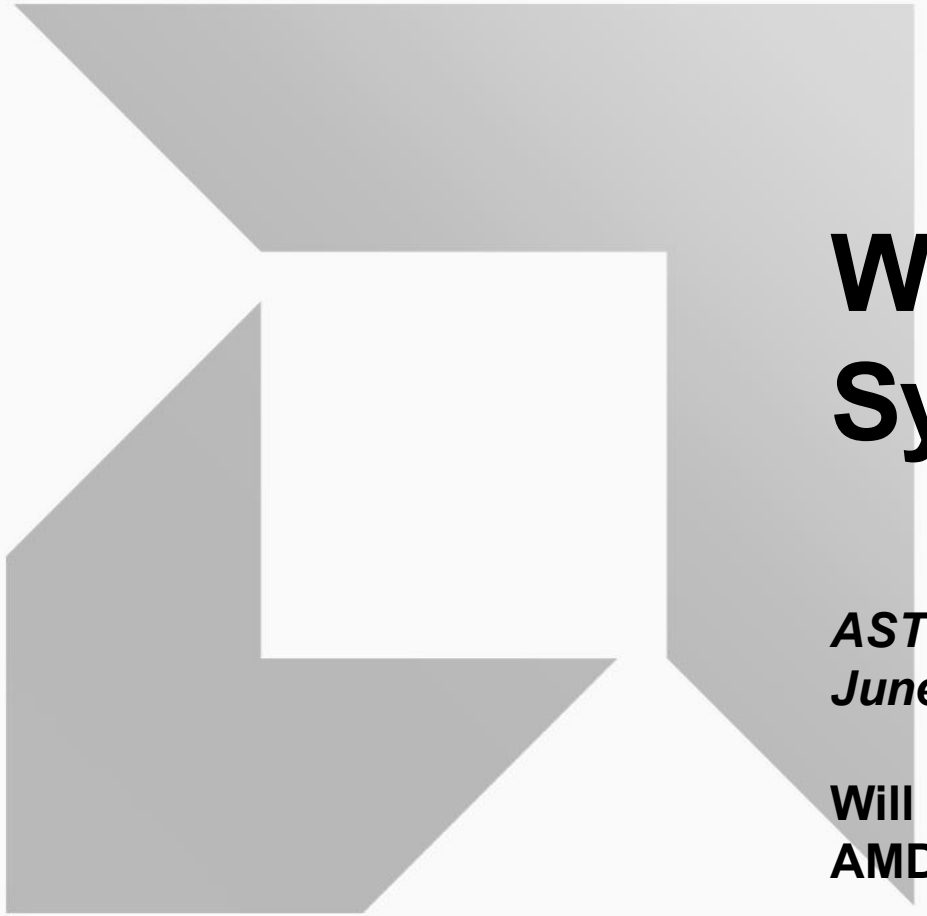


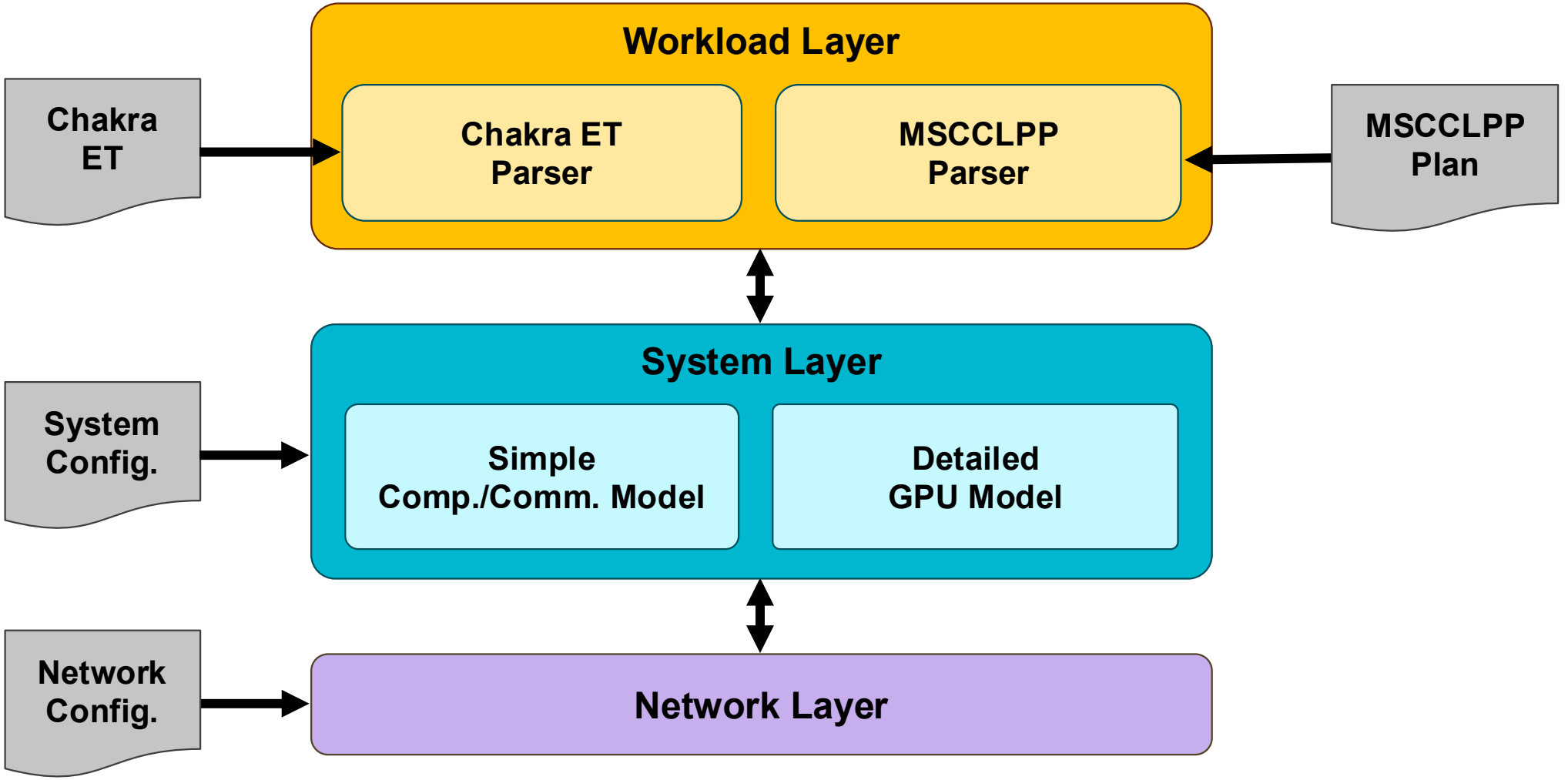
Workload and System Layers

ASTRA-sim Tutorial at ISCA '26
June 27, 2026

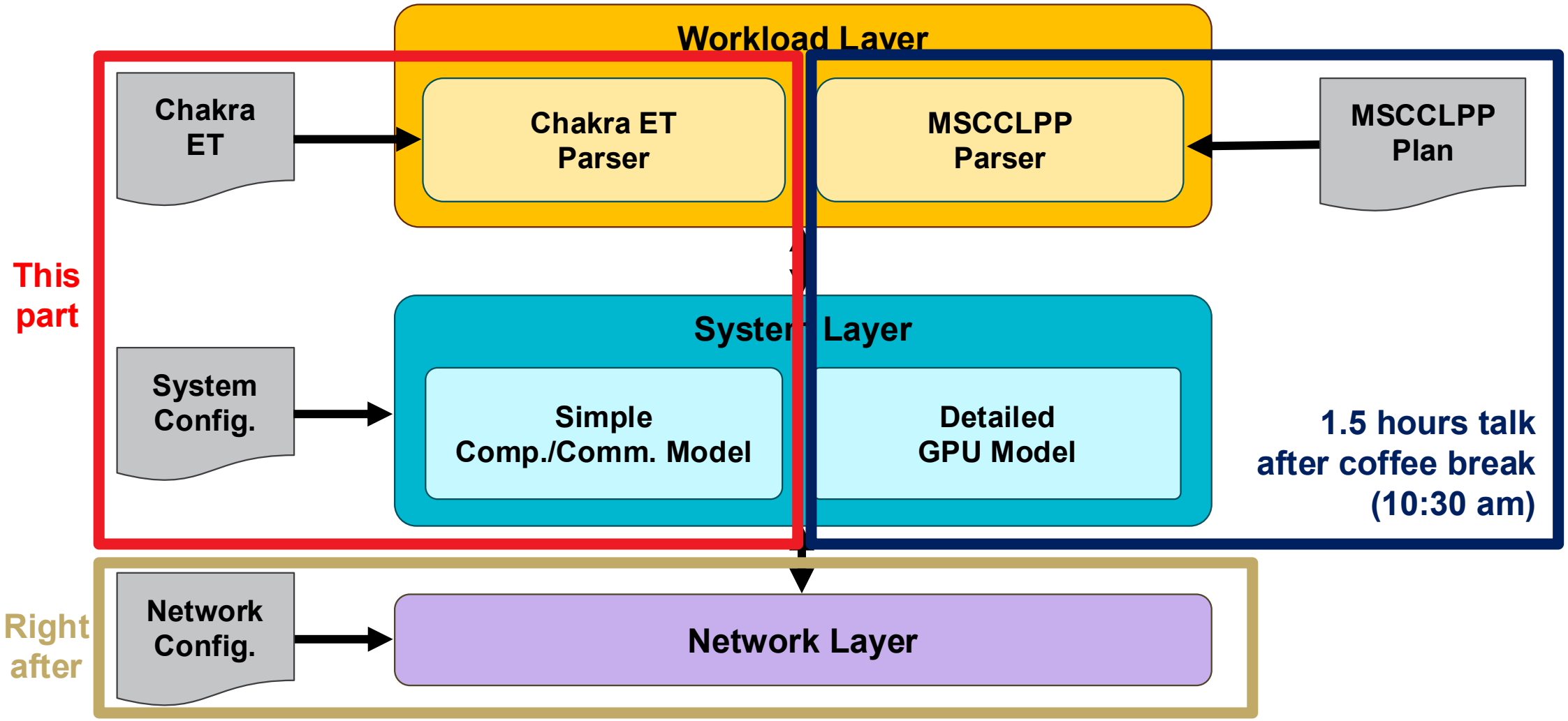
Will Won <William.Won@amd.com>
AMD Research and Advanced Development

AMD
together we advance_

ASTRA-sim Overview



