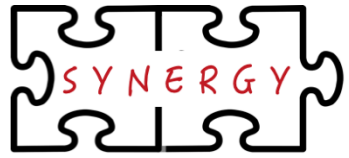


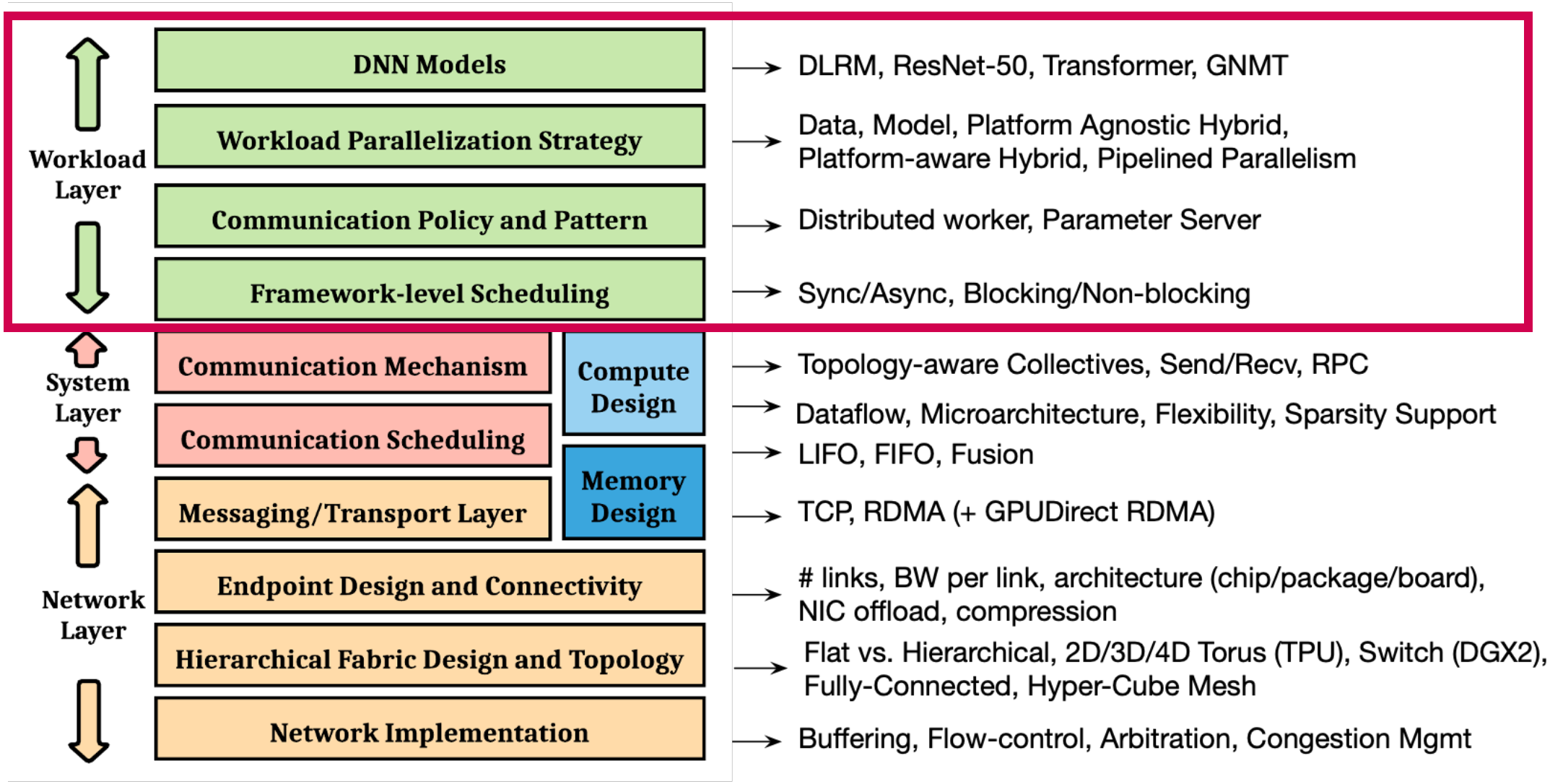
https://synergy.ece.gatech.edu

ASTRA-sim Tutorial
@MICRO 2024
Nov 3rd

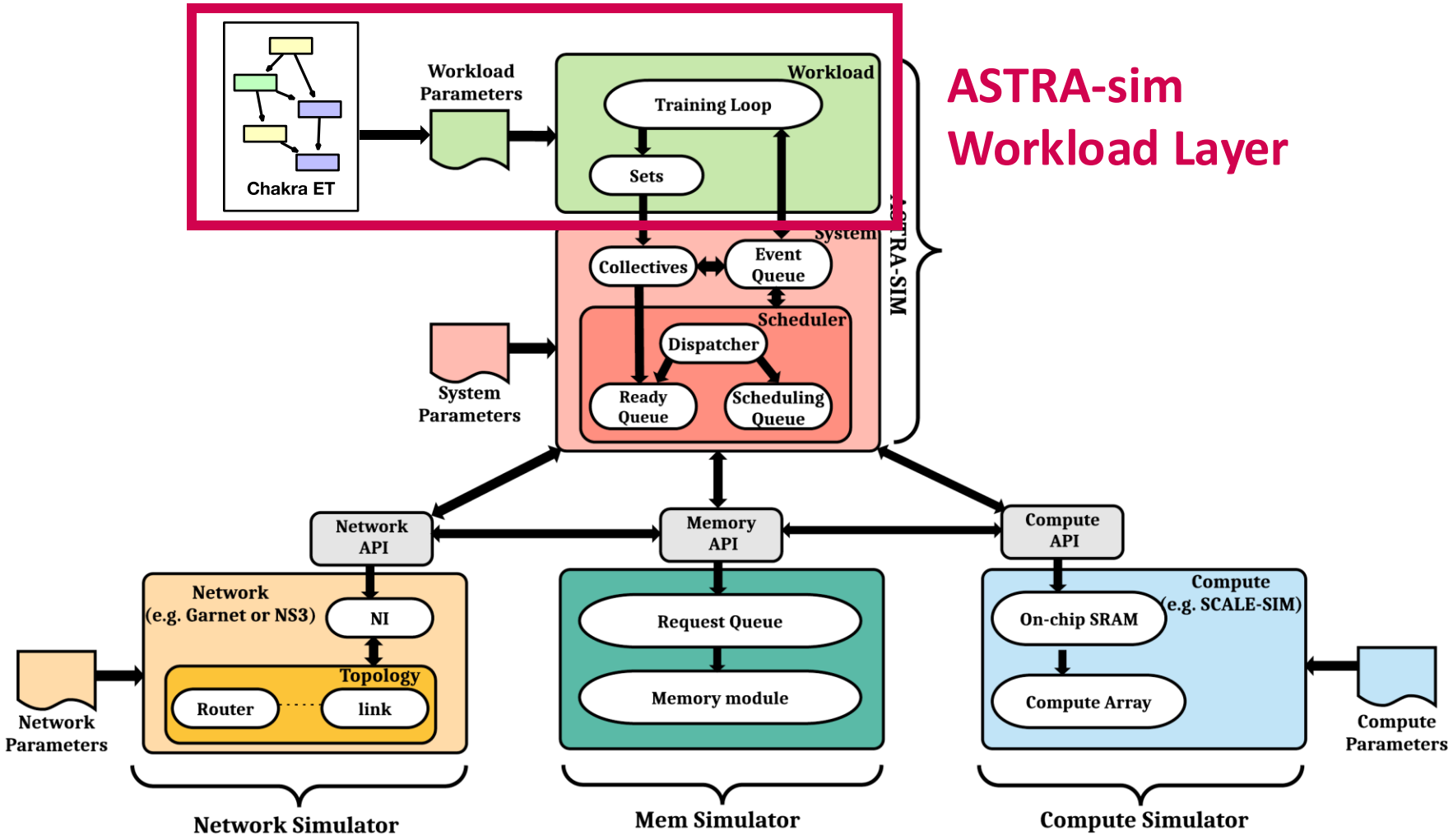
ASTRA-sim and Chakra Tutorial:
Workload Layer

