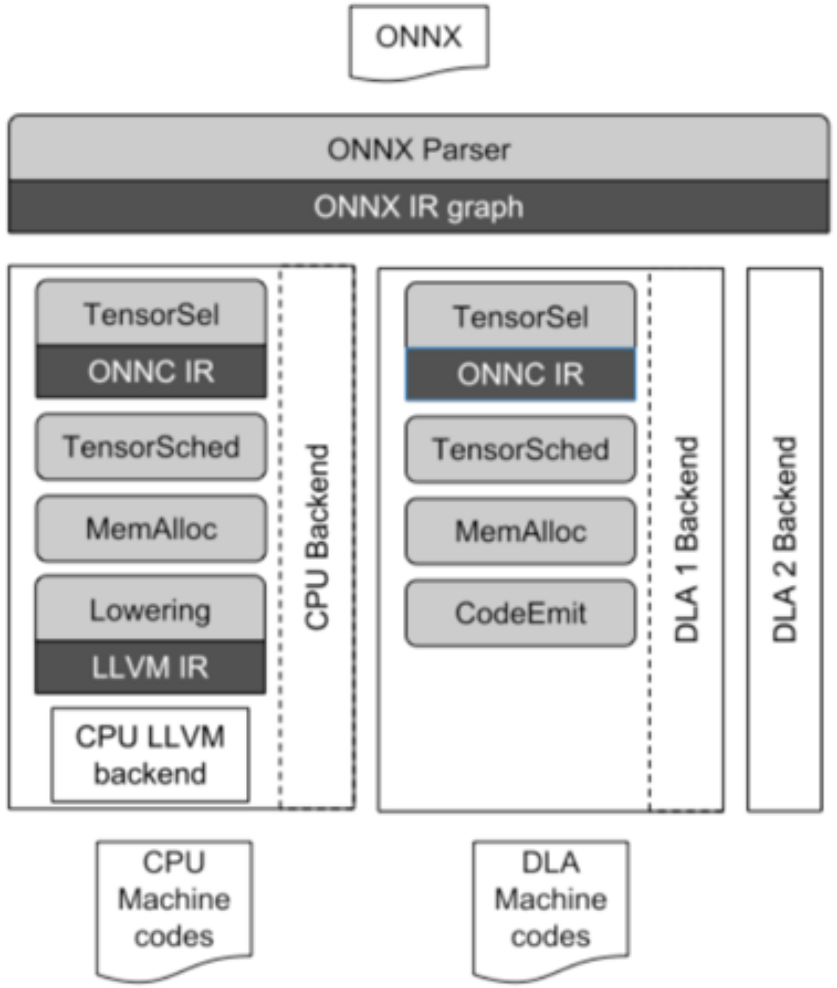
Open Neural Network Compiler Tutorials

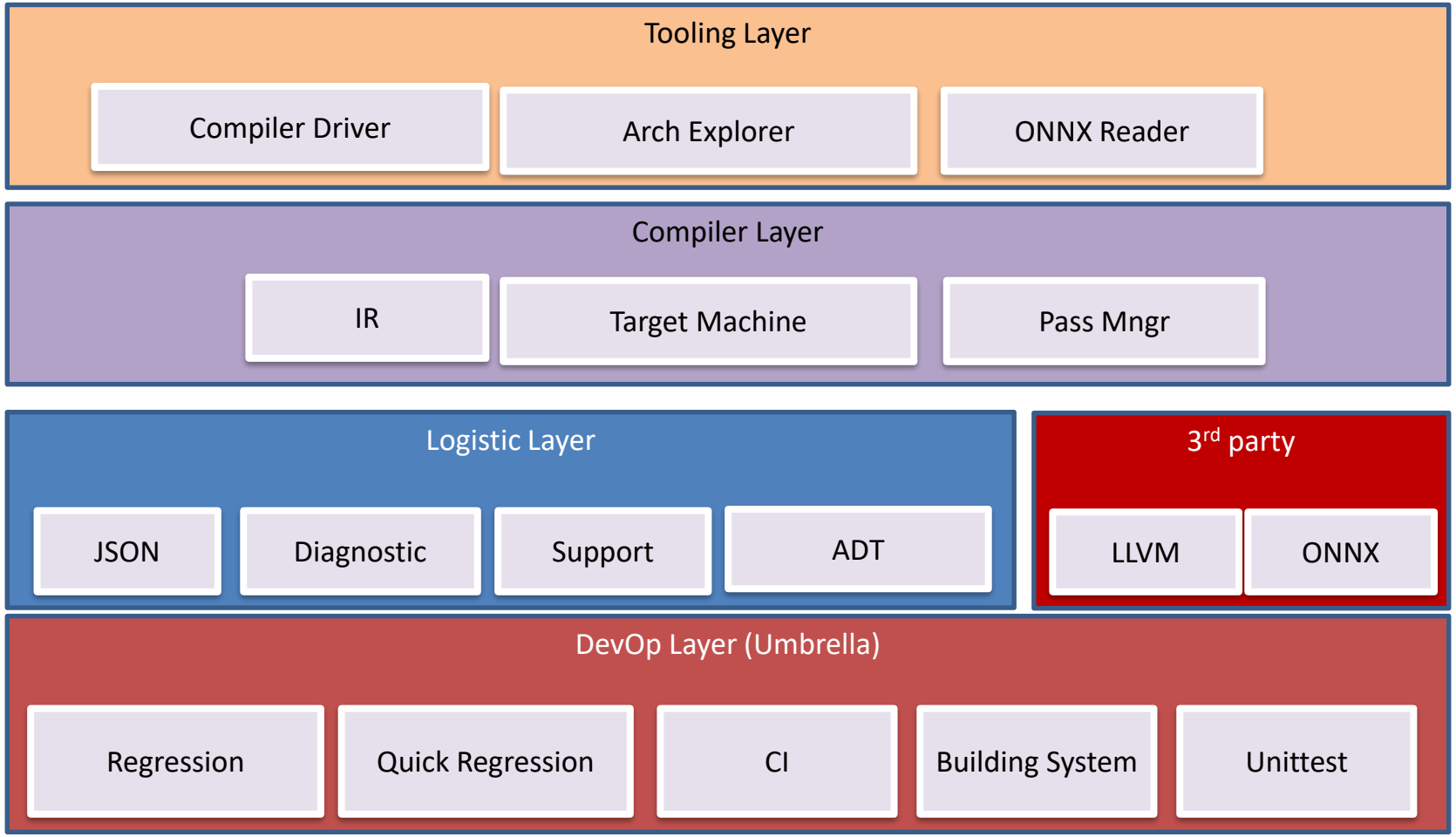
人工智能系统与集成电路研讨会 | 2019年8月18-19日

Outline

- ONNC Overview
- ONNC Logistic Layers
- ONNC Intermediate Representation
- ONNC Pass Management
- ONNC Target Backend



High Level Concept of the Architecture Structure



ONNC
Logistic Layers

ONNC Logistic Layers

- General Routines
- ADT (Abstract Data Type)
  - StringRef
  - Rope
  - IList
  - Flag
  - BinaryTree
- JSON (JSON format read/write/access)
- Support (System, Memory and I/O)
  - IOStream
  - MemoryPool
  - ManagedStatic
  - DataTypes
- Diagnostic (error message handling)

Dev-defined error vs System-defined error

- System-defined error
  - defined by operating system
  - For example, EBUSY, ENOSYS
- An error defined by developer
  - Most defined by compilers
  - For example, Success, NotStartedYet, UnknownError
- All errors that happens in Linux, Mac OS X, FreeBSD and Windows are listed in <Support/ErrorCode.h>

class SystemError

- To encapsulate all errors, we provide SystemError class
- sizeof(SystemError) == sizeof(int)
- You can use it with std::ostream and campare operators

```
#include <onnc/Support/ErrorCode.h>
#include <onnc/Support/IOStream.h>
#include <errno.h>

using namespace onnc;

int fd = open("/", O_WRONLY);
SystemError err(errno); // get a copy from errno
if (!err.isGood())
  errs() << err; // permission deny
```

SystemError is the standard error handler in ONNC

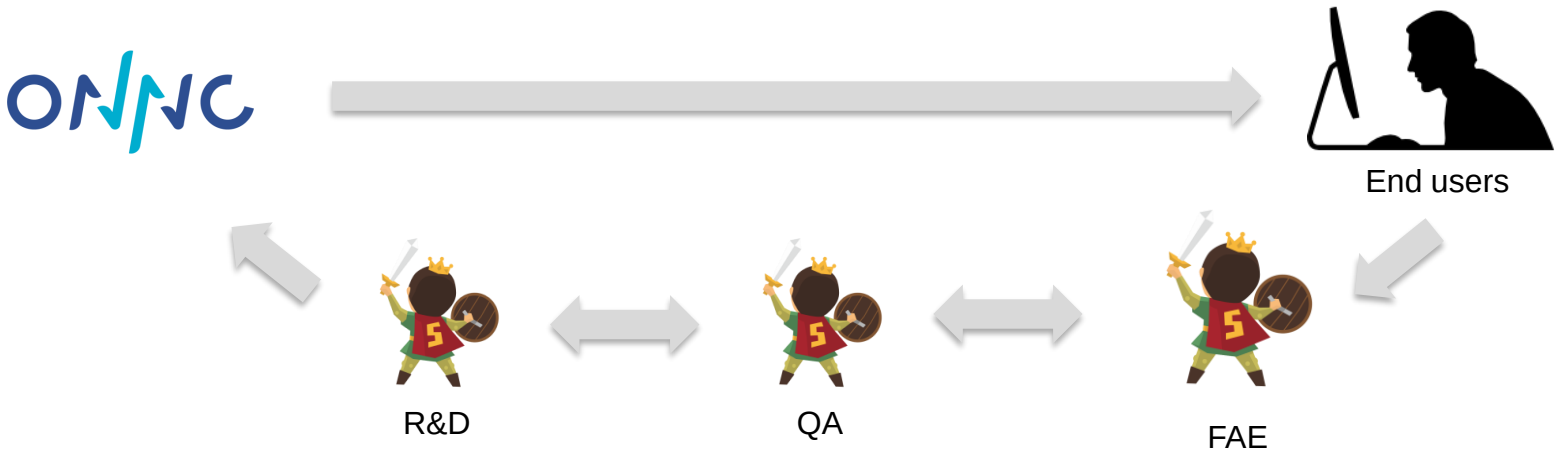
- Most interfaces in Supports use SystemError
  - FileHandle::open

Diagnostic - ONNC specific exception handler

- Most compilers don’t introduce exception and RTTI (runtime type information) in their building system
  - exception slows down the performance of compiler
  - RTTI introduces uncontrollable exceptions
- Most compilers provide her own exception handling system.
- ONNC provides Diagnostic system as its own exception handling system

When to use diagnostics

- If the end users should read the message, then we use diagnostics
- FAE follows the lead of the messages returned by ONNC
- FAE can not follow segmentation fault. We should do our utmost to avoid segmentation fault.
- You can keep using naïve printf only if you are
  - making a debugging message by your own, and
  - you won‘t submit it into master



Three modes: Normal/Verbose/Engineering/Noisy

- Most tools in ONNC have three modes: normal, engineering and noisy
- Users can turn on verbose mode by giving one `-v` (verbose)
- Engineering mode by giving more than three `-v`
- Noisy mode by giving more than five `-v`

```
$ onnc -h # normal mode
$ onnc -h -v # verbose mode
$ onnc -h -v -v -v # engineering mode
$ onnc -h -v -v -v -v -v # noisy mode
```