ASTRA-sim Overview



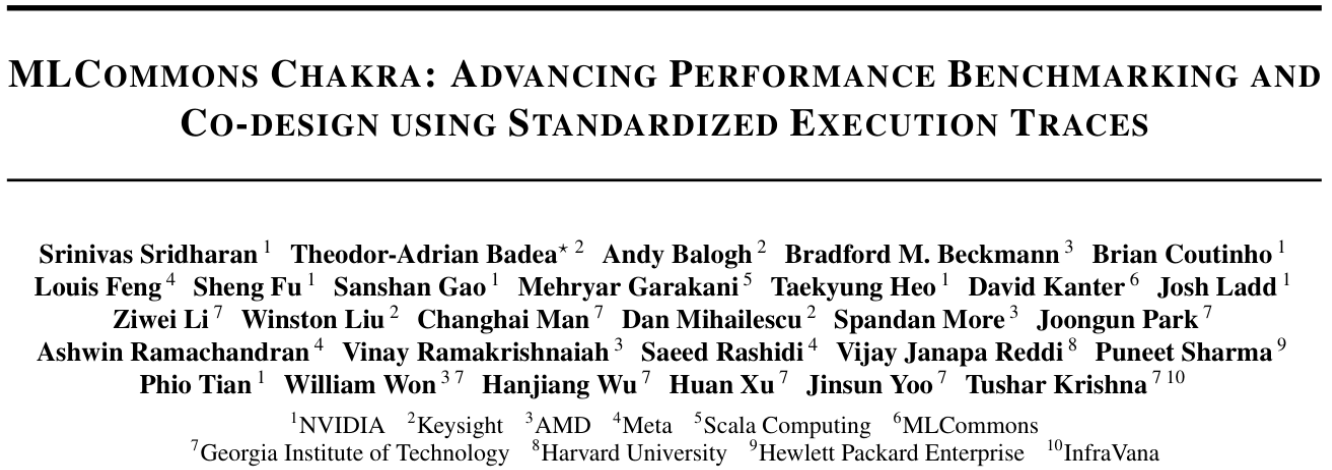
Workload and System Layers

- Workload layer captures/simulates kernel-level, workload-specific characteristics
  - ML **model**
  - **Parallelization** strategies
  - Compute and communication **orders**
  - **Dependencies** across different kernels