Taekyung Heo
NVIDIA

Design Space: Workload



ASTRA-sim: Workload Layer



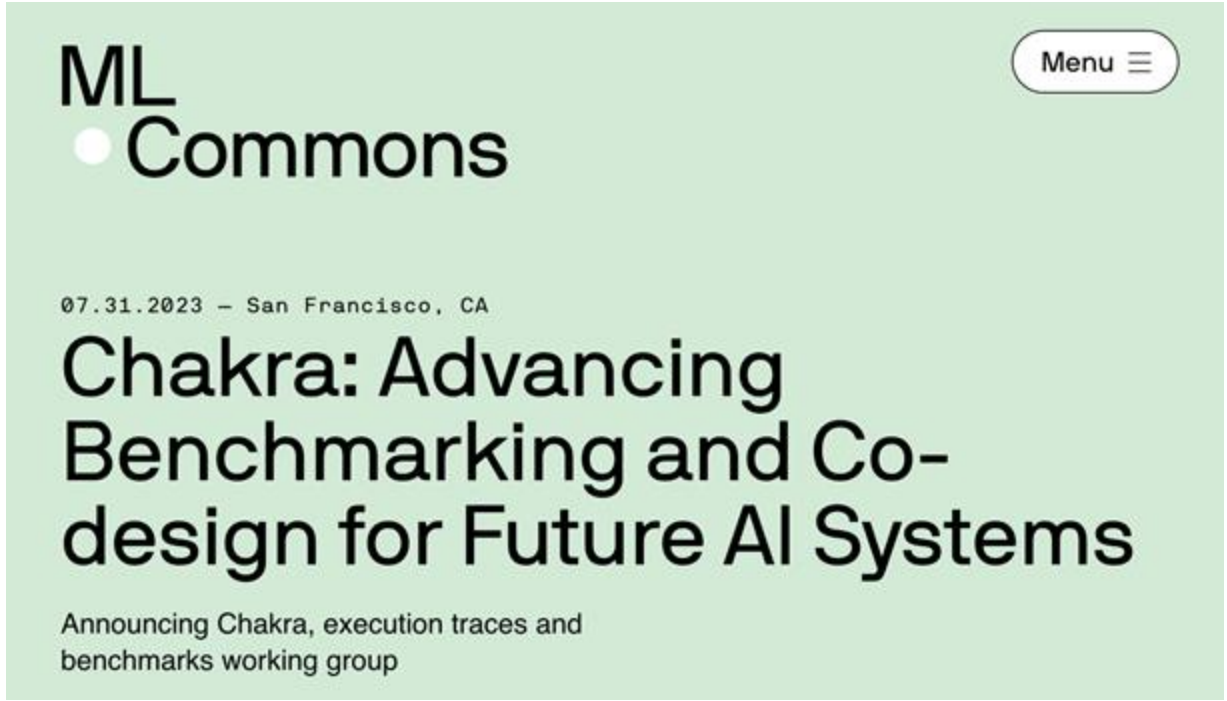
Workload Layer

- Workload layer captures **workload-specific characteristics**
  - DNN **Model**
  - **Parallelization** strategies
  - Control and Data **dependencies**
  - Compute and Communication **order**

- All workload characteristics are captured through MLCommons **Chakra Execution Trace** representation