Error Levels

- include <onnc/Diagnostic/MsgHandling.h>

Mode	*Handler*	Meaning
normal (silent)	*unreachable*	Impossible error
	fatal	Serious error. ONNC stop immediately
	error	Normal error. ONNC will try to run further paces.
verbose	*warning*	warning
engineering	*debug*	You can see the debugging message
	note	You can see the notes for debugging
noisy	*ignore*	Show you every thing

Examples

```
#include <skymizer/Diagnostic/MsgHandling.h>
using namespace skymizer;

if (I got a serious problem)
  fatal(fatal_open_folder) << “this folder” << 2;
```

- Format
  handler(ID) << mesg0 << mesg1 << .. << mesg9;
- There are most `10`messages in one print
  - If you put more then 10 messages, you shall get a *compilation error* when you’re building ONNC.
- Messages can be one of the types:

bool/int32_t/int64_t	*Support/DataTypes*
char*/std::string/onnc::StringRef	*ADT/StringRef.h*
Path	*Support/Path.h*
onnc::SystemError	*Support/ErrorCode.h*

Add a new error handler

- Add a handler in include/onnc/Diagnostics/DiagCommonKinds.inc
  - DIAG(tag, error level, message)
- Message format

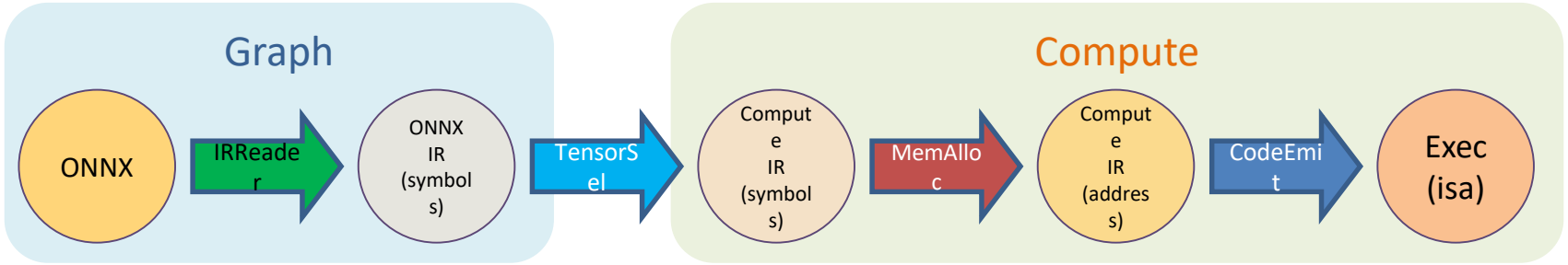
```
DIAG(fatal_open_folder, Fatal,   "cannot open the folder `%0`. (Code: %1)")
```

```
#include <skymizer/Diagnostic/MsgHandling.h>
using namespace skymizer;

fatal(fatal_open_folder) << “this folder” << 2;
```

ONNC
Intermediate Representation (IR)

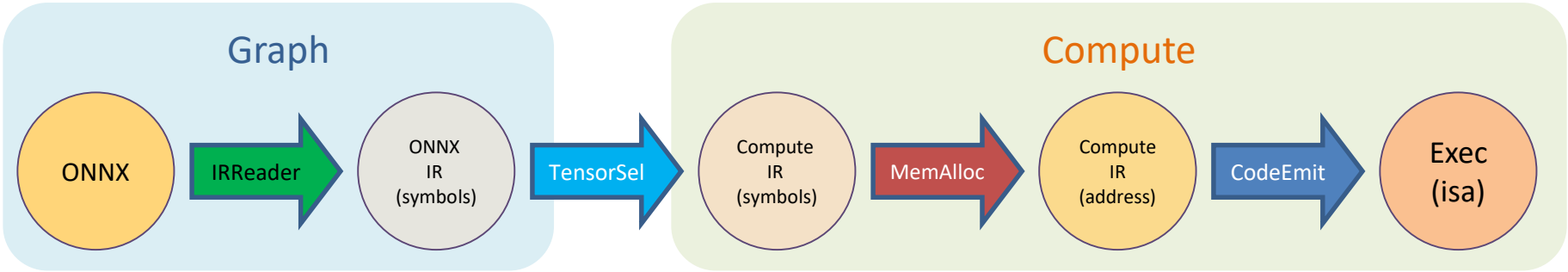
ONNC IR: The heart of ONNC

- Core design thought - from network domain to compute unit
- Four phases in the compilation process
  - IRReader - read ONNX prototex and build ONNX IR
  - TensorSel - select corresponding instruction for target devices
  - MemAlloc - turn symbolic operands into memory address
    - instruction scheduling
    - memory partition
    - memory allocation
- CodeEmit - emit binary code for target devices



Two Levels of IR: Graph IR and Compute IR

- Graph IR
  - original ONNX IR
  - used to represent ONNX models
- Compute IR
  - ONNC IR
  - used to add hardware information on
- ONNX allows multiple subgraphs in one model
- ONNC allows multiple subgraphs in one model, too



Module - The façade of all IRs

- In ONNC, a module represents a single unit of network that is to be processed together
- A module contains all instances of graph IR and compute IR.
- Pass developers and target backend developers handle with module all the time

```
// include/onnc/IR/Module.h

class Module
{
public:
  xGraph* getRootTensorGraph();
  ComputeGraph* getRootComputeGraph();

  tg_iterator tgBegin(); //< used to traverse all tensor graphs
  tg_iterator tgEnd();
  cg_iterator cgBegin(); //< used to traverse all compute graphs
  cg_iterator cgEnd();
};
```

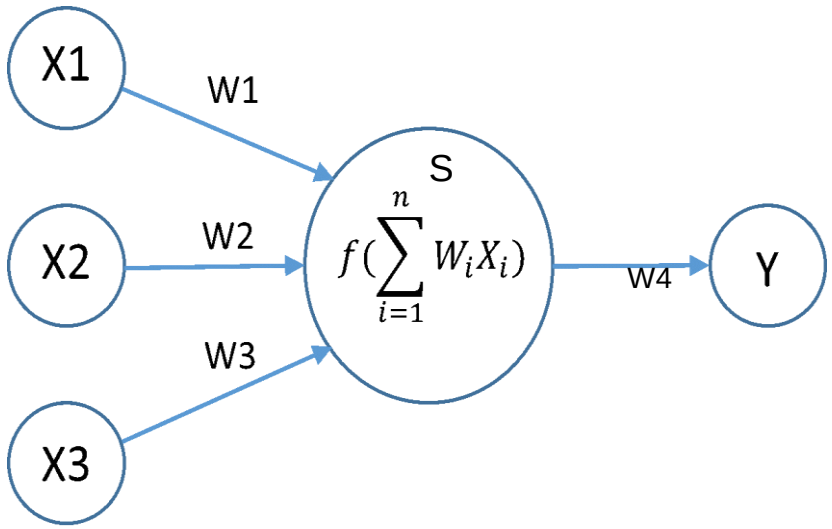
Read data from ONNX file

- onnc::onnx::Reader can read ONNX file and put data in a module

```
#include <onnc/IRReader/ONNXReader.h>
#include <onnc/IR/Module.h>
#include <onnc/Support/Path.h>
#include <onnc/Support/IOStream.h>
using namespace onnc;

void main()
{
  Module module;
  onnc::onnx::Reader reader;
  reader.parse("my path/bvlc_alexnet.model", module);
  module.print(outs()); // print module information
}
```

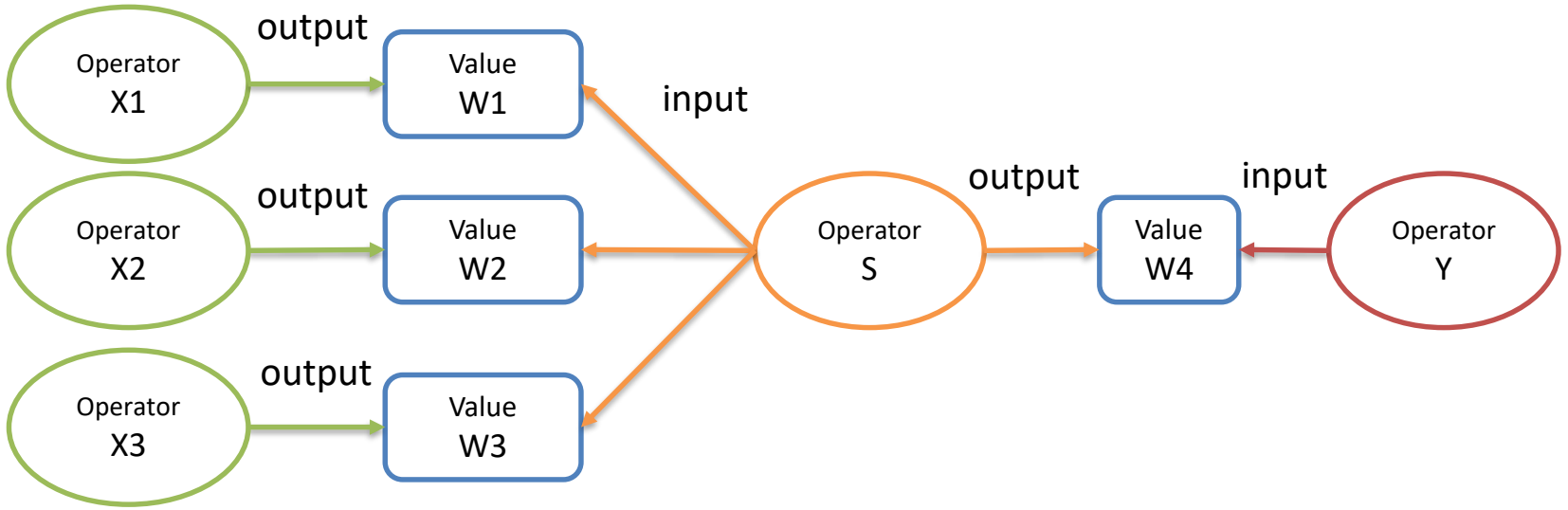
The core of compiler - Use-Define Chain

- X1 layer defines a new value of tensor w1
- Conv layer uses three tensors w1, w2, w3
- The relationships between use and define forms a chain - Use-Define Chain



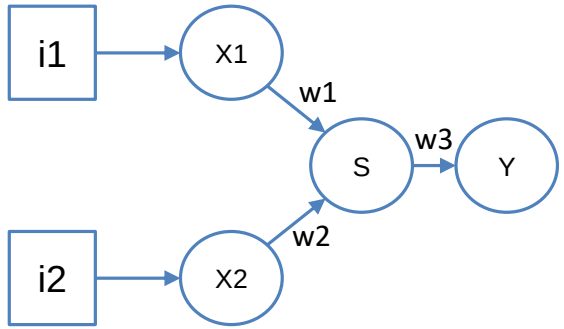
Conceptual data structure of a Use-Define Chain

- Operators are definers. They have a pointer to their output value
- Operators are also users. They have pointers to their input values
- Input value and output value is the same object