- System layer executes the kernel, dispatches events to the simulation backend
  - **Compute** kernel → **estimates compute time**
  - **Communication** kernel → **breaks down collective** into network simulation units, **delegates** to the network simulator

Chakra Execution Trace

- Workload trace representation: **MLCommons Chakra** execution trace (ET)
  - **Standardized** traces for distributed ML workload representation
- Check the:
  - Working group: **https://mlcommons.org/working-groups/research/chakra/**
  - MLSys paper: **https://doi.org/10.48550/arXiv.2605.11333**
  - GitHub: **https://github.com/mlcommons/chakra**



MLCOMMONS CHAKRA: ADVANCING PERFORMANCE BENCHMARKING AND CO-DESIGN USING STANDARDIZED EXECUTION TRACES

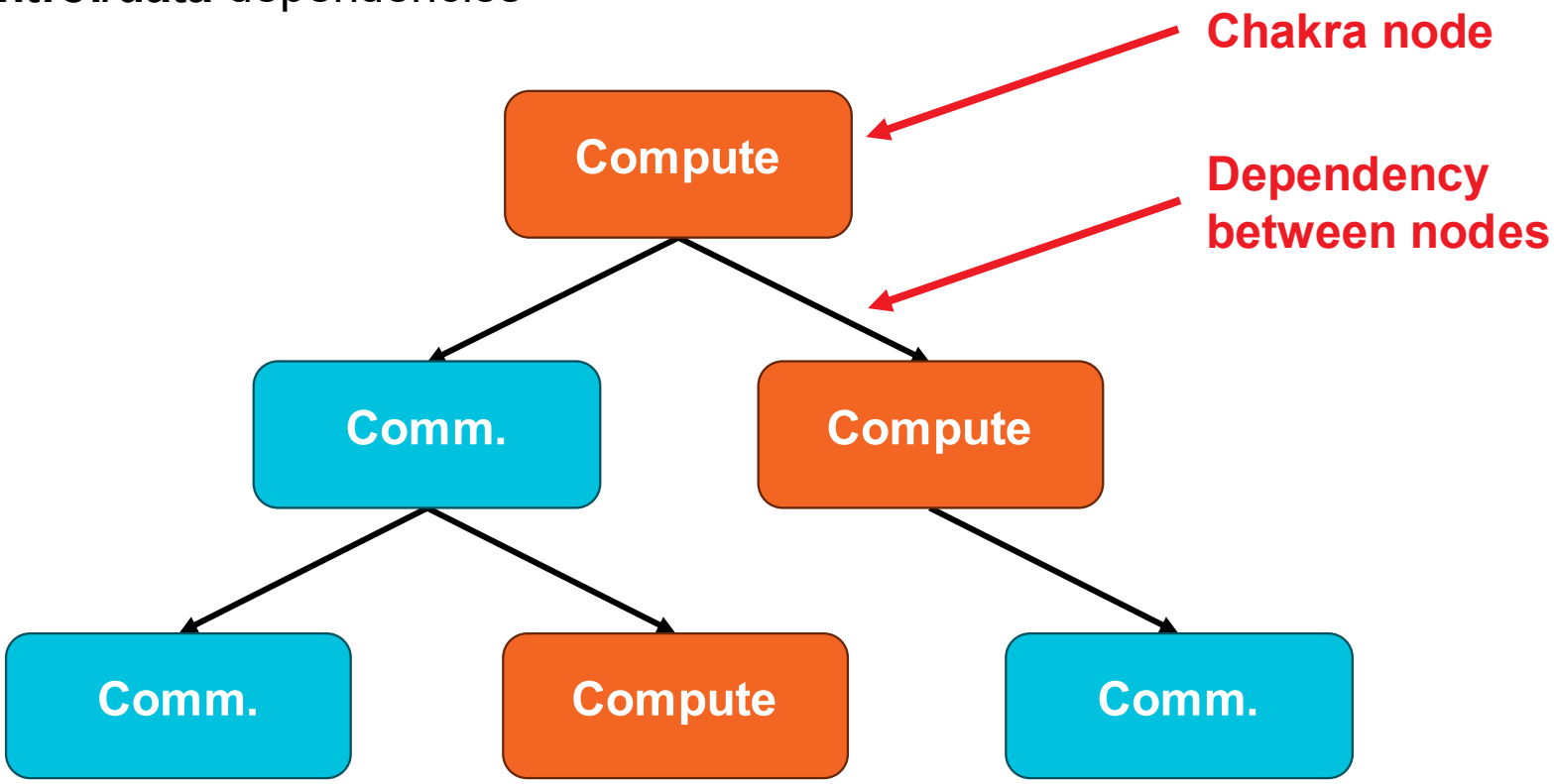
Srinivas Sridharan [1] Theodor-Adrian Badea* [2] Andy Balogh [2] Bradford M. Beckmann [3] Brian Coutinho [1]
Louis Feng [4] Sheng Fu [1] Sanshan Gao [1] Mehryar Garakani [5] Taekyung Heo [1] David Kanter [6] Josh Ladd [1]
Ziwei Li [7] Winston Liu [2] Changhai Man [7] Dan Mihailescu [2] Spandan More [3] Joongun Park [7]
Ashwin Ramachandran [4] Vinay Ramakrishnaiah [3] Saeed Rashidi [4] Vijay Janapa Reddi [8] Puneet Sharma [9]
Phio Tian [1] William Won [3 7] Hanjiang Wu [7] Huan Xu [7] Jinsun Yoo [7] Tushar Krishna [7 10]

[1]NVIDIA [2]Keysight [3]AMD [4]Meta [5]Scala Computing [6]MLCommons
[7]Georgia Institute of Technology [8]Harvard University [9]Hewlett Packard Enterprise [10]InfraVana

S. Sridharan et al., "MLCommons Chakra: Advancing Performance Benchmarking and Co-design using Standardized Execution Traces," in Proc. of the 9th Conference on Machine Learning and Systems (MLSys), 2026.