Chakra Execution Trace

- Standardized **distributed ML workload representation**

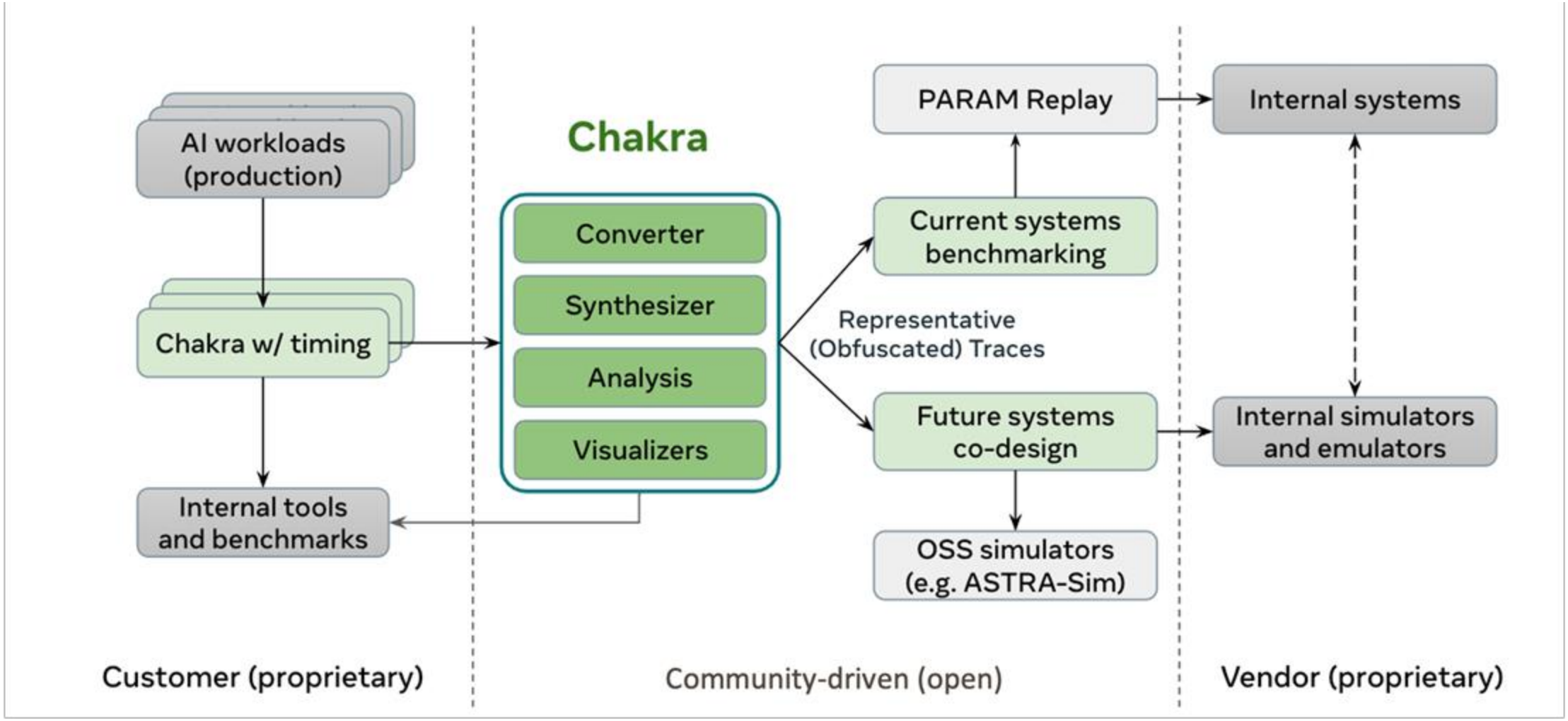


MLCommons

Paper

https://mlcommons.org/working-groups/research/chakra
https://arxiv.org/abs/2305.14516

Chakra Ecosystem



Chakra ET: Basic Structure

- Three types of **Chakra ET Nodes** (basic building blocks)

chakra/schema/protobuf/et_def.proto

Compute Node

- Encapsulates distributed ML compute operations
  - Mostly GEMM + other kernels

Compute

- **#FLOPs**
- **CPU Operation?**
- **Operand Tensor Size**
- **Estimated/measured compute time**

Communication Node

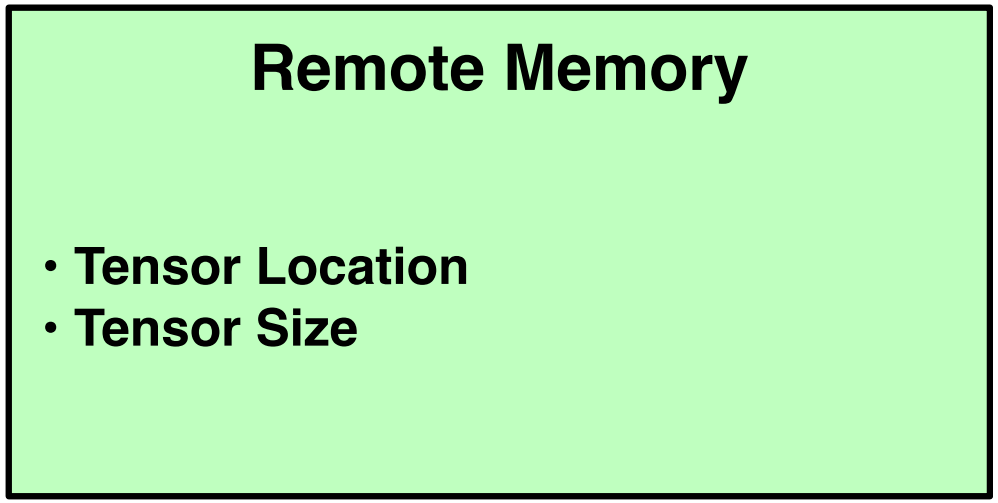
- Captures single send-receive pairs, or collective communications

Communication

- **Communication Type**
- **Communication Size**
- **Involved NPUs**
- **Priority**