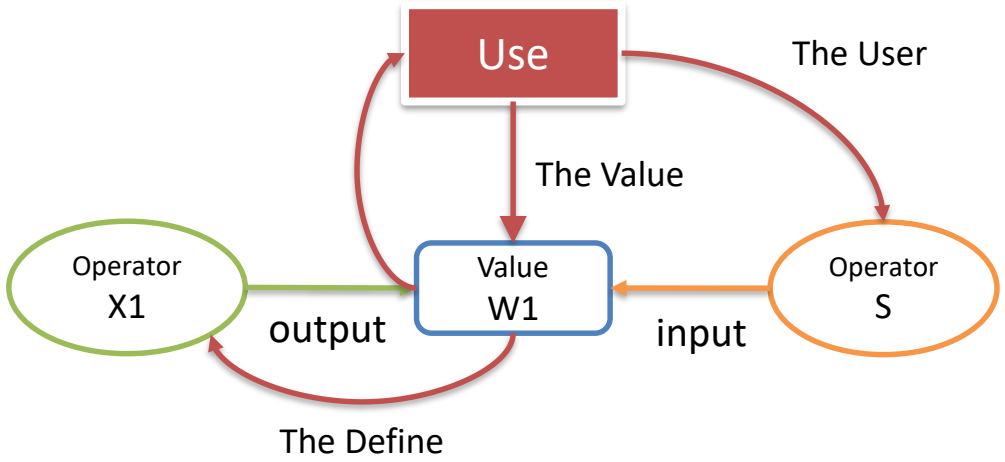
Graph IR - defined by ONNX

- Optimizing passes shall change semantics in compute IR, not graph IR.
- Developers read graph IR to know the original semantics of the neural network.
- Because ONNX IR is still changing, ONNC has to re-define all ONNX data structure in onnc namespace with `x` prefix.
  - onnx::Node -> onnc::xNode
- You can see the definitions in onnc/Config/ONNX.h.in

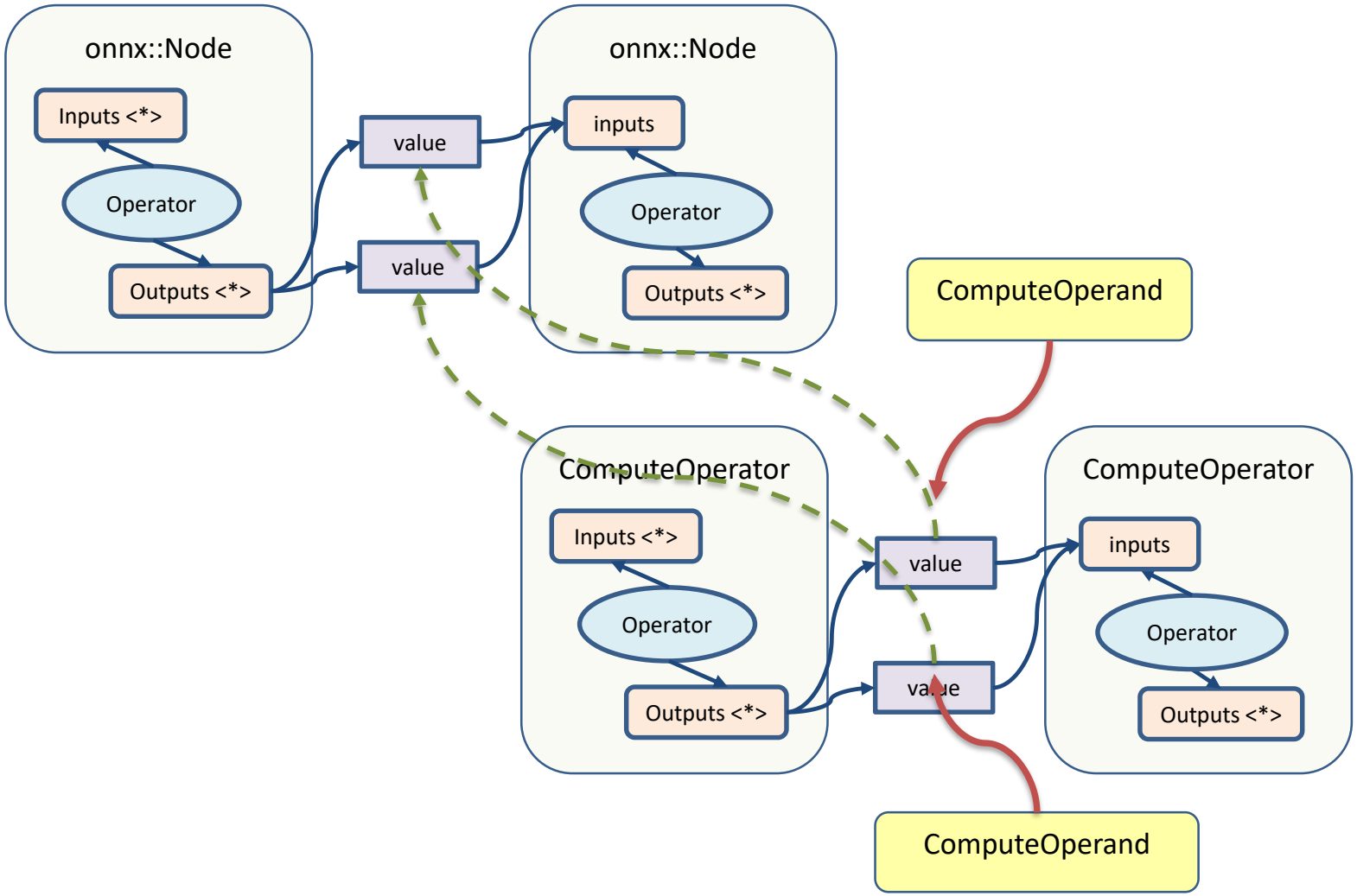


Data structure of a Use-Define Chain in ONNC

- To chain users and definers, ONNC provides a special data structure called `Use` to point out users and its value.
- Since in Neural Network, every value has only one definer, we bookkeep definers in value.



ONNX IR with ONNC IR



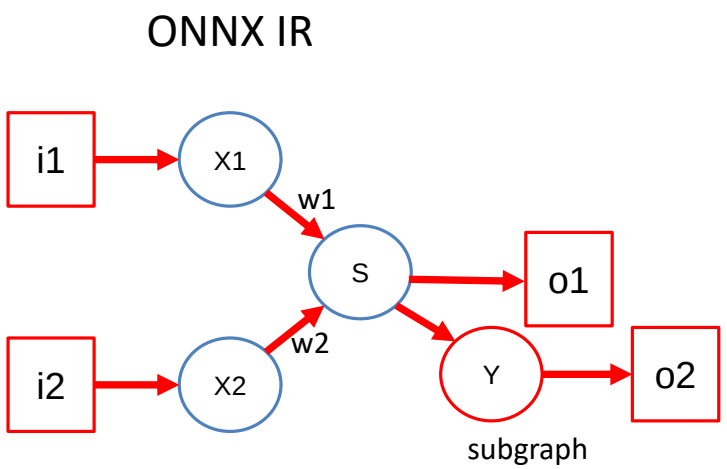
IRBuilder

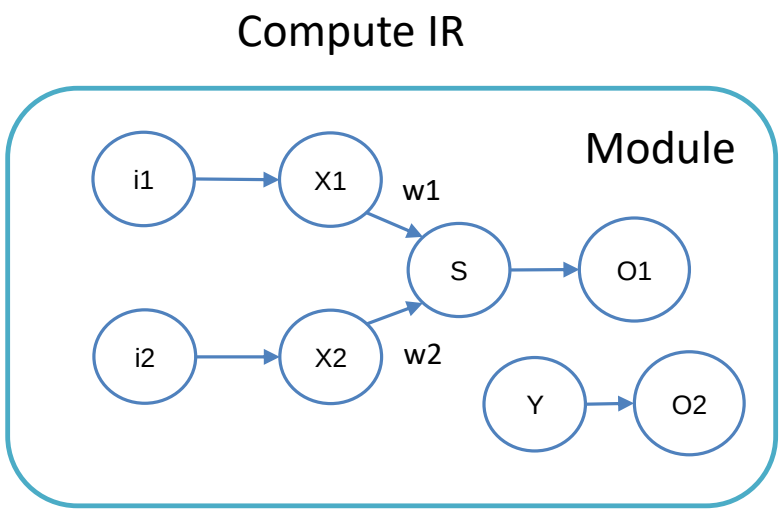
IRBuilder encapsulates creating process of a module.

- Handle with ONNX IR
  - create a new ONNX graph
  - create/clnoe ONNX layers
  - create ONNX input tensors
  - create ONNX output tensors
- Handle with compute IR
  - create a new compute graph
  - create/clone layers
  - create input layers
  - create output layers
- See src/tools/unittests/ComputeIRTest.cpp

Some notes for using ONNX Graph IR

1. Compute IR is smaller
   - ONNX IR keeps data in a tensor even we don't need it. That is, ONNX IR may be fat.
   - ONNC compute IR will use symbolic representation instead of keeping data.
2. It's much easier to find input/output tensors in compute IR
   - ONNX IR doesn't provide initializer operator for the initial inputs, developers must find initial inputs by names.
   - ONNC IR provides initializer/output operator and it reduces a lot works in optimization algorithm
3. It's much easier to find subgraph in compute IR
   - ONNX IR uses node attributes to keep subgraphs. If you want to list all subgraphs, you must traverse the whole nodes and edges.
   - ONNC module contains multiple subgraphs of compute IR and ONNX IR.



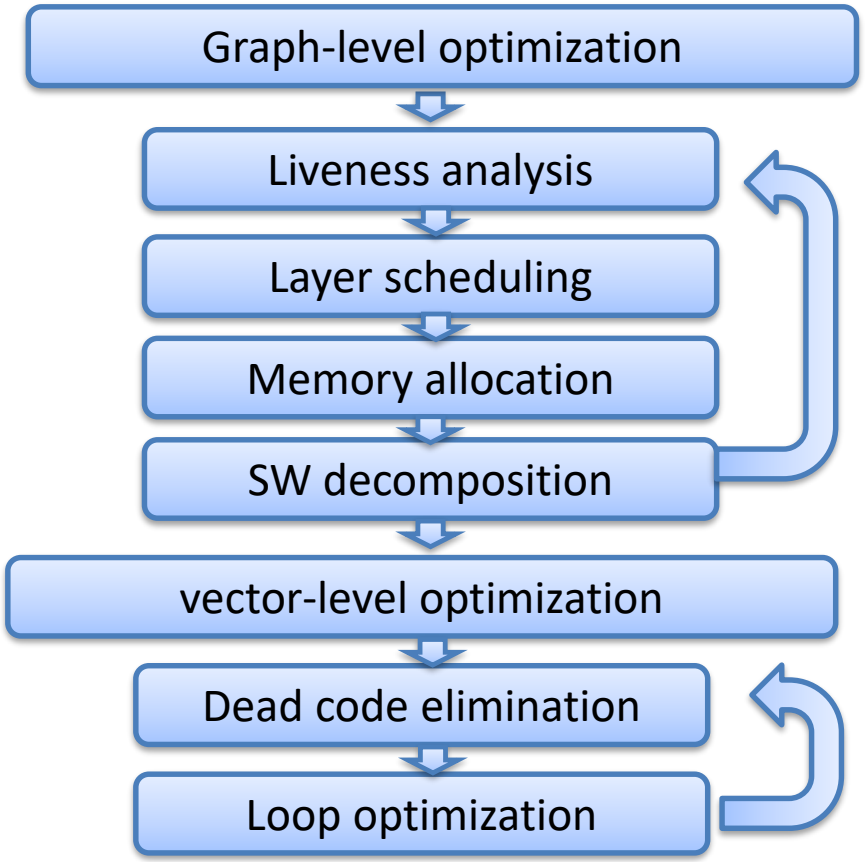


ONNC
Pass Management

ONNC Optimization Flows

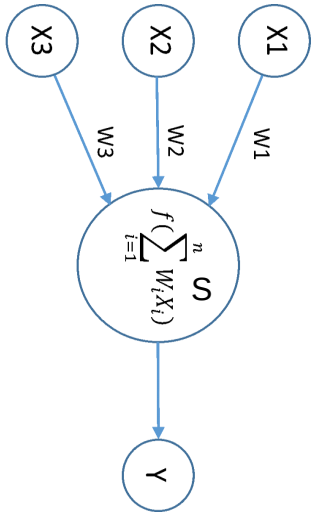
- There are two kinds of optimization algorithms for neural network
  - Graph-level optimization
  - Vector-level optimization
- Graph-level optimization handles with matrices
  - Separate a matrix into pieces
  - Merge several matrices into big one
  - Set the order of matrix multiplications
- Vector-level optimization handles with vectors
  - Reorder the cross products of vectors

