网络数据结构），层的名字是caffe的命名方式
  - **build nvdla\_layers**: 把network转成新的layer向量，层的名字变成Nv前缀的命名方式, 形成后续生成所有list entry的基础 \*

# Generate FlatBuf

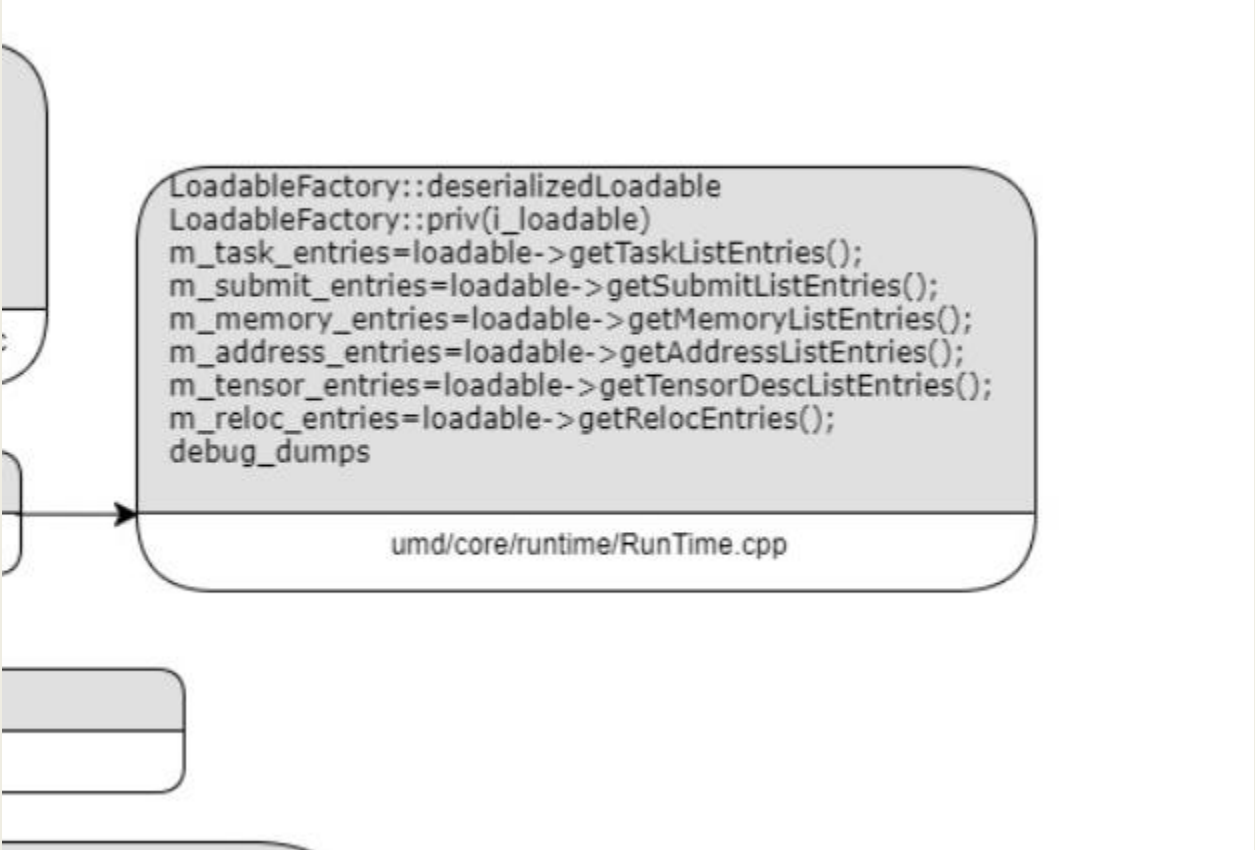
- FlatBuffers是一种由谷歌开发的跨平台序列化文件格式和库
- 按照NVDLA的UMD所规定的loadable格式进行存储
  - **接口规范**
    - `nvdla/sw/umd/core/common/include/priv/Loadable.h`