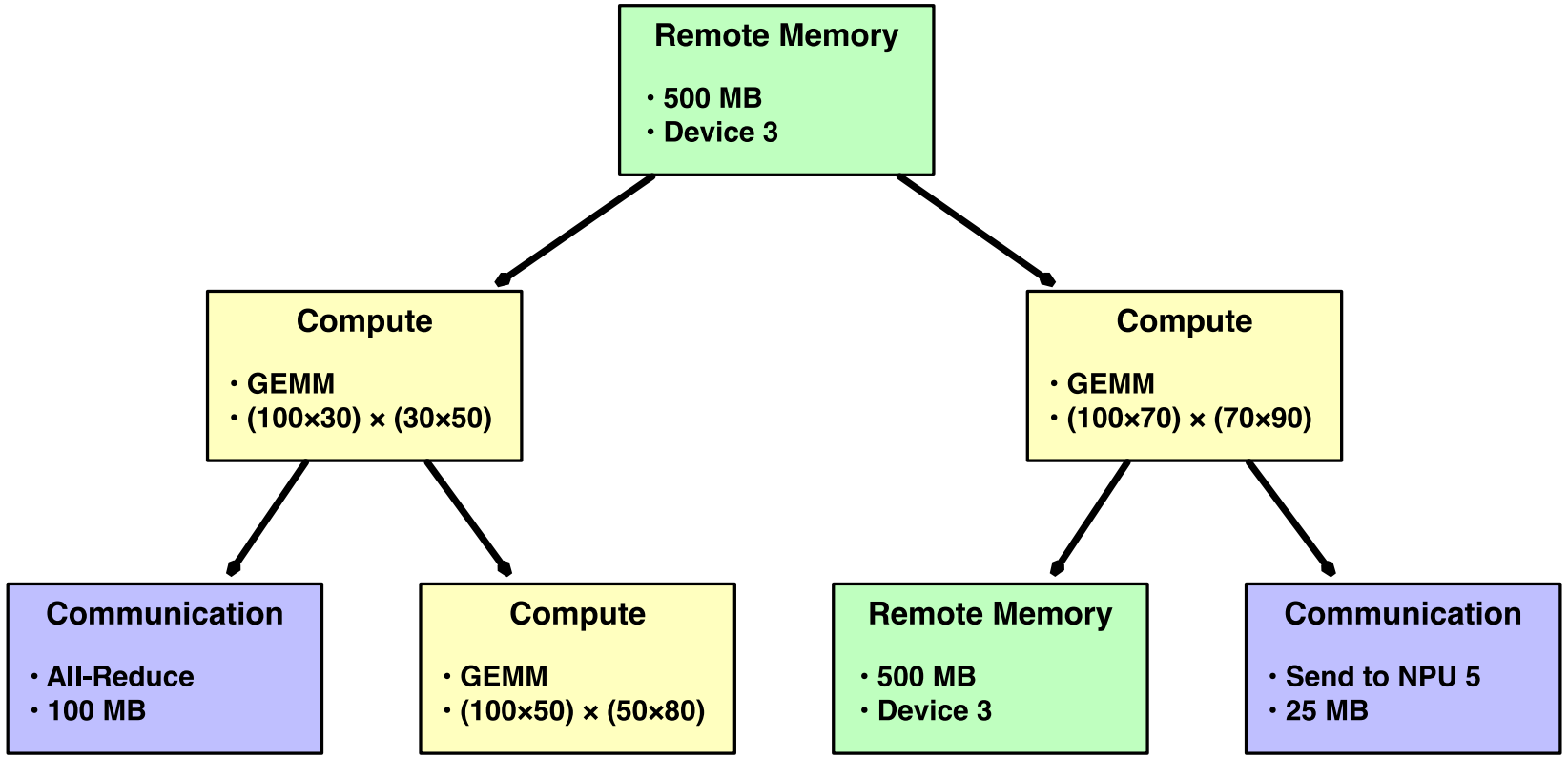
Remote Memory Node

- Models remote (e.g., pooled or disaggregated) memory accesses

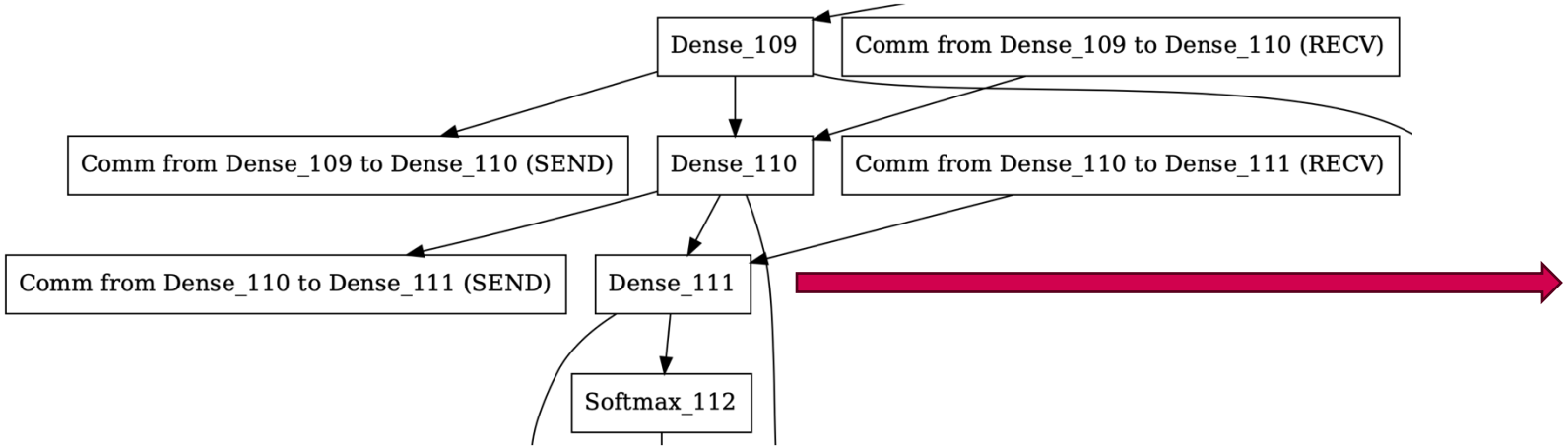


Chakra ET:

- Arbitrary distributed ML workload is represented in Directed Acyclic Graphs (DAGs)



Example Chakra ET




```
"id": "5526",
"name": "aten::transpose",
"type": "COMP_NODE",
"dataDeps": [
  "5525"
],
"inputs": {
  "values": "[[5524, 5133, 0, 829
  "shapes": "[[288, 288], [], []]"
  "types": "['Tensor(c10::BFloat16
},
"outputs": {
  "values": "[[5528, 5133, 0, 82944
  "shapes": "[[288, 288]]",
  "types": "['Tensor(c10::BFloat16)
},
"attr": [
  {
    "name": "is_cpu_op",
    "boolVal": true
  },
```

Chakra ET Generation

- Via Provided **ET Generation API**
- Wrapper/Automation of APIs to support End-to-End Workloads
  - **Text-based** representation to e2e Chakra ET
  - **Synthetic** e2e Chakra ET generator for transformer-based LLMs
- **Profiling/Collection** from real system PyTorch execution