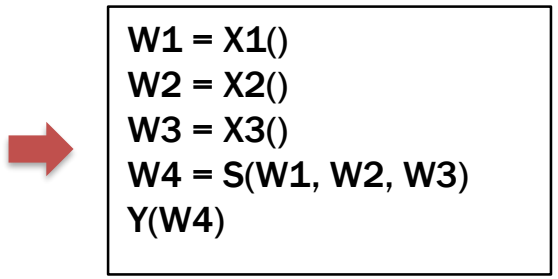
ONNC Optimization Flows



Liveness analysis of tensors

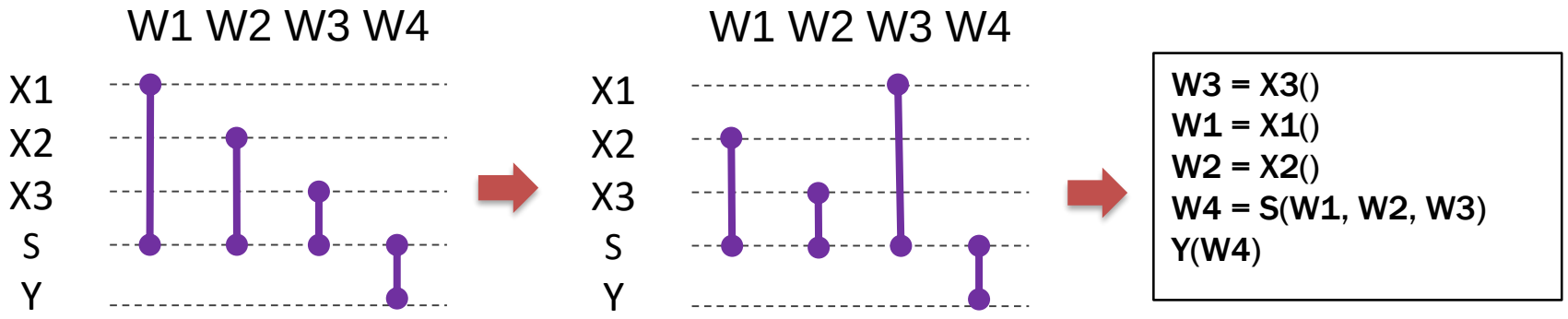
- Find out the live range of every tensor
- Leverage use-define chain of ONNX
- By the help of simple liveness analysis, we can reuse local memory and eliminate ½ memory consumption with greedy allocation





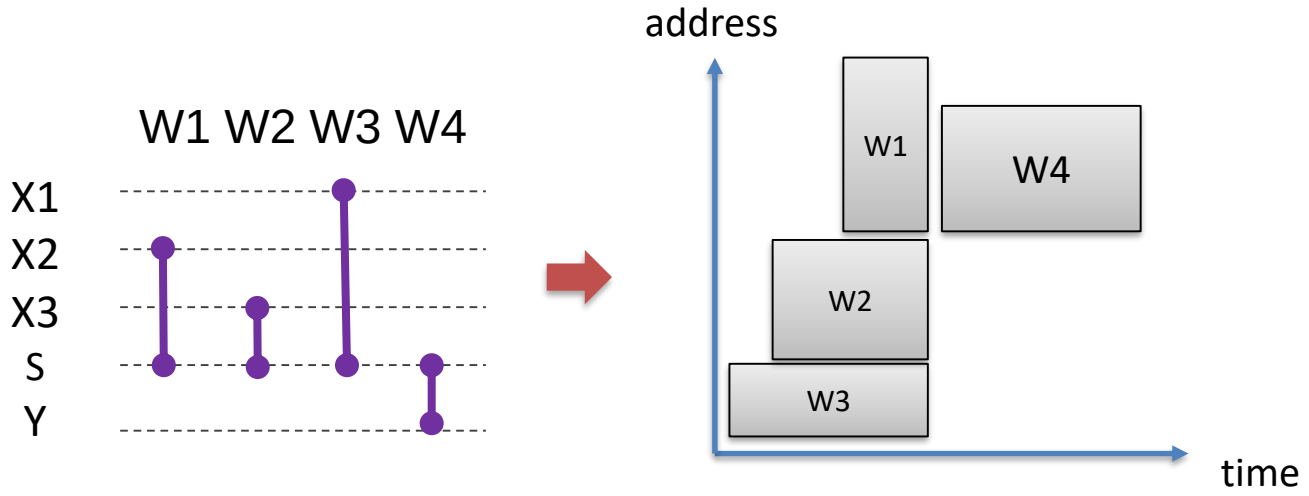
Layer Scheduling

- If size W2 > W1 > W3, then we can reorder X1 X2 X3 to reduce the memory consumption

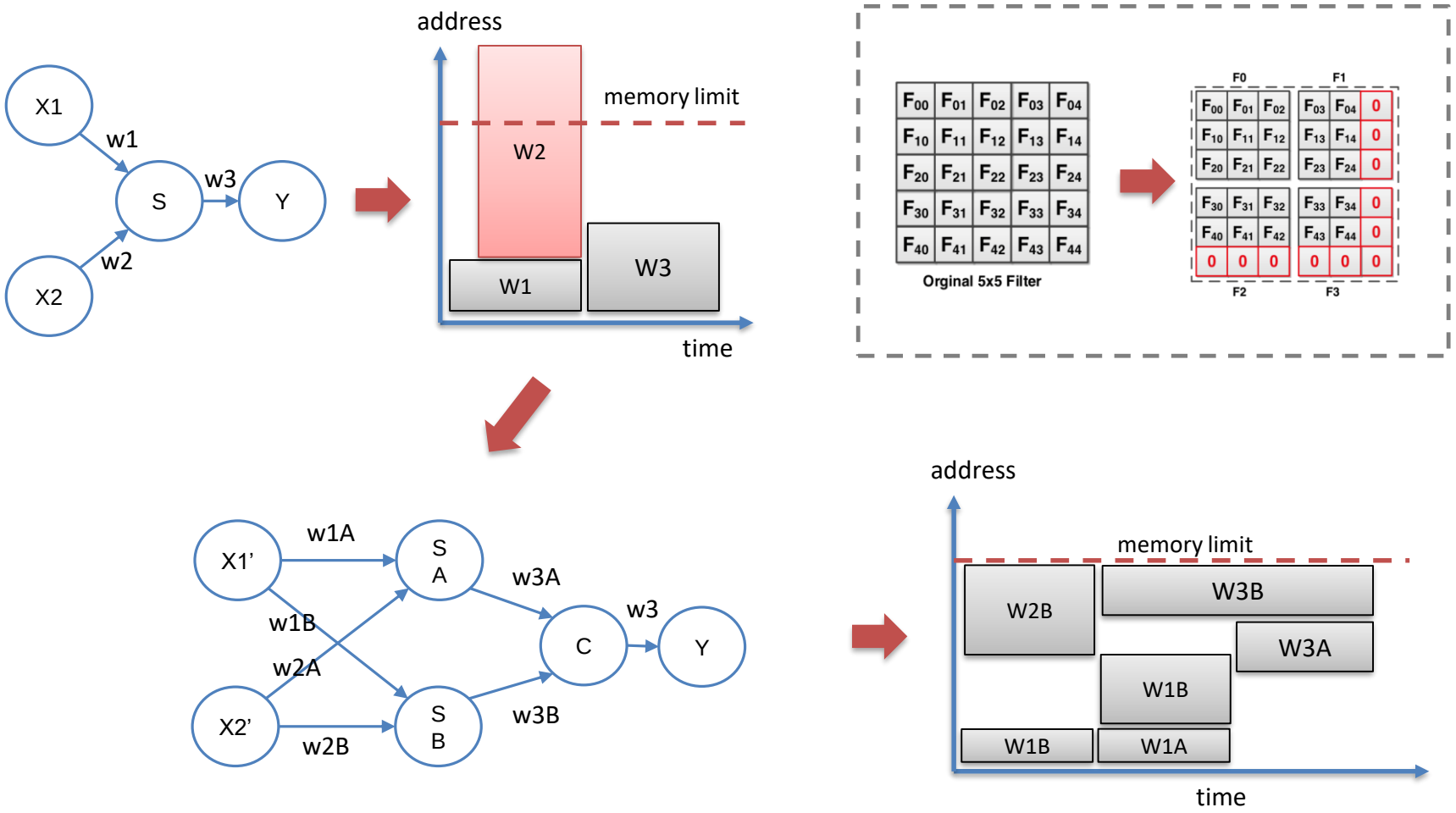


Memory Allocation

- Memory allocation is to use to allocate memory for each layer
- Layer Scheduling affects the results of memory allocation

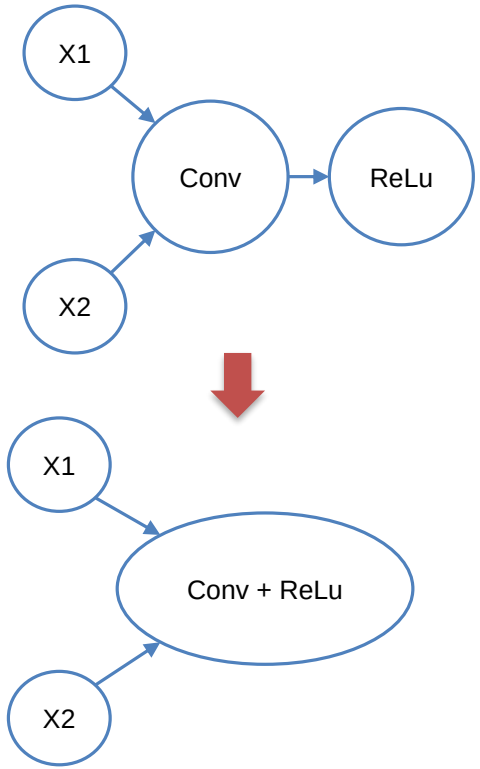


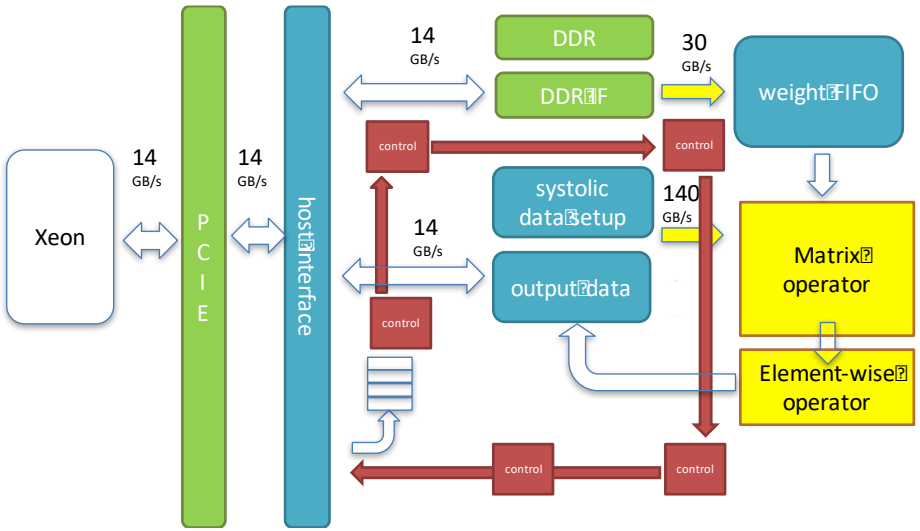
Layer Splitting - Handle the memory limit