Chakra ET: Structure

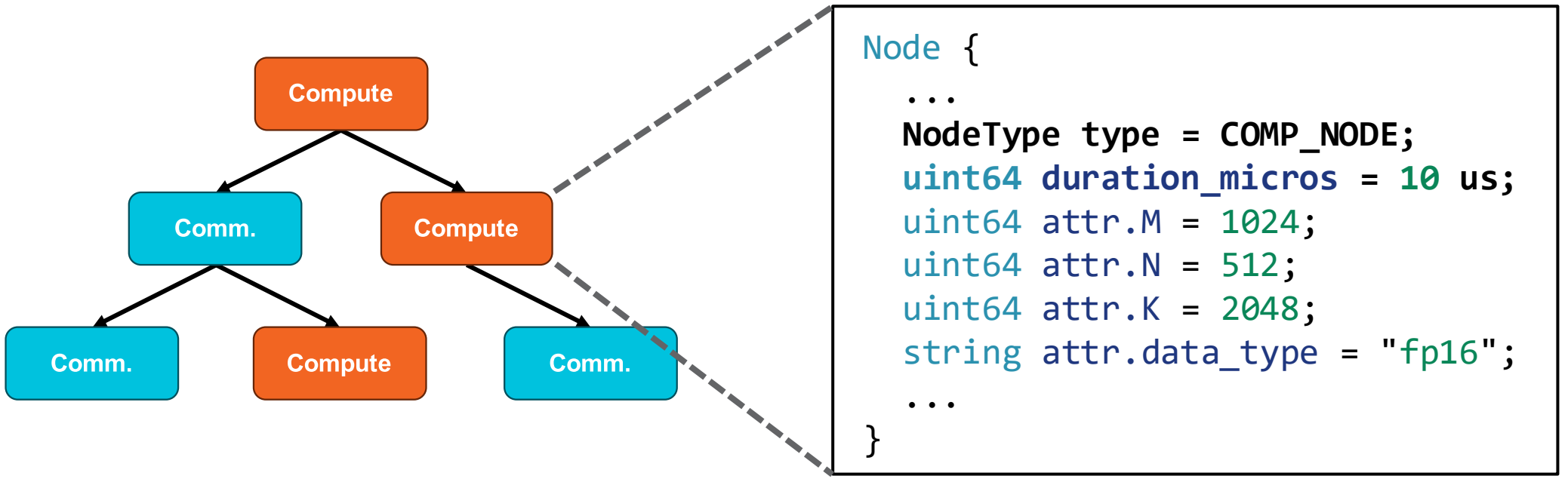
- A directed acyclic graph (DAG) of Chakra **nodes**
- Edges represent **control/data** dependencies



Example Chakra ET with six nodes

Chakra ET: Compute Node

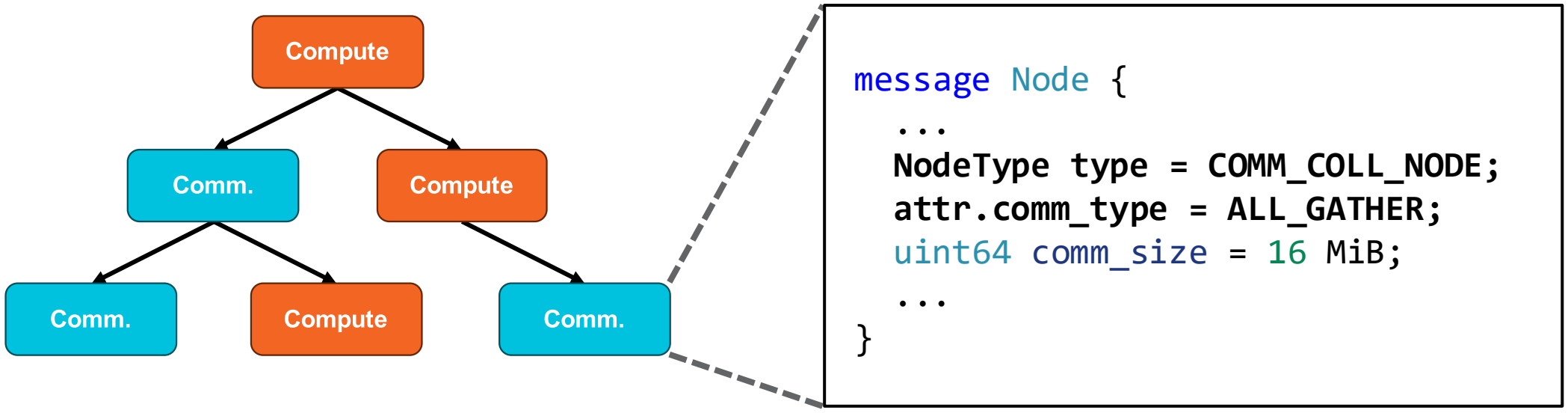
- Captures **compute kernel metadata**
  - e.g., (measured) compute durations, GEMM size, data types



Example compute node

Chakra ET: Communication Node

- Captures **communication kernel metadata**
  - e.g., collective type, buffer size, source/destination GPUs

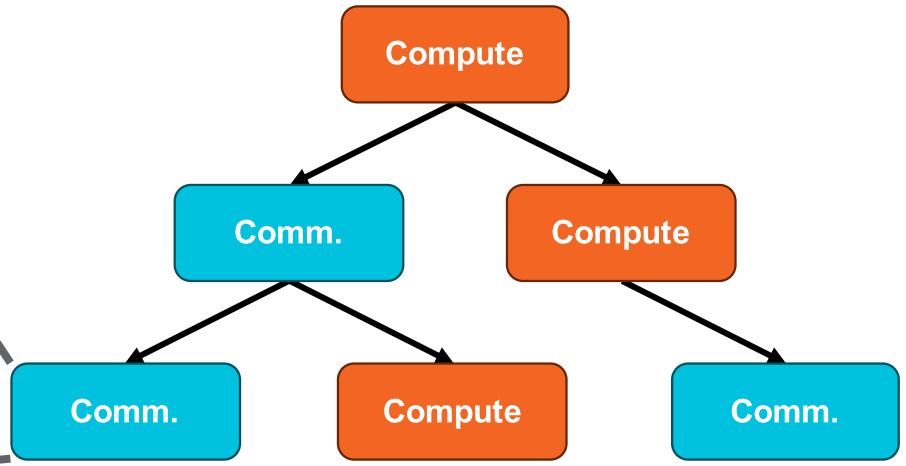


Example: All-Gather node

Chakra ET: Communication Node (cont'd)