Will be covered in Demo session

Chakra ETFeeder

- Offers clean APIs to read and manage Chakra ETs

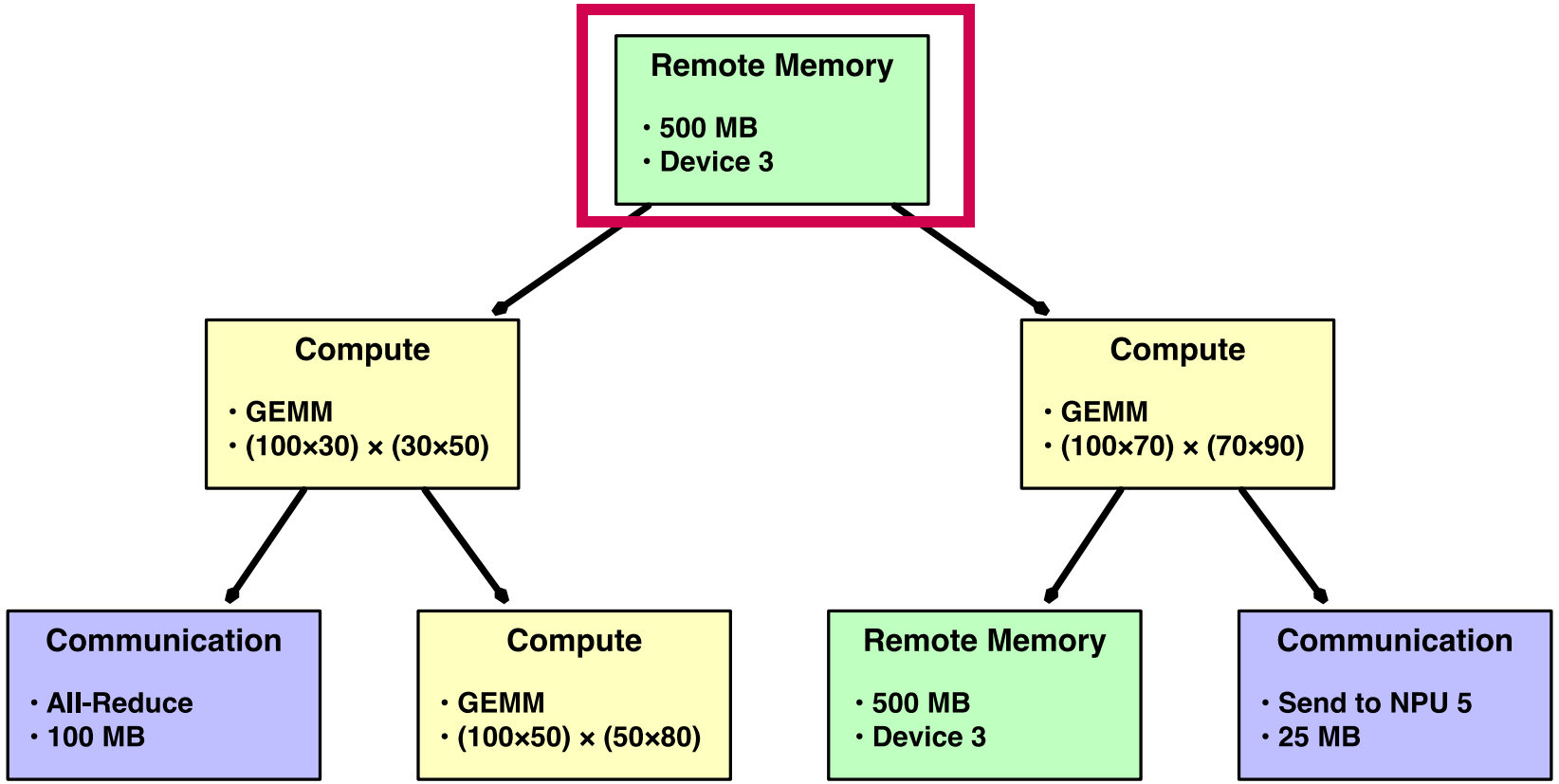
chakra/src/feeder/et_feeder.h

`getNextIssuableNode()` → Returns a dispatchable, free ET Node

`freeChildrenNodes(node_id)` → Mark a node as finished

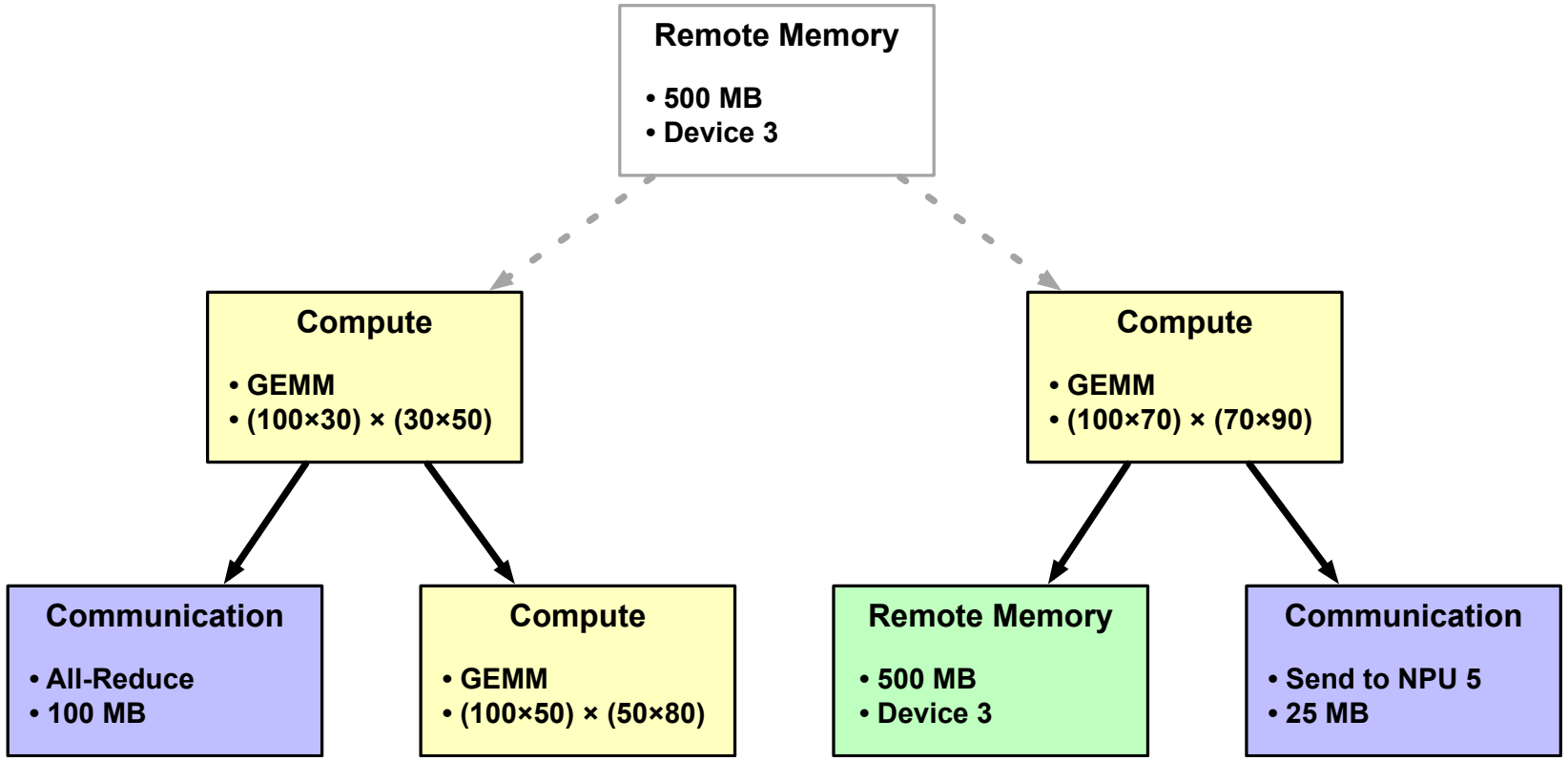
Chakra ETFeeder

getNextIssuableNode()