Layer Fusion

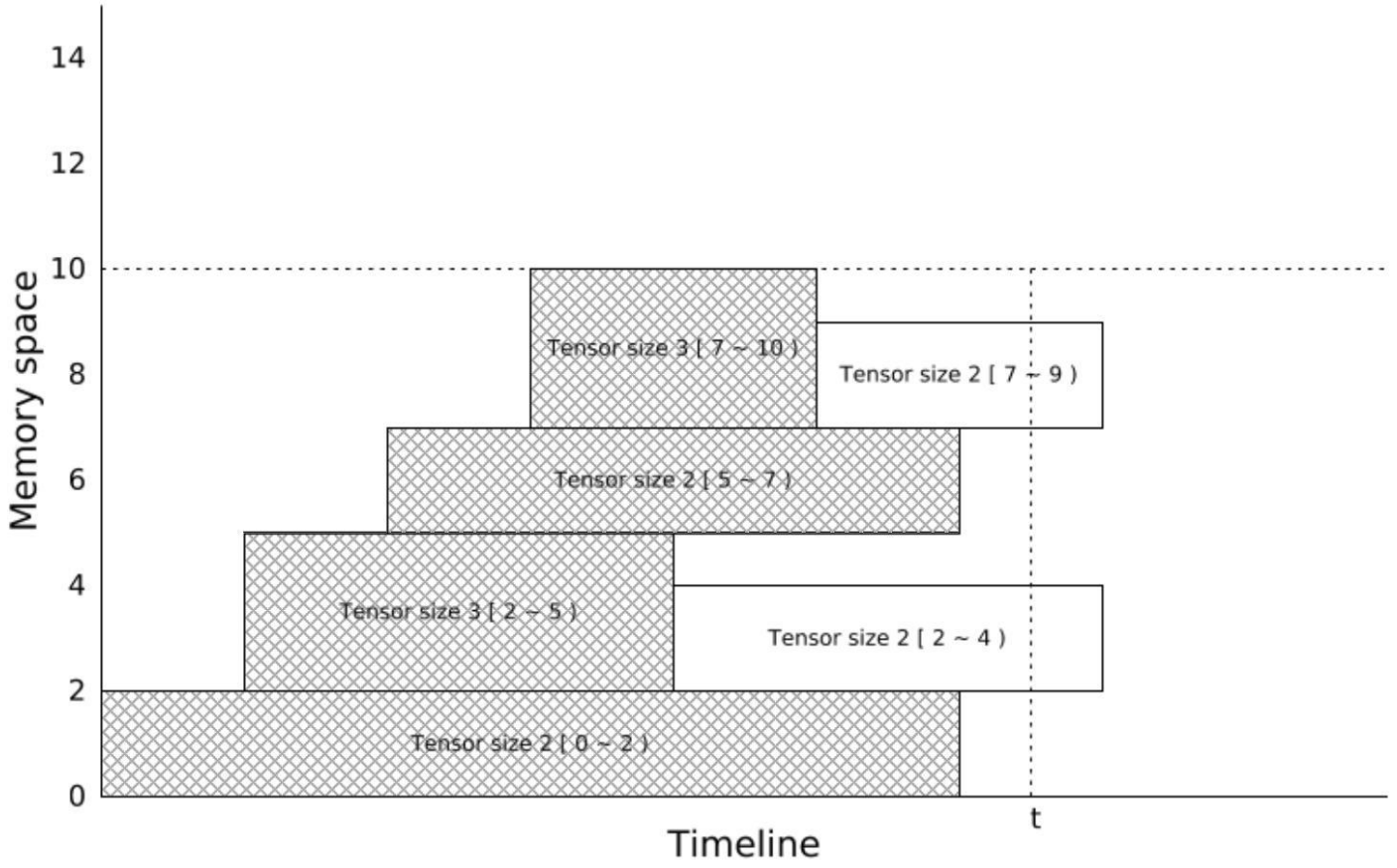
- Weight stationary and output stationary architectures usually have dedicated element-wise function unit.
- If we can leverage the element-wise function unit, then we can save data movement from outside to the inside core



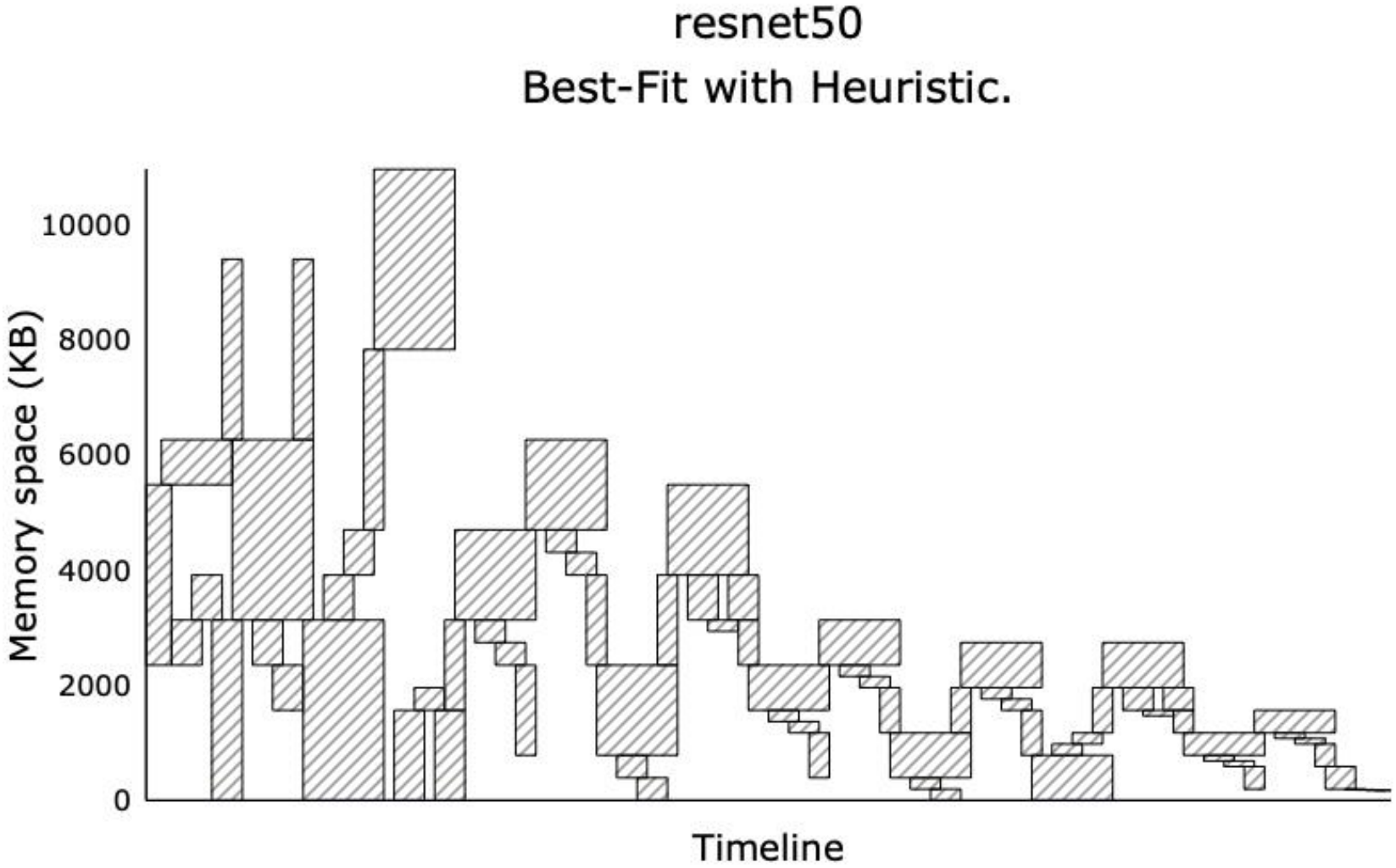


Observation: Fragmentation at The Boundaries

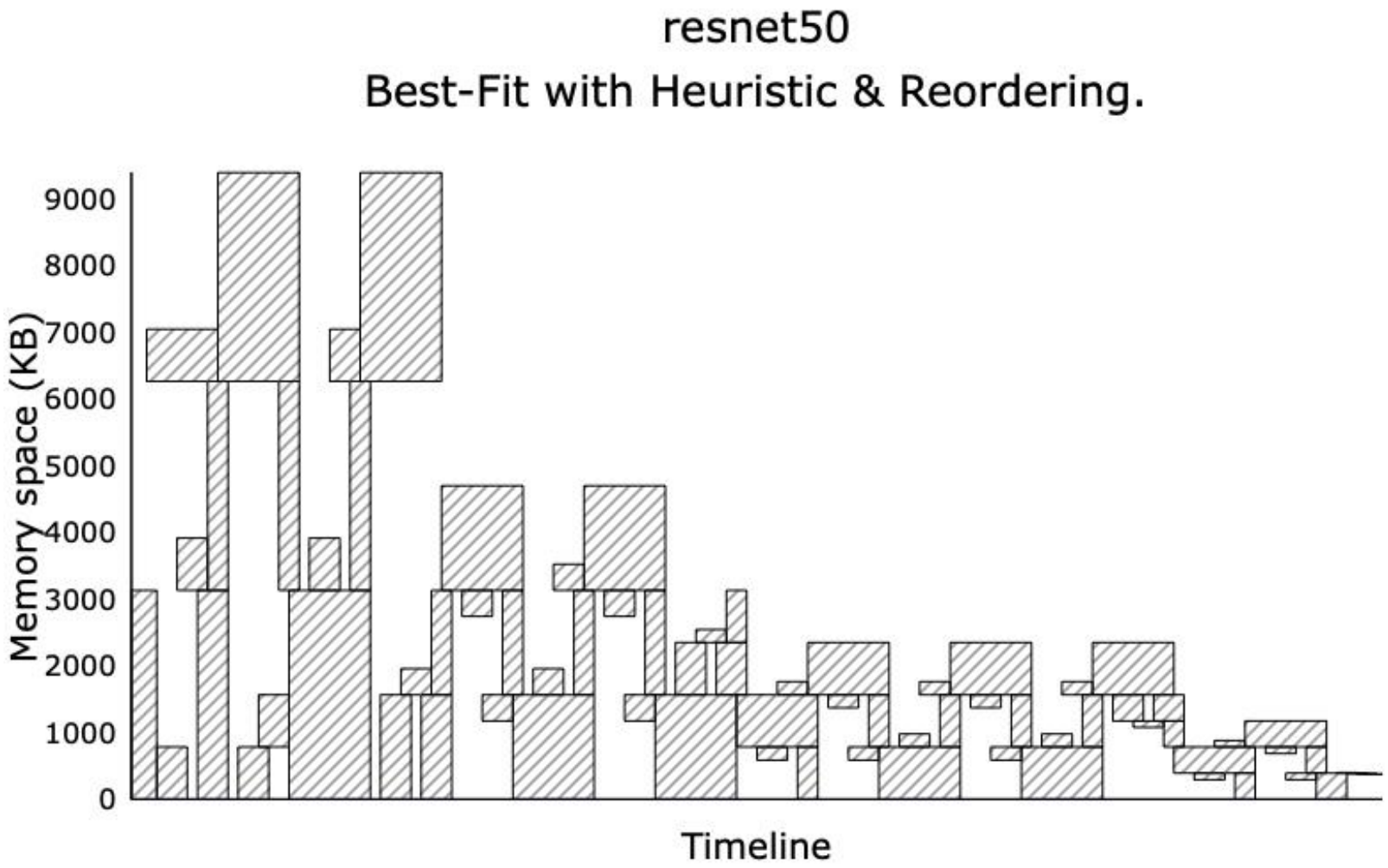
At time t, both the top and bottom memory space have free space for allocation