# FillLists-8个List

- 按顺序依次为
  - Task \*
  - Submit
  - Memory \*
  - Address
  - Event: dummy
  - TensorDesc: Input and Output 描述
  - Symbol: malloc memory for blobs and memcpy \*
  - Reloc: dummy
- 对应list\_parser下的文件



# 为什么是8个list



```
LoadableFactory::deserializedLoadable
LoadableFactory::priv(i_loadable)
m_task_entries=loadable->getTaskListEntries();
m_submit_entries=loadable->getSubmitListEntries();
m_memory_entries=loadable->getMemoryListEntries();
m_address_entries=loadable->getAddressListEntries();
m_tensor_entries=loadable->getTensorDescListEntries();
m_reloc_entries=loadable->getRelocEntries();
debug_dumps
```

umd/core/runtime/RunTime.cpp

# A demo: LeNet

id	Type	Name
0	NvInput	Input
1	NvConv	Conv
2	NvSDP	SDP
3	NvPDP	PDP
4	NvConv	Conv
5	NvSDP	SDP
6	NvPDP	PDP
7	NvConv	Conv
8	NvSDP	SDP
9	NvSDP	SDP
10	NvConv	Conv
11	NvSDP	SDP
12	NvSoftmax	Softmax

其他支持的type类型: NvCDP, NvRUBIK, NvBDMA

# TaskList build-1

- Nvdla\_task是用户空间提交到KMD的任务信息载体
- 两种Task：
  - DLA task: 通过UMD submit到KMD, 最后配置NVDLA采用硬件完成
  - Emu task: NVDLA不能做softmax, 所以在UMD中模拟
- 一个DLA task可以包含若干计算图中非softmax节点
- 一个Emu task可以包含若干连续的softmax节点
- 通过一个状态机来生成tasklist

# TaskList build-2

## ■ Heuristic:

1. 遍历每一个非Input的layer, 对layer(i)
2. 如果是softmax层, 且前一层不是softmax, 则置位emu\_flag, 设置emu\_start\_index=i;
3. 如果是softmax层, 且前一层是softmax, task\_count++
4. 如果不是softmax层, 且前一层是softmax 或者input, 则置位dla\_flag, 设置dla\_start\_index=i
5. 如果不是softmax层, 且前一层不是softmax 或者input, task\_count++
3. 当emu\_flag置位, 且当前层不是softmax时, 将当前层跟之前的层隔断, 之前的层构建一个DLA task, 然后将dla\_flag和tak\_count清0
4. 当emu\_flag置位, 且当前层是softmax, 同时当前层是网络的最后一层, 则构建一个emu task, 然后将emu\_flag和task\_count清0
4. 当dla\_flag置位, 如果当前层不是softmax, 将当前层跟之前的层隔断, 之前的层则构建一个emu task, 然后将task\_count和emu\_flag清0

# Tasklist build-3 result

id	Type	Name
0	NvInput	Input
1	NvConv	Conv
2	NvSDP	SDP
3	NvPDP	PDP
4	NvConv	Conv
5	NvSDP	SDP
6	NvPDP	PDP
7	NvConv	Conv
8	NvSDP	SDP
9	NvSDP	SDP
10	NvConv	Conv
11	NvSDP	SDP
12	NvSoftmax	Softmax

Task0 Task1

# SubmitList build

- 把每个task的id依次放入Submit list中

# MemoryList build -1

- 按照nvdla layer的顺序, 解析每一个layer, 给layer所包含的变量分配Memorylist entry, 记录:
  - 线性增长的mem\_id作为唯一标识, 后续的编译、UMD、KMD都要使用\*
  - Size: n/c/h/w for input/filter/weight
  - Stride: line/ plane/ surf
    - Ref to: <http://nvdla.org/hw/format.html>
- 按照task list的顺序, 给每个task增加用于描述属性的数据结构的memorylist entry
- 将mem\_id序号反填回每一个task中, 填充内容包括:
  - 所有layer的Memorylist entry
  - 该task的6个描述性数据结构