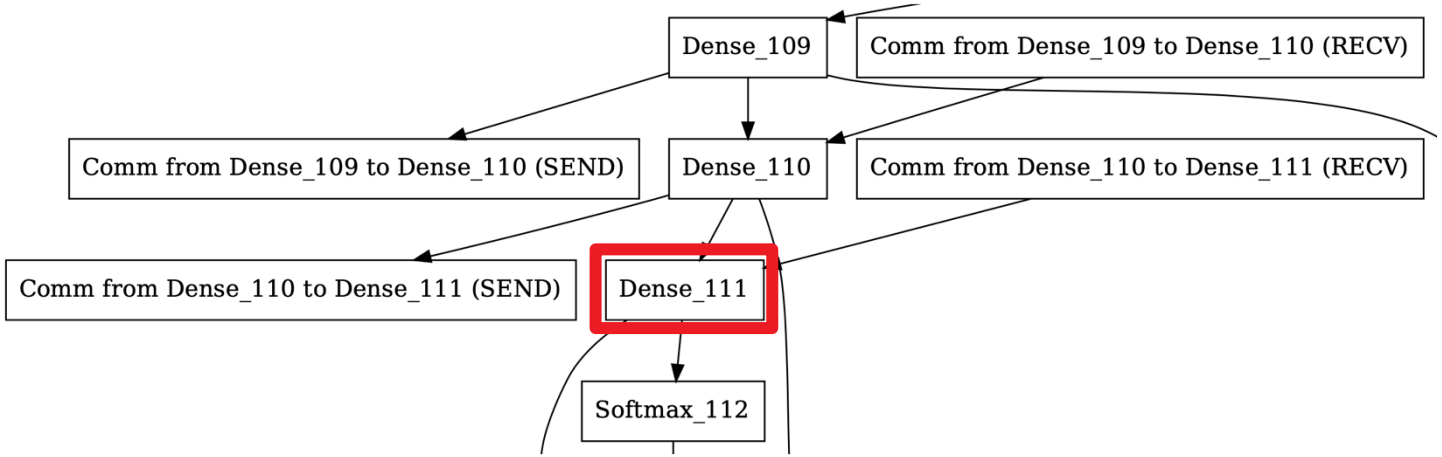
- Captures **communication kernel metadata**
  - e.g., collective type, buffer size, source/destination GPUs

```
message Node {
  ...
  NodeType type = COMM_SEND_NODE
  uint32 attr.comm_dst = GPU 5;
  uint64 comm_size = 1 MiB;
  ...
}
```



Example: single send node

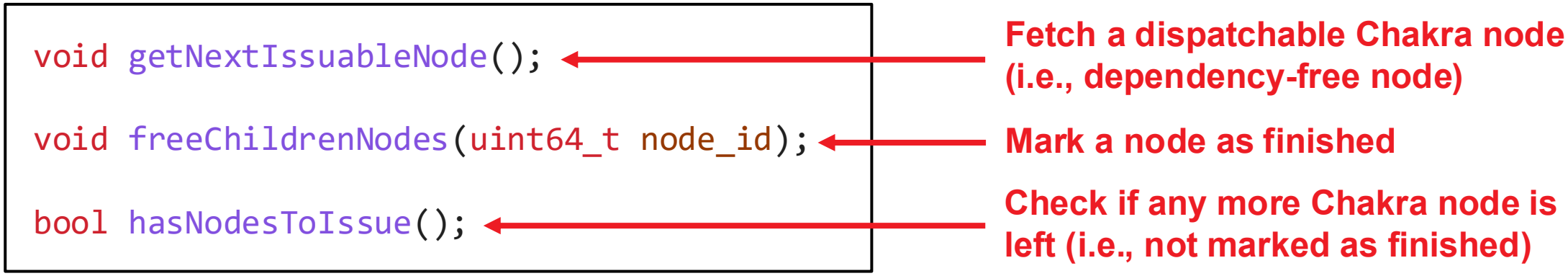
Chakra ET Example




```
"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 APIs

- Chakra's `ETFeeder` provides application programming interfaces (APIs) to query Chakra ET

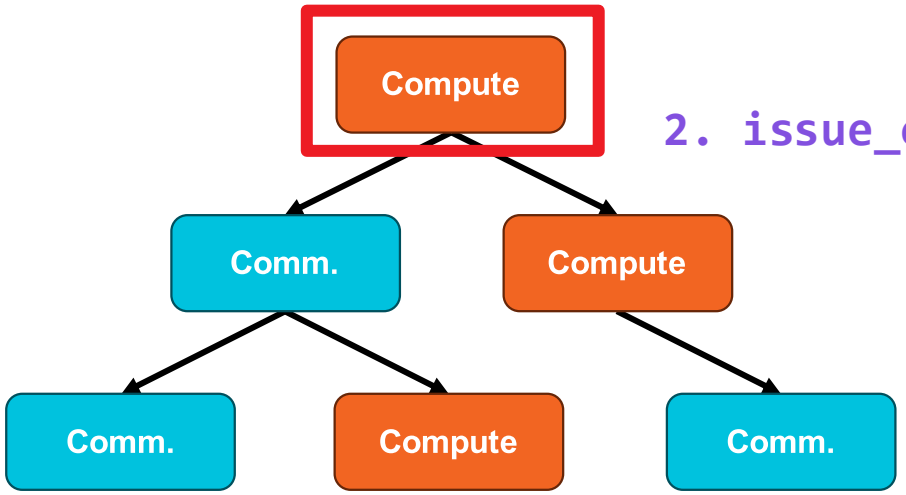


`Chakra:src/feeder/wrapper_node.h`

Workload Layer: Issuing a Node

- First, **query** the Chakra ET, retrieve a dispatchable Chakra node
- Then, **issue** the Chakra node to the system layer