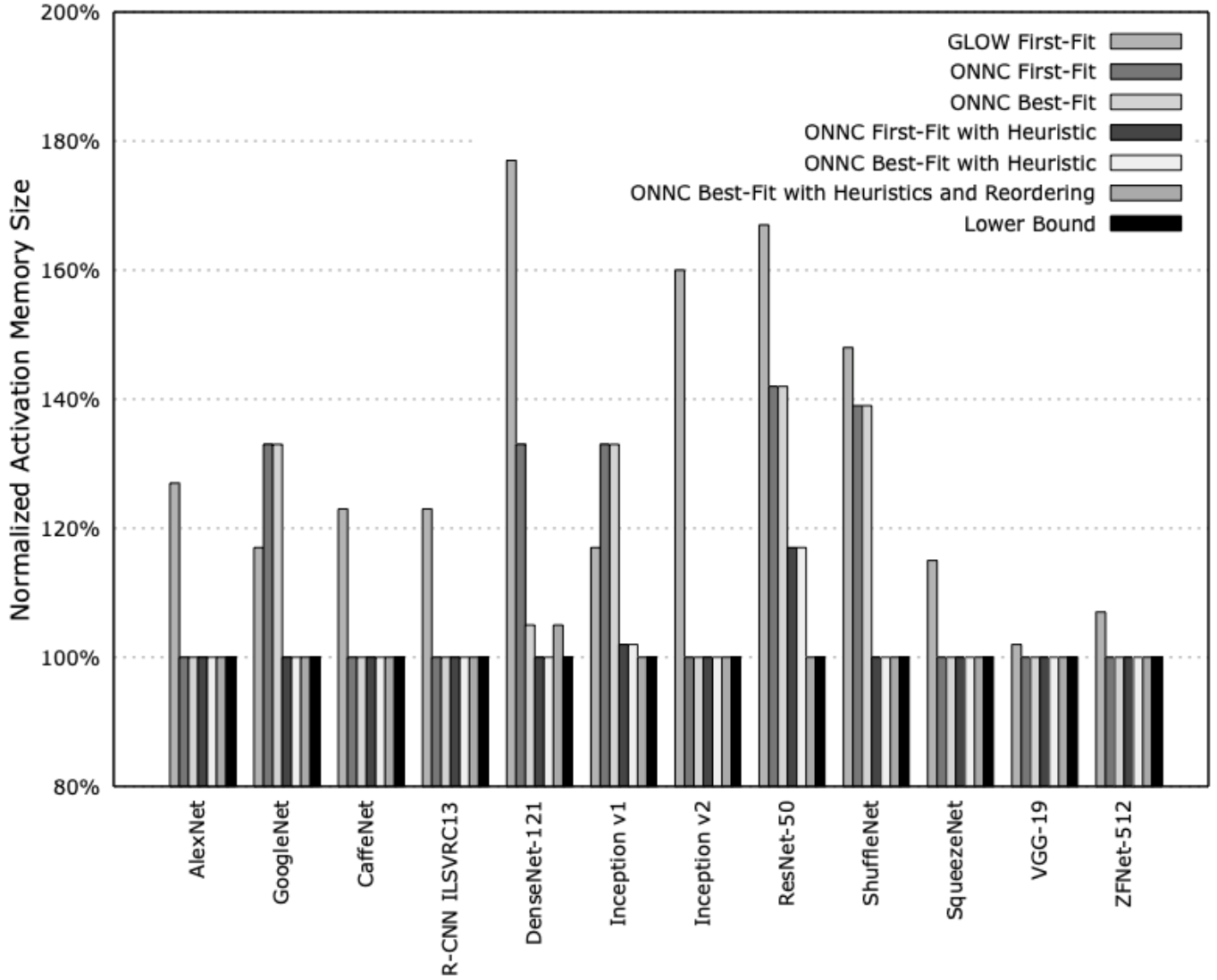
Best-Fit With Heuristics



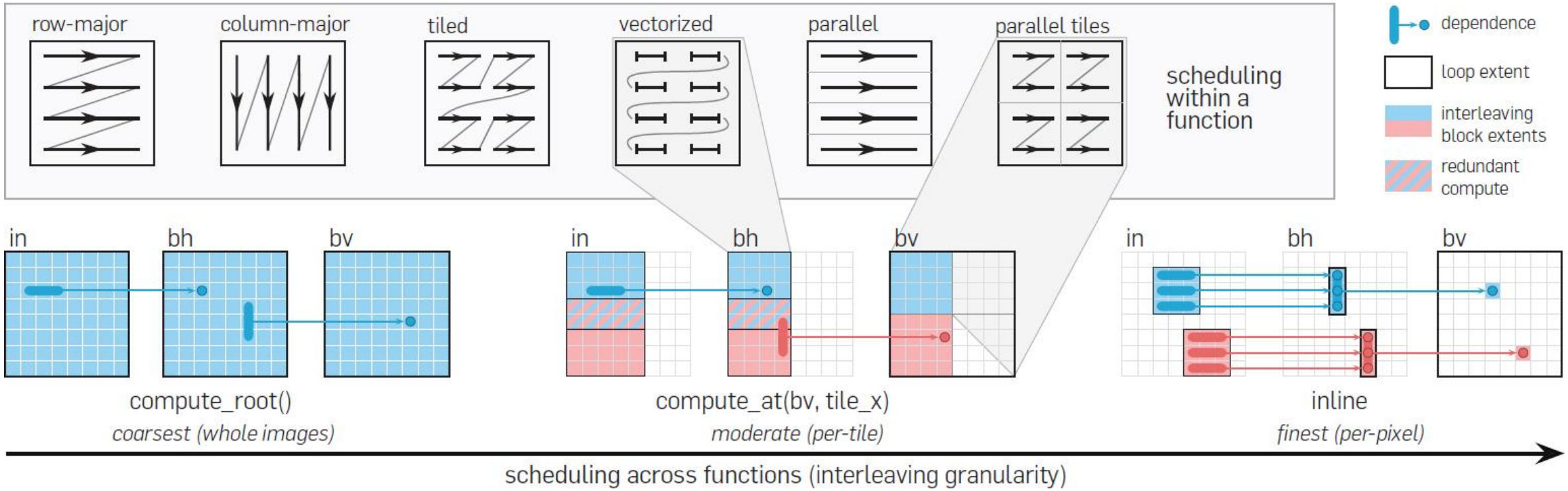
Best-Fit With Heuristics & Layer Reordering



Near-optimal results:
ONNC Best-Fit with Heuristic and Reordering

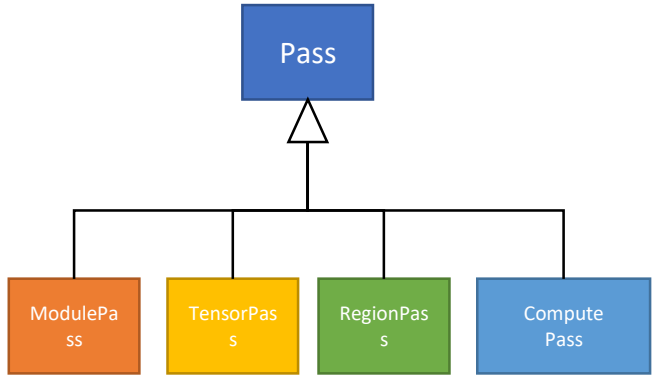


Vector-level Optimization



Four Kinds of Passes in ONNC

- ModulePass
  - The most general of all superclasses that you can use
  - Use entire network as a unit
- TensorPass
  - Use Tensor Graph as a unit
  - Tensor Graph bases on ONNX IR
- RegionPass
  - Use each signle-entry-signe-exit region in a tensor graph as a unit
  - For example, groups in GoogLeNet
- ComputePass
  - Use Compute Graph as a unit




```
// methods in class Pass
bool run(Module& pModule);
virtual bool doInitialization(Module& pModule);
virtual bool doFinalization(Module& pModule);
```

```
// methods in class ModulePass
virtual bool runOnModule(Module& pModule) = 0;
```

```
// methods in class TensorPass
virtual bool runOnTensor(TensorGraph& pGraph) = 0;
```

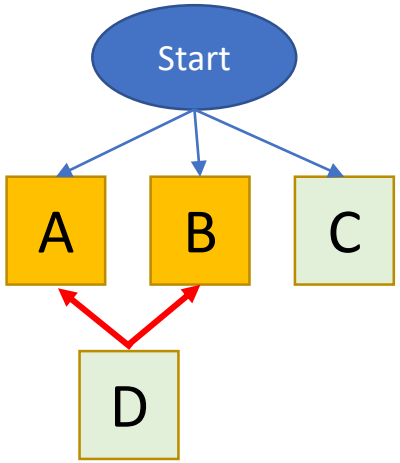
AnalysisUsage describes the dependencies between Passes

- PassManager automatically creates all Passes that used by the other Passes.
- Similar to LLVM. Engineers who already familiar to LLVM can understand ONNC in a short time

```
/// in A.cpp
INITIALIZE_PASS(A, “pass_a”)
```

```
/// in B.cpp
INITIALIZE_PASS(B, “pass_b”)
```

```
/// methods in Pass D. Override
void D::getAnalysisUsage(AnalysisUsage& pUsage) const
{
  pUsage.addRequiredID(A::ID);
  PUsage.addRequiredID(B::ID);
}
```



Write a Simple Dead Code Elimination Pass

- *Dropout layers* are used in training, not inference
- ONNC can safely remove *Dropout* layers
- In this tutorial, we will develop a simple dead code elimination algorithm - remove all dropout in the graph
- See lib/Transforms/RemoveTrainingNodes.cpp

ONNC
Target Backend

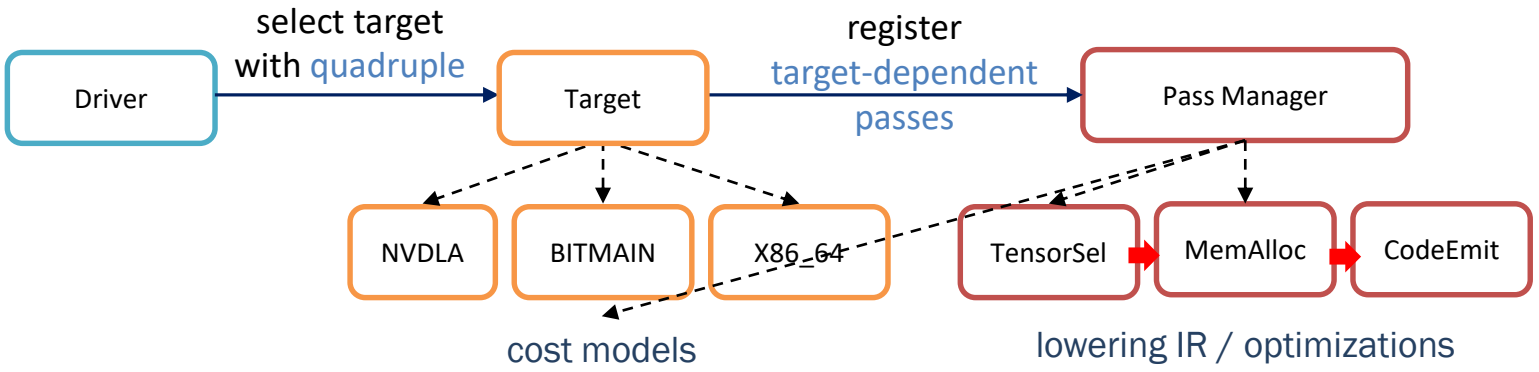
Quadruple - new way to select a NN compute unit

- Quadruple is a string representation that represents
  - Target hardware architecture (micro-architecture, ISA, etc.)
  - Target software environment (ABI, OS, etc.)
  - Target tool (compiler, loader, calibration, etc.)
- for example,
  - LLVM triple: arm-nono-linux-gnueabi
  - ONNC quadruple: arm-none-linux-gnueabi-calibration-0.2