# MemoryList build-2

每个Task对应的6个描述性数据结构:

Task Type	Entry Name	Data Structure(according to NVDLA's API)
DLA task	task_*_network_desc	dla_network_desc
	task_*_dep_graph *	dla_common_op_desc
	task_*_op_list *	dla_operation_container
	task_*_surf_list *	dla_surface_container
	Lut	4K buffer
	Null	4K buffer
Emu Task	task_*_network_desc	emu_network_desc
	task_*_op_list	emu_operation_container
	task_*_op_buffer_list	emu_operation_buffer_container
	null	4K buffer
	null	4K buffer
	null	4K buffer

带\* 的memory list entry 的size要乘以该task的layer数



# MemoryList build -3

- 将mem\_id序号反填回每一个task中，填充内容包括：
  - 所有layer的Memorylist entry
  - 该task的6个描述性数据结构

MemoryList  
Entry for each  
layer's each  
variable

6 descs for DLA  
task

MemoryList  
Entry for each  
layer's each  
variable

6 descs for  
EMU task

# AddressList build

- 依次把每个memory list entry的mem\_id, size, offset依次放入Address list中

# TensorDescList build

- 对NvInput的层，构建名为data的描述符
- 对最后一层，构建名为probe的描述符
- 把对应层的信息填入描述符

# SymbolList building

- 遍历所有层，对conv和sdp类型的层，分配和拷贝weight
- 遍历所有task

Task Type	Entry Name	Comment
DLA task	task_*_network_desc	Non-trival
	task_*_dep_graph	<b>Special for fill prefetch and parent layers for each layer *</b>
	task_*_op_list	Fill for all the layers
	task_*_surf_list	Fill for all the layers
	Lut	4K
	Null	4K
Emu Task	task_*_network_desc	Softmax specific
	task_*_op_list	Softmax specific
	task_*_op_buffer_list	Softmax specific
	null	4K
	null	4K
	null	4K

# 概览

- NVDLA 是什么
- NVDLA 的软件栈
- 基本版NVDLA compiler: 3件事
- 高阶NVDLA compiler: 再加1件事
- 总结

# 高阶NVDLA\_Compiler-TODO

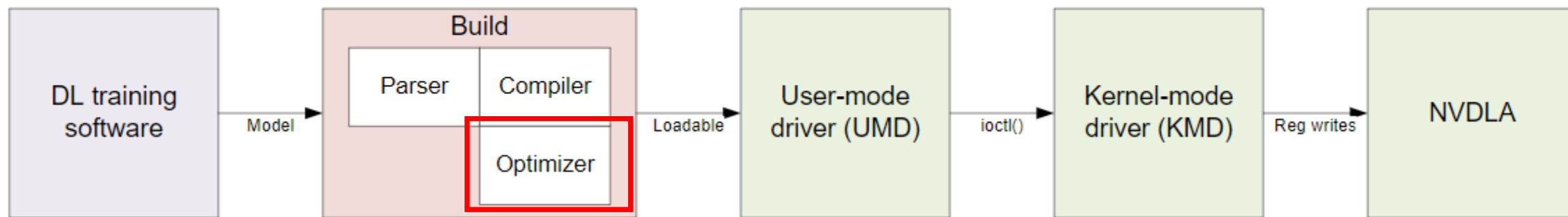


Fig. 3 - Dataflow diagram inside of NVDLA system software.

依赖项 (TODO):

1. 完成更大规模CNN的编译和在NVDLA上能跑通
2. NCNN+other optimization framework
3. VP无法模拟性能

# 概览

- NVDLA 是什么
- NVDLA 的软件栈
- 基本版NVDLA compiler: 3件事
- 高阶NVDLA compiler: 再加1件事
- 总结

# 总结

- “一个Target并不算完全开源的开源DLA编译器”
  - ISA spec: no
  - UMD API spec/KMD API spec : in code
- 3件事情
- 8个列表 (3个很重要)
- 对具有类似架构 (配置寄存器而不是指令集, 双缓冲) 的DLA的软件栈和编译器开发具有一定借鉴意义