```
1. getNextIssuableNode();
```

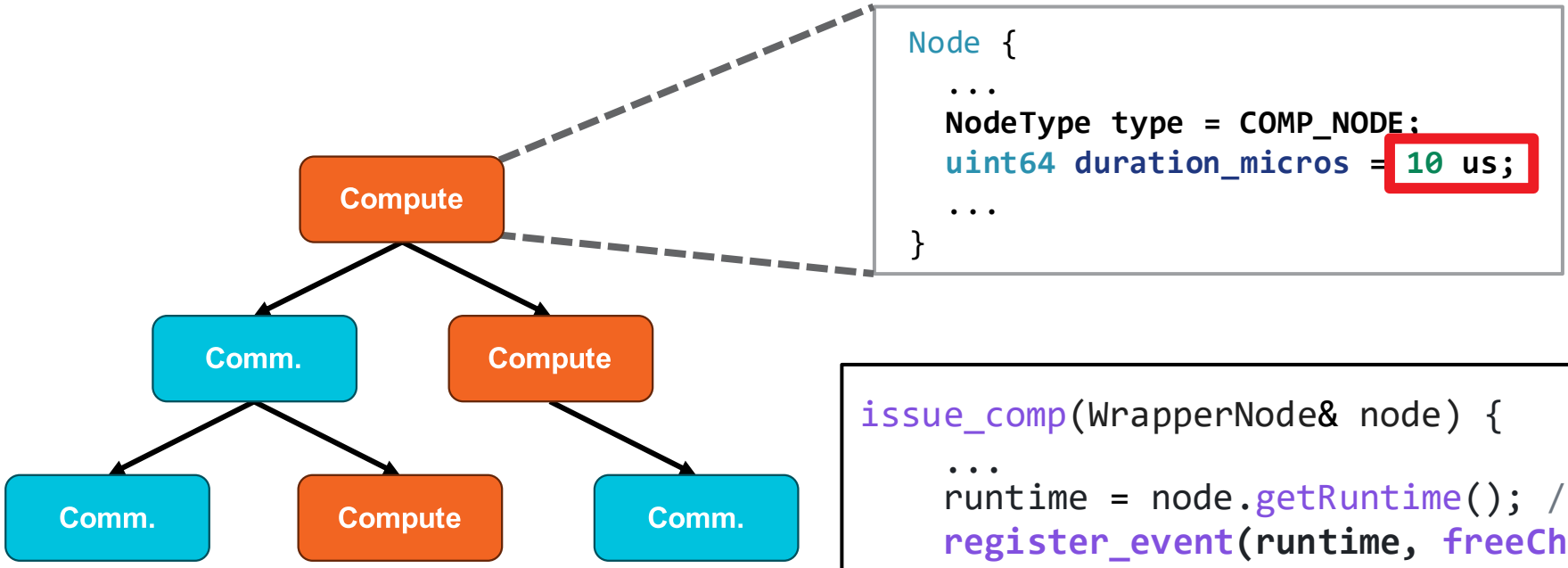



```
2. issue_comp();
```

Delegates compute modeling to the system layer

System Layer: Simple Compute Model

- Parse compute kernel information
  - Either (1) **replay** compute duration or (2) use **roofline** model to estimate compute time
- Retire compute node after this duration



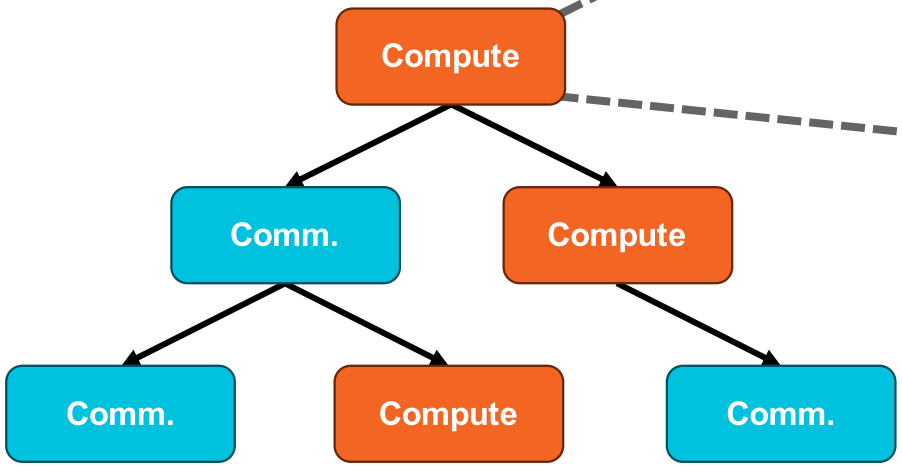

```
Node {
  ...
  NodeType type = COMP_NODE;
  uint64 duration_micros = 10 us;
  ...
}
```

```
issue_comp(WrapperNode& node) {
    ...
    runtime = node.getRuntime(); // 10 us
    register_event(runtime, freeChildrenNodes(node.id));
}
```

System layer: simple compute model

System Layer: Simple Compute Model

- Parse compute kernel information
  - Either (1) **replay** compute duration or (2) use **roofline** model to estimate compute time
- Retire compute node after this duration



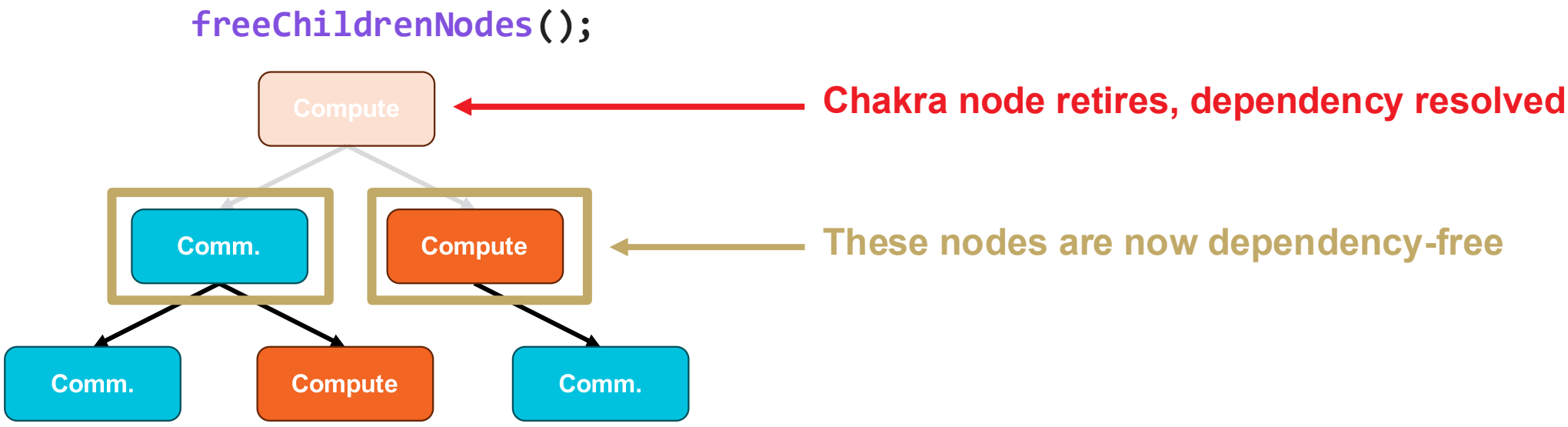