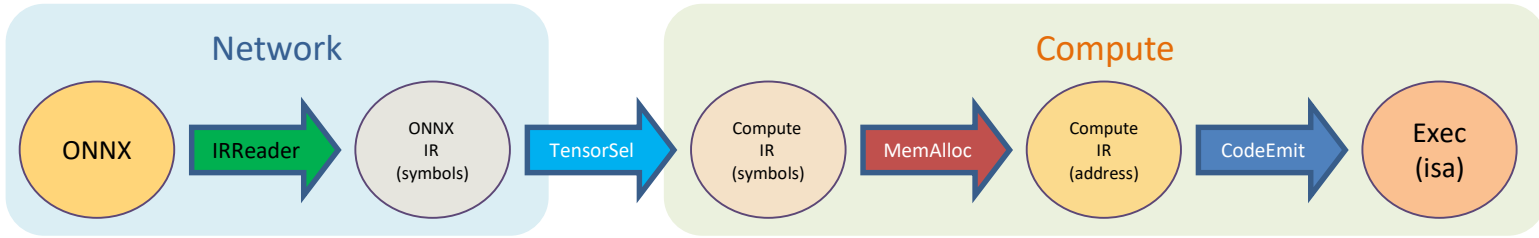
LLVM triple = HW x SW
ONNC quadruple = HW x SW x Tool

ONNC supports various target devices

- Use LLVM-like triple to select compiler target backend
  - compiler
  - loader
  - calibration
- Every Target instance represents a target device
  - contains cost model
  - contains target-dependent passes
- Target instance registers target-dependent passes into PassManager



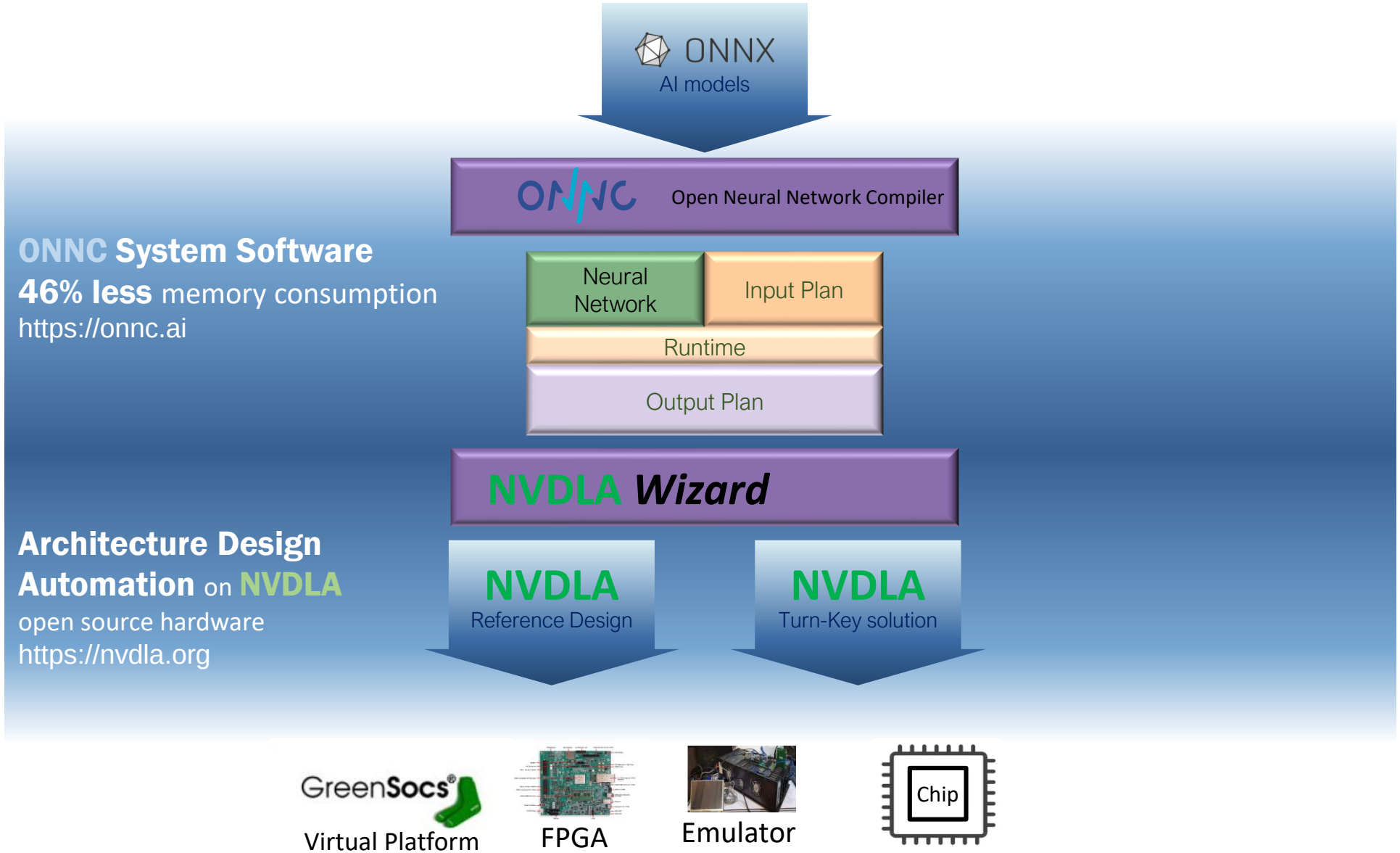
TargetBackend Controls The Phases of Lowering




```
// compiler bone

PassManager pm;

TargetBackend* backend =
    TargetRegistry::Lookup(“DLA-A”);

backend->addTensorSel(pm);
backend->addMemAlloc(pm);
backend->addCodeEmit(pm);

pm.run();
```

```
// Core/TargetBackend.h
class TargetBackend
{
  virtual void addTensorSel(PassManager& pPM) { return; }
  virtual void addMemAlloc (PassManager& pPM) { return; }
  virtual void addCodeEmit (PassManager& pPM) { return; }
};
```

```
// ATargetBackend.cpp

void ABackend::addCodeEmit(PassManager& pPM)
{
  pPM.add(createRemoveUnusedNodePass());
  pPM.add(createUpdateOutputInfoPass());
  pPM.add(createTGMemAllocInfoPass(this));
  pPM.add(createTargetLoweringPass(this));
  pPM.add(createTGCodeEmitPass(this));
}
```

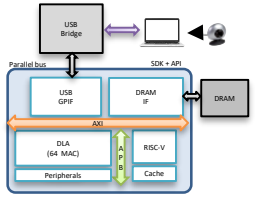

ONNX
AI models
ONNC
Open Neural Network Compiler
Neural Network
Input Plan
Runtime
Output Plan
NVDLA Wizard
NVDLA
Reference Design
NVDLA
Turn-Key solution
ONNC System Software
46% less memory consumption
https://onnc.ai
Architecture Design Automation on NVDLA
open source hardware
https://nvdla.org
GreenSocs
Virtual Platform
FPGA
Emulator
Chip

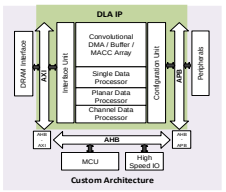
NVDLA
Reference Design

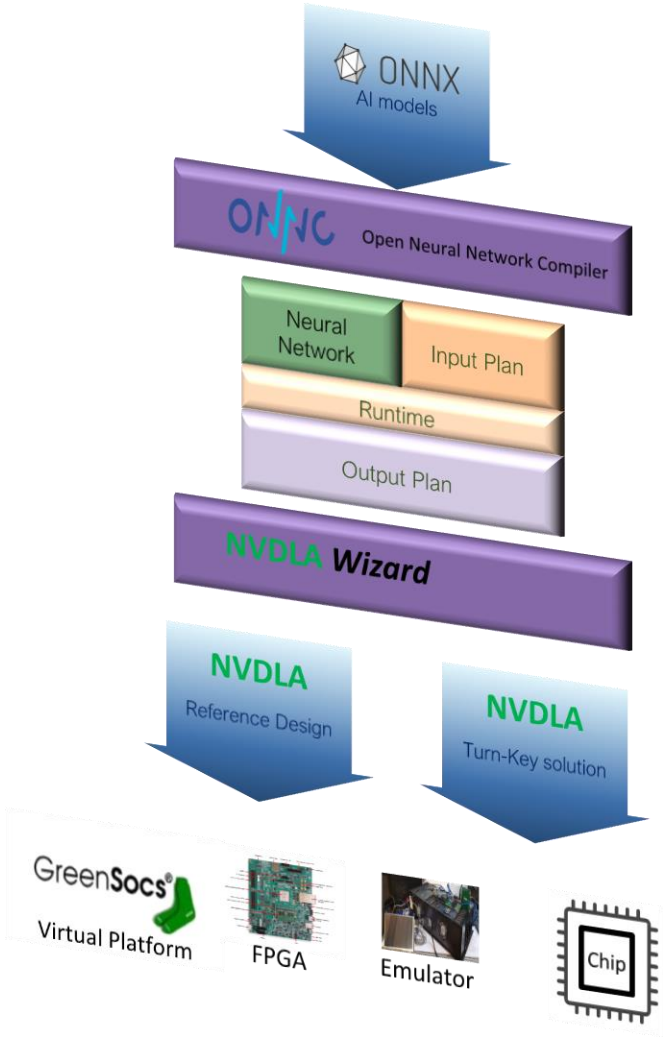
Original NVDLA opens hardware, but its software is not

ONNC is the first open source compiler supports NVDLA

- **QEMU-based virtual platform**
  - Generic and open source machine emulator and virtualizer
- **Virtual modeling for CPU and DLA**
  - Cortex-A processor
  - RISC-V processor
  - NVDLA with difference configuration
- **Performance Analysis Kit**
  - ONNC-based SDK
  - Running popular ML framework
  - Support debug and performance monitoring