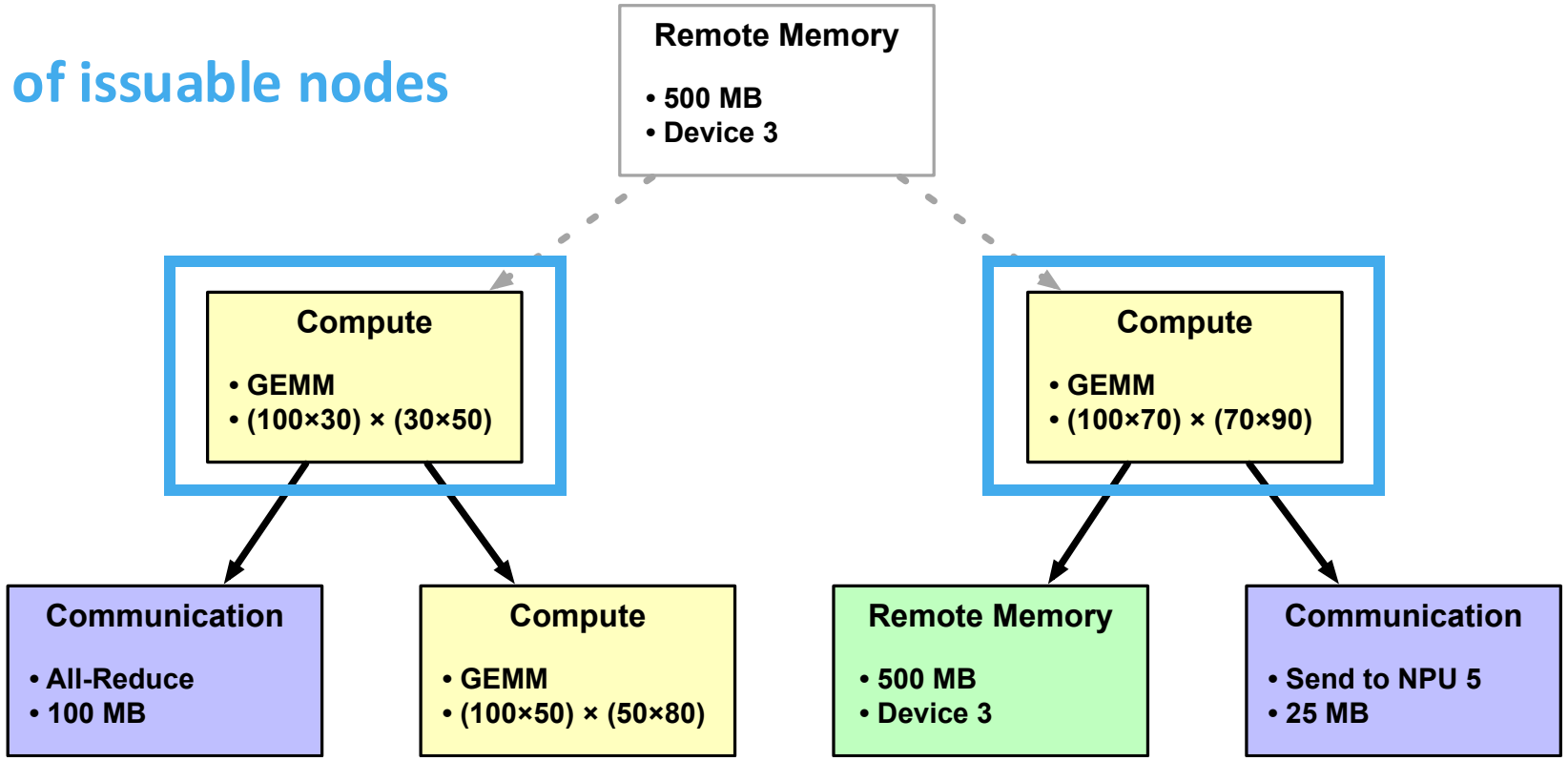
Chakra ETFeeder

freeChildrenNodes()