```
Node {
  NodeType type = COMP_NODE;
  uint64 attr.M = 1024;
  uint64 attr.N = 512;
  uint64 attr.K = 2048;
  string attr.data_type = "fp16";
}
```

```
issue_comp(WrapperNode& node) {
    ...
    runtime = roofline(1024, 512, 2048, "fp16");
    register_event(runtime, freeChildrenNodes(node.id));
}
```

System layer: simple compute model (roofline)

Workload Layer: Freeing a Node

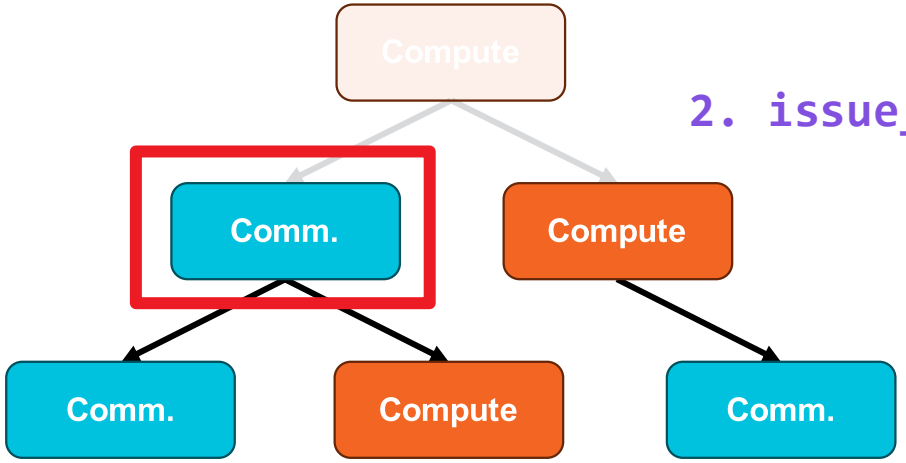
- **Retire** the Chakra node
- Mark the **dependency as resolved**



Workload Layer: Issuing Next Node

- Query and issue a **next issuable node**
  - Among the dependency-free nodes, one is randomly chosen

```
1. getNextIssuableNode();
```

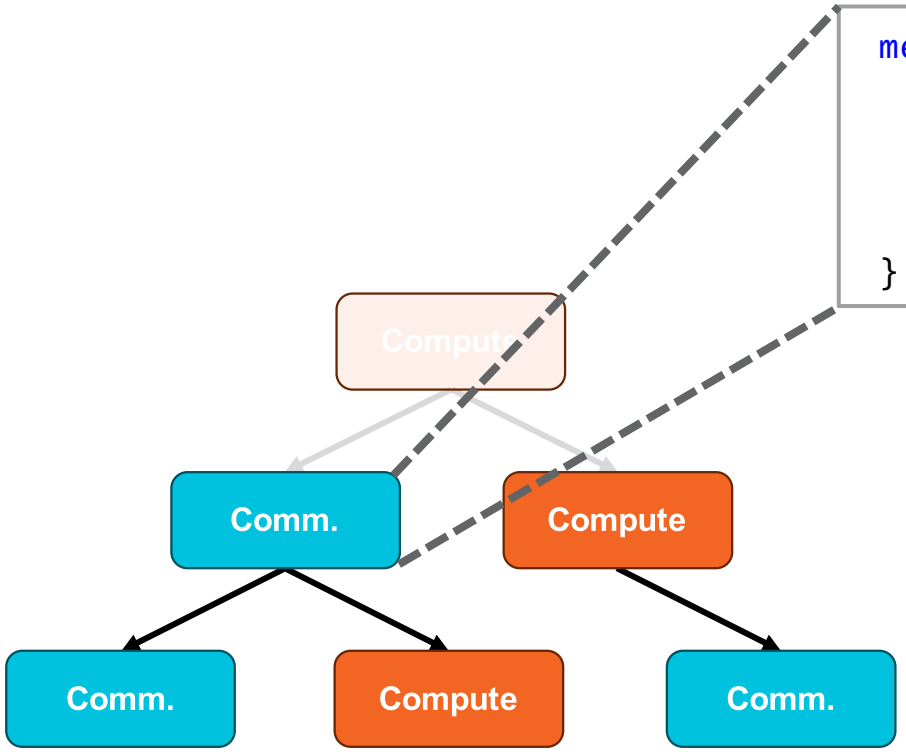



```
2. issue_comm();
```

Delegates communication modeling to the system layer

System Layer: Single Send/Receive

- Request the network backend to simulate a single network transfer




```
message Node {
  NodeType type = COMM_SEND_NODE
  uint32 attr.comm_src = GPU 2;
  uint32 attr.comm_dst = GPU 5;
  uint64 comm_size = 16 MiB;
}
```