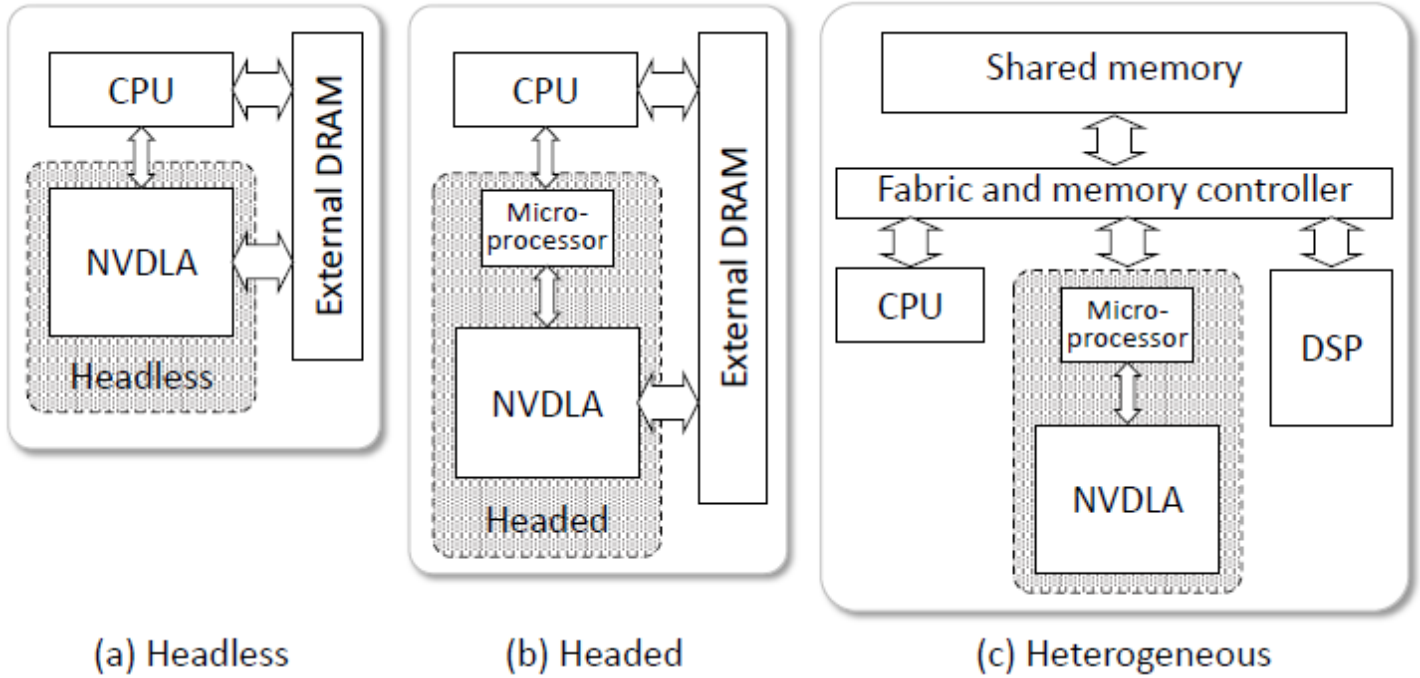






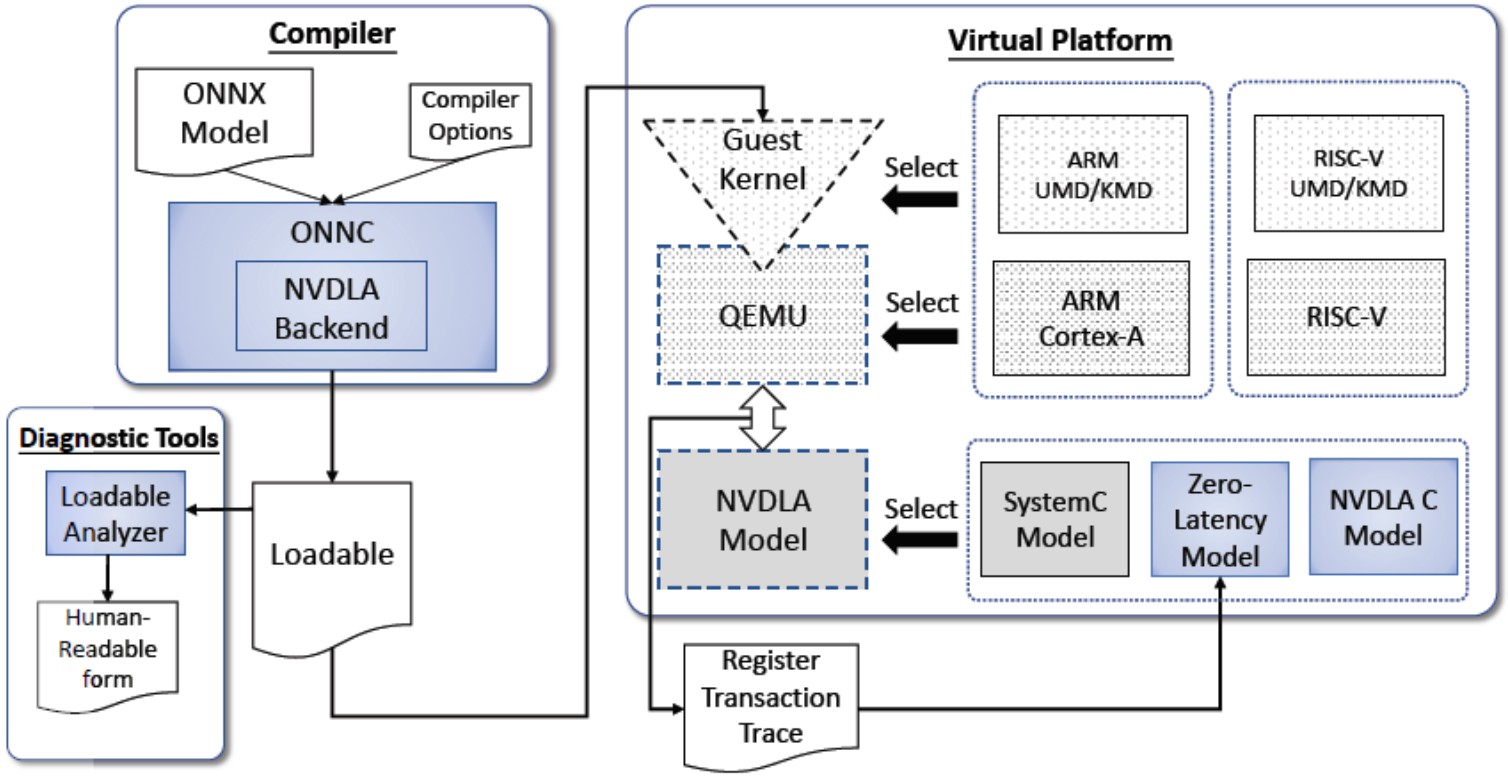
NVDLA System Architecture

- NVDLA needs an embedded CPU running Linux to leverage the NVDLA software stack.
- The headed implementation has an additional micro-controller aside the embedded CPU to offload the NVDLA control task



NVDLA Software Development Platform

- ONNC-based software development platform is built to support various hardware design tradeoffs. Explore the hardware/software co-design and optimize at the system level



Compiler Optimization – Performance Evaluation

- Table compares a couple of performance metrics to run an Alexnet inference for two optimization schemes
- The SystemC time shows roughly 25% performance improvement for layer fusion
- Without hardware calibration, those metrics only provide relative referencing for discussion
- Regarding the first-layer convolution mode is that nv_small outperforms nv_full in the direct mode because more than 80% of zero-padding in the first layer is required for a 3-channel image input

Table 2: Performance Evaluation of bvlc-alexnet.

Optimization	Time	nv_full		nv_small	
Layer fusion		Off	On	Off	On
	SystemC time (ms)	4900	3800	5100	3800
	Guest OS time (sec)	4164	4079	4796	4888
	Hardware layers	42	36	44	38
First-layer convolution	Mode	Direct	Image	Direct	Image
	SystemC time (ms)	4100	3400	3900	3700
	Guest OS time (sec)	3996	1786	4503	3971

ONNC on GitHub – Getting Started

ONNC on GitHub – Getting Started

GitHub, Inc. [US] | https://github.com/ONNC/onnc

Supported platforms

ONNC supports Ubuntu/x86_64 and MacOSX.

Here is a list of verified versions:

- Ubuntu/x86_64
  - 16.04
- MacOSX
  - High Sierra

Getting Started

There are three ways to build ONNC:

1. Build ONNC via Docker
   Please refer to the ONNC Utilities document.
2. Build ONNC via ONNC umbrella
   Please follow the instructions of README.md in **onnc-umbrella**.
   Here is the version of external library we are using in ONNC.
3. Build ONNC without ONNC umbrella
   Please refer to the ONNC Automake build instruction or ONNC CMake build instruction

https://github.com

onnc

Open Neural Network Compiler

C++ ★ 197 37 BSD-3-Clause Updated 3 days ago

onnc-umbrella

umbrella project helps you to build up onnc from scratch

Shell ★ 8 13 BSD-3-Clause Updated 14 days ago

ONNC on GitHub – Build ONNC with Docker Image

GitHub, Inc. [US] | https://github.com/ONNC/onnc/blob/master/docs/ONNC-Utilities.md

4. Build ONNC with the Docker Image

Although the Docker image include a source code tree, it might not be the latest release version of ONNC. We strongly suggest you clone the latest version of ONNC from the GitHub repository, mount the source code directory to the Docker image, and modify the source code with your favorite editor on your host machine. You may clone the source code from the GitHub ONNC repository (https://github.com/ONNC/onnc). To download large model files when cloning the ONNC source, you have to install Git LFS (https://github.com/git-lfs/git-lfs/wiki/Installation) first.

```
$ mkdir -p <source_dir> && cd <source_dir>
$ git clone https://github.com/ONNC/onnc.git
```

Once the latest source code is ready, you may invoke the following command to enter ONNC build environment:

```
$ docker run -ti --rm --cap-add=SYS_PTRACE -v <source_dir>/onnc:/onnc/onnc onnc/onnc-community
```

`<source_dir>` is the directory where you cloned the latest ONNC source code and the `-ti` option provides a interactive interface for the container, the `-v` option mounts the directory to the Docker image, the `--cap-add=SYS_PTRACE` option enables debug support (e.g. gdb) in the container. You can make some change to the source code (`<source_dir>/onnc`) and run the following command to build ONNC.

```
// run in the container cli, under build directory `/onnc/onnc-umbrella/build-normal` by default
$ smake -j8 install
```

ONNC on GitHub – Running ONNX Models

GitHub, Inc. [US] | https://github.com/ONNC/onnc/blob/master/docs/ONNC-Utilities.md

Running a Single Benchmark

You may run a single model for benchmarking using the following shell command:

```
// run in the container cli
$ onni <model_file_path>/model.onnx <input_file_path>/input_0.pb -verbose=<level>
```

where `<model_file_path>` is the path to the model file for the pre-trained ONNX model and `<input_file_path>` is the path to the corresponding input file. In the ONNC Docker container, the model file path is `/models/<model_name>` and the input file path is `/models/<model_name>/test_data_set_<0~6>`. `<level>` indicates different levels of verbose information. Higher-level information is a superset of all lower-level information. For example, level 4 will include all information from level 1 to level 4.

Information for each verbose level:

- Level 1: Inference time & memory usage
- Level 2: ONNX operator statistics
- Level 3: Inference time & ONNX operator statistics per layer
- Level 4: Memory allocation log

Here is an example of running AlexNet and printing out all information.

```
// run in the container cli
$ onni /models/bvlc_alexnet/model.onnx /models/bvlc_alexnet/test_data_set_0/input_0.pb -verbose=4
```

Running ONNX models

```
$ onni /models/bvlc_alexnet/model.onnx /models/bvlc_alexnet/test_data_set_0/input_0.pb -verbose=4

[Statistics] Operator | Count |Description
[Statistics] ---------+------
[Statistics]  Softmax |     1 |count for Softmax
[Statistics]     Conv |     5 |count for Conv
[Statistics]     Gemm |     3 |count for Gemm
[Statistics]      LRN |     2 |count for LRN
[Statistics]     Relu |     7 |count for Relu
[Statistics]  MaxPool |     3 |count for MaxPool
[Statistics]  Reshape |     1 |count for Reshape
[Statistics] ---------+------
[Statistics]    Total |    22
[v1] weight memory: 243860912
[v1] internal memory: 2239488
```

```
$ onni /models/resnet50/model.onnx /models/resnet50/test_data_set_0/input_0.pb -verbose=4

[Statistics]          Operator | Count |Description
[Statistics] -------------------+------
[Statistics]               Sum |    16 |count for Sum
[Statistics]           Softmax |     1 |count for Softmax
[Statistics]              Conv |    53 |count for Conv
[Statistics]              Gemm |     1 |count for Gemm
[Statistics]              Relu |    49 |count for Relu
[Statistics]       AveragePool |     1 |count for AveragePool
[Statistics]           MaxPool |     1 |count for MaxPool
[Statistics] BatchNormalization |    53 |count for BatchNormalization
[Statistics]           Reshape |     1 |count for Reshape
[Statistics] -------------------+------
[Statistics]             Total |   176
[v1] weight memory: 102440624
[v1] internal memory: 12042240
```

https://onnc.ai

人工智能系统与集成电路研讨会

2019年8月18-19日