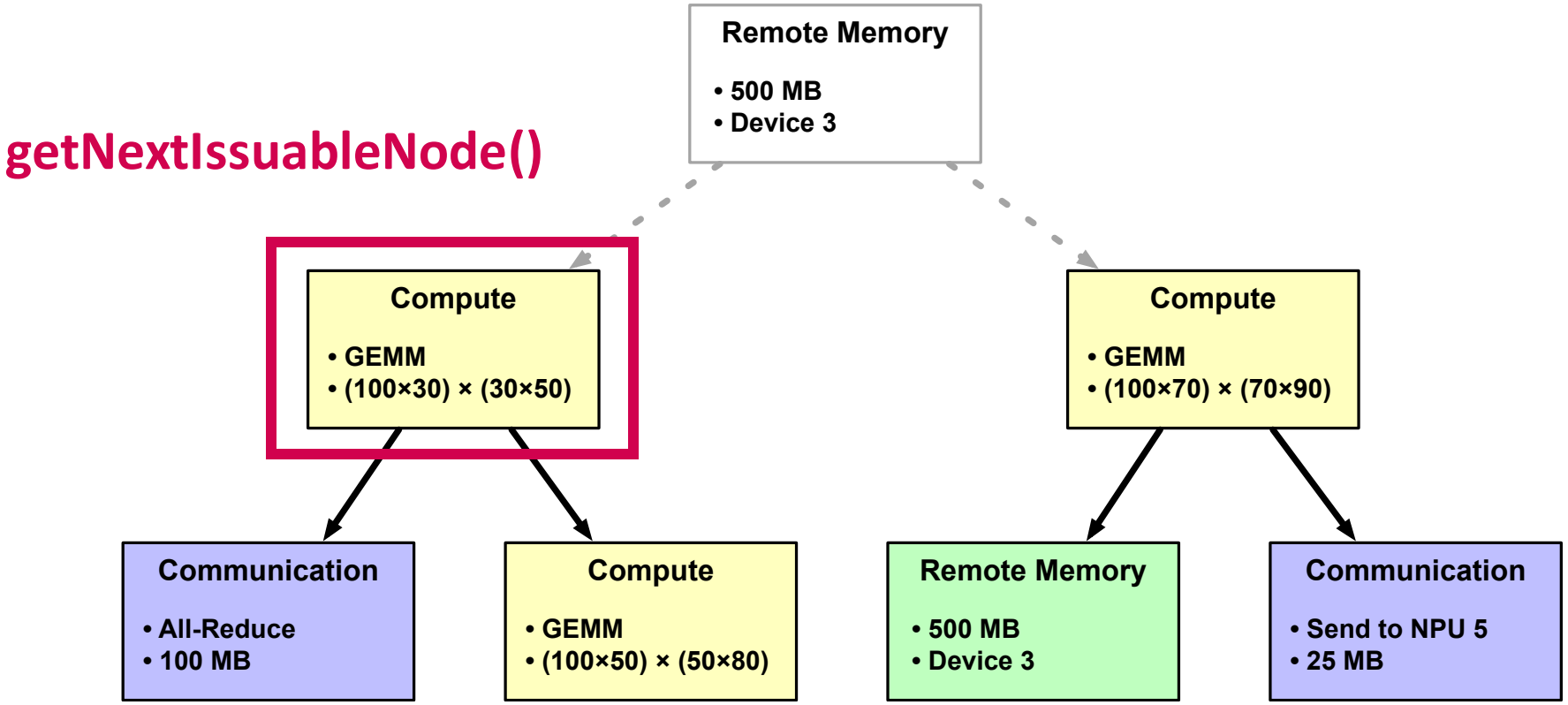


Chakra ETFeeder

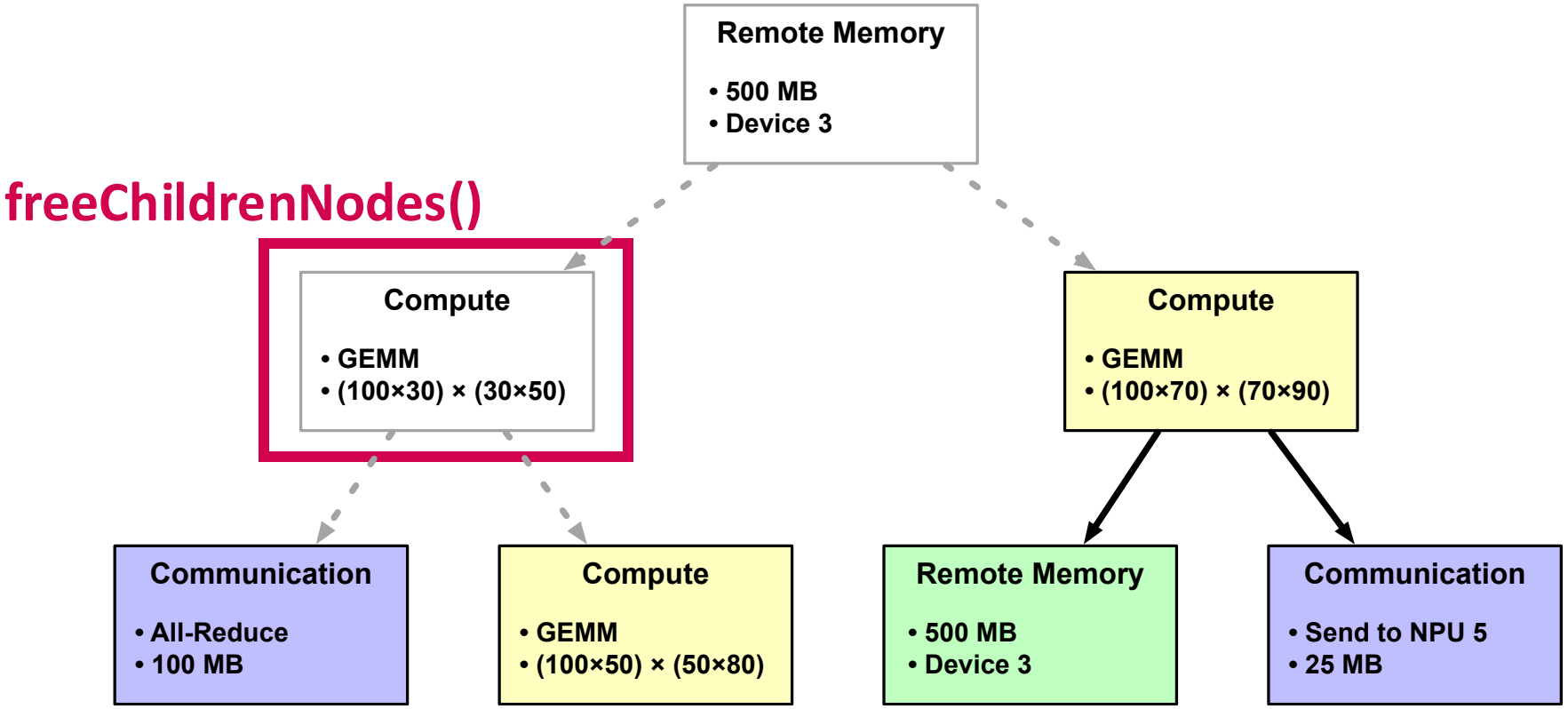
list of issuable nodes



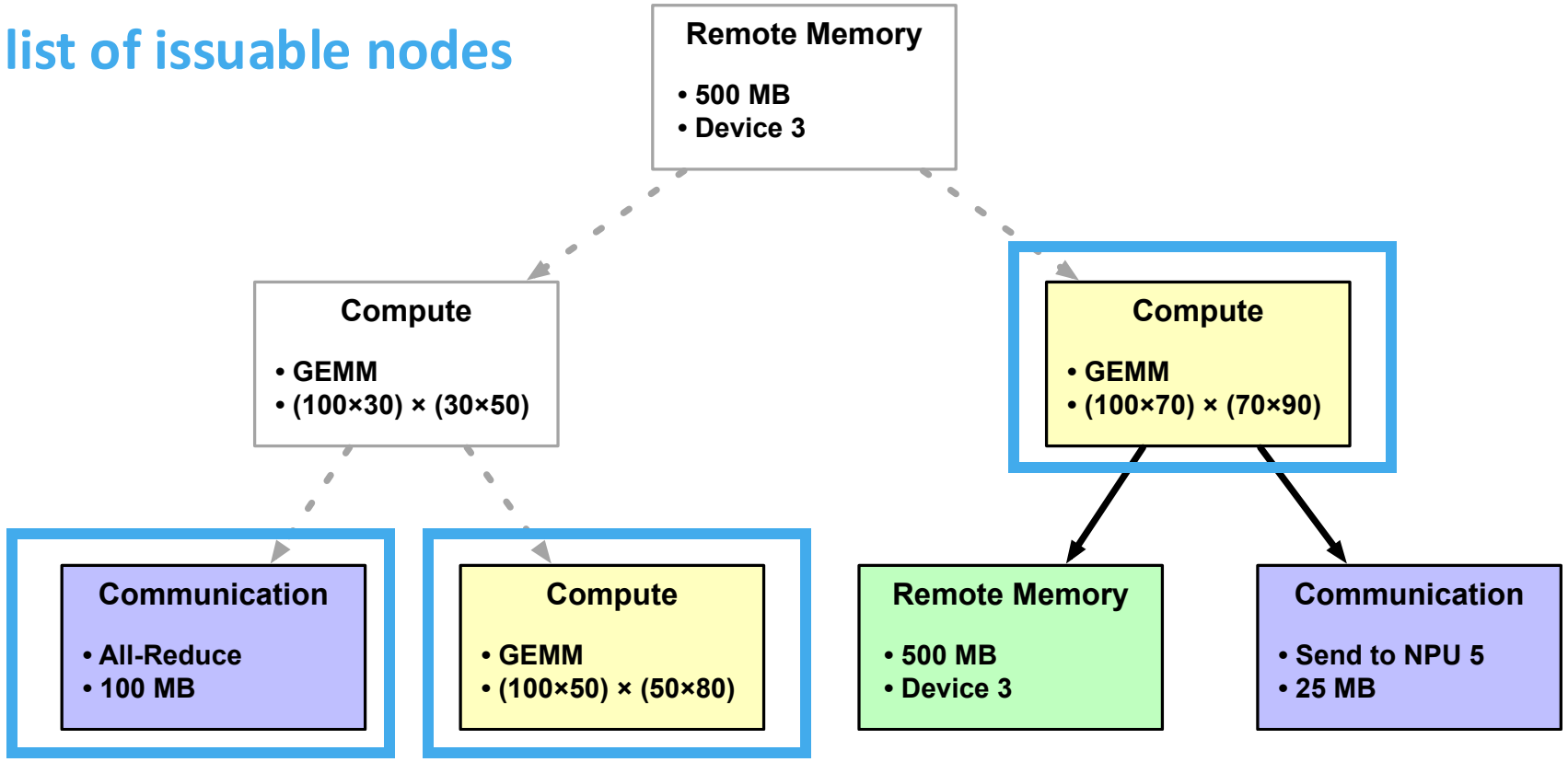
Chakra ETFeeder



Chakra ETFeeder



Chakra ETFeeder



Pseudocode: ASTRA-sim Workload Layer

- Iterate over:
  - Get Issuable Node
  - "Issue" the node appropriately

```
while not Finished:
    node = getNextIssuableNode()

    issue_simulation(node)

    freeChildrenNodes(node)
```

ASTRA-sim Workload Layer

astra-sim/workload/Workload.cc

```
node = et_feeder->getNextIssuableNode();

while (node != nullptr) {
        if (hw_resource->is_available(node)) {
                issue(node);
        } else {
                push_back_queue.push(node);
        }
        node = et_feeder->getNextIssuableNode();
}
```

get an issuable node

"issue" the operation if HW is free

repeat the process

Issue() method

- Trigger appropriate ASTRA-sim methods

```
void Workload::issue(node) {
        if (node->type() == MEM_STORE_NODE) {
                issue_remote_mem(node);
        } else if (node->type() == COMP_NODE) {
                issue_comp(node);
        } else if (node->type() == COMM_COLL_NODE) {
                issue_comm(node);
        } else if (node->type() == INVALID_NODE) {
                skip_invalid(node);
        }
}
```

check node type

trigger appropriate method

Issuing Computation

- Estimate compute time: via Roofline model
- Register an event handler

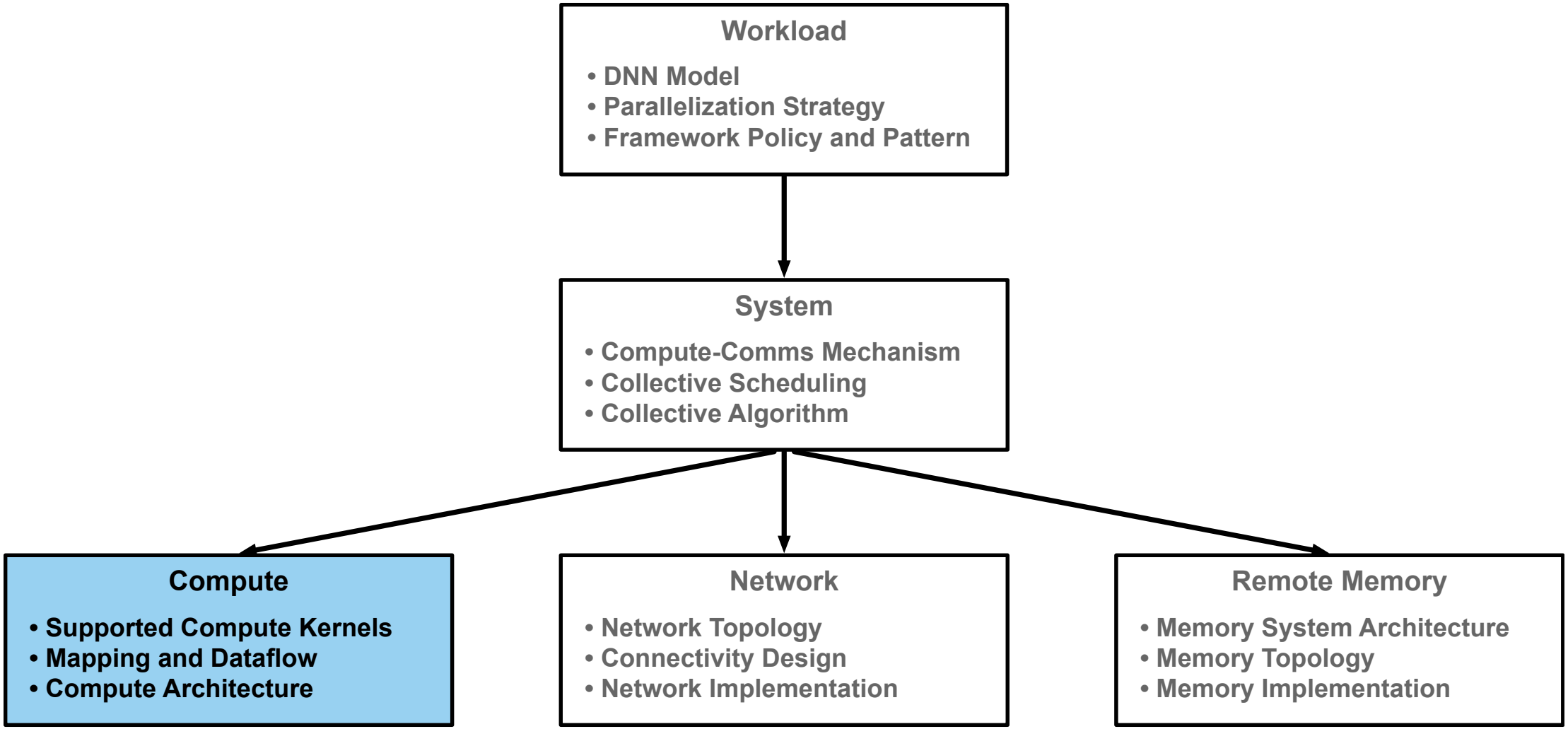
```
void Workload::issue_comp(node) {
    hw_resource->occupy(node);

    double operational_intensity = static_cast<double>(node->num_ops()) / static_cast<double>(node->tensor_size());
    double perf = sys->roofline->get_perf(operational_intensity);
    double elapsed_time = static_cast<double>(node->num_ops()) / perf;
    uint64_t runtime = static_cast<uint64_t>(elapsed_time);

    sys->register_event(this, EventType::General, wlhd, runtime);

}
```

occupy HW resource

estimate runtime

register event at the end of "runtime"

Design Space: Compute



Sneak Peek: ComputeAPI

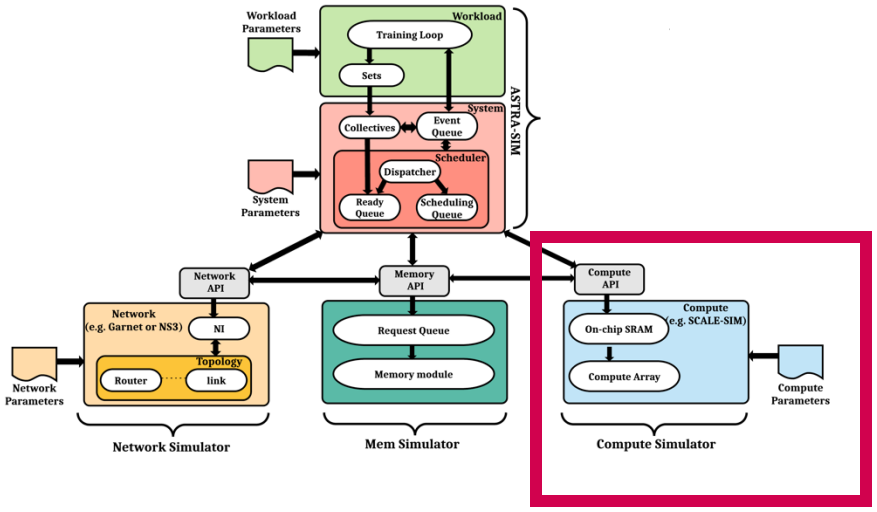
- Model different compute models via Roofline setup

```
"peak-perf": 60
"local-mem-bw": 50
```

TFLOPS of compute device

Local memory BW

- Flexibility via ComputeAPI




```
void Workload::issue_comp(node) {
    hw_resource->occupy(node);

    simulate(node->kernel, this, EventType::General);
}
```

Offload to a separate compute simulator/modeling

Work in Progress!

https://github.com/astra-sim/astra-sim/pull/185

Issuing Communication

- Trigger appropriate System layer methods (covered next)

```
void Workload::issue_comm(node) {
      hw_resource->occupy(node);

      if (node->comm_type() == ChakraCollectiveCommType::ALL_REDUCE) {
            DataSet* fp = sys->generate_all_reduce(node->comm_size(), ...)

            fp->set_notifier(EventType::CollectiveCommunicationFinished);

      }

      (...)
}
```

occupy HW resource

start All-Reduce

register event at the end of "runtime"

Event Handler

- Release HW occupancy
- Free child nodes
- Issue next nodes, if there's one

```
void Workload::call(event, data) {
    hw_resource->release(node);

    et_feeder->freeChildrenNodes(node->id());

    issue_dep_free_nodes();
}
```

release HW resource

mark as done!

repeat the process