```
issue_comm(WrapperNode& node) {
    ...
    sim_send(from: 2, to: 5, size: 16 MiB,
             callback: freeChildrenNodes(node.id));
}
```

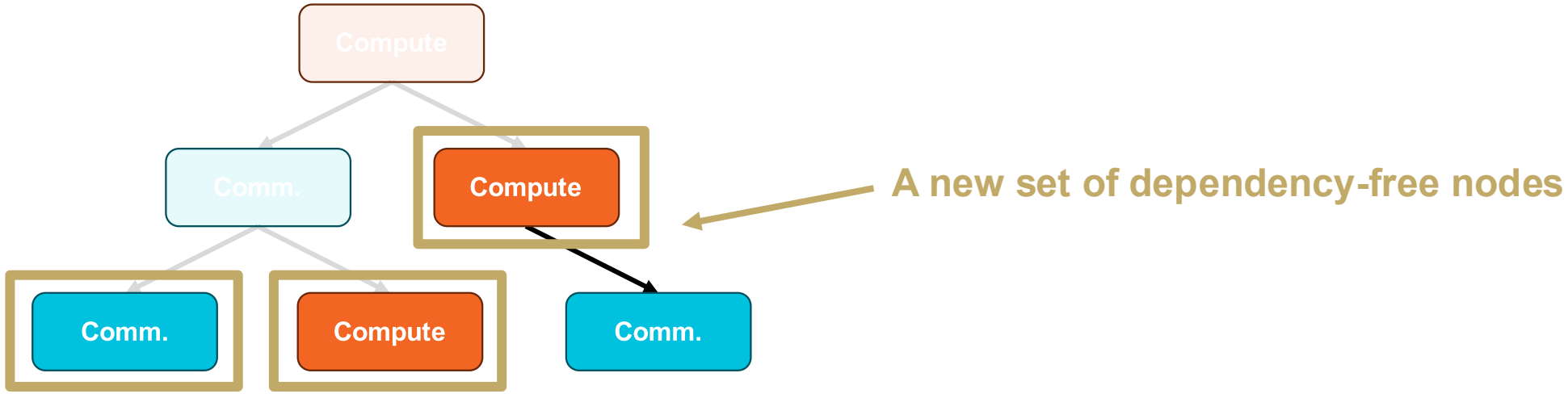
System layer: simple communication model

Further delegates actual network simulation to the network layer (later talk)

Workload Layer: Freeing a Node

- **Retire** the Chakra node and mark **dependency as resolved**

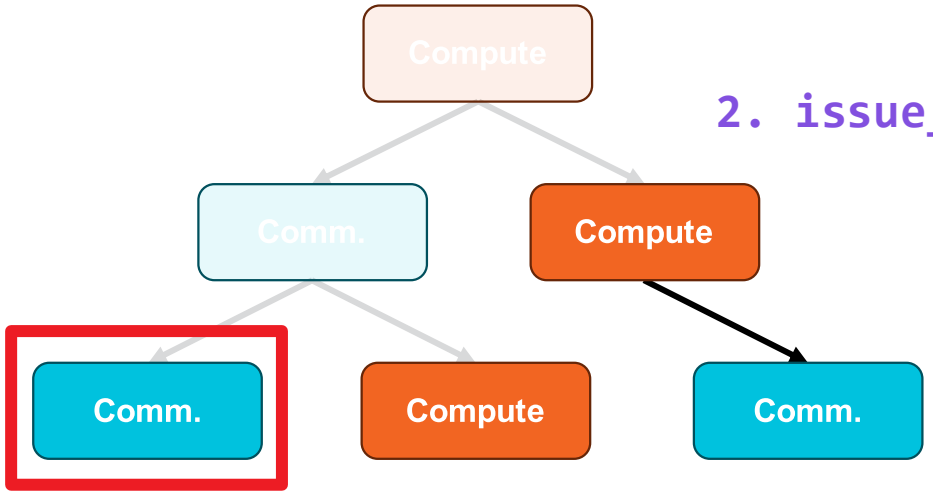
```
freeChildrenNodes();
```



Workload Layer: Freeing a Node

- Repeat the process: issue a new dependency-free Chakra node

```
1. getNextIssuableNode();
```

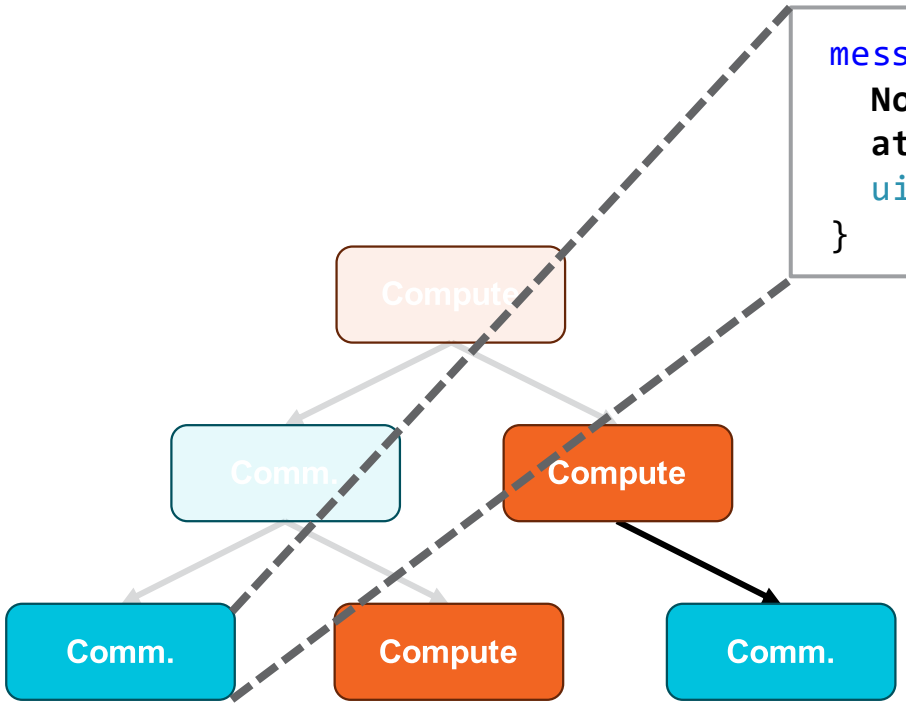



```
2. issue_comm();
```

Delegates communication modeling to the system layer

System Layer: Collective Communication

- Based on collective type and algorithm, call the appropriate **finite state machine (FSM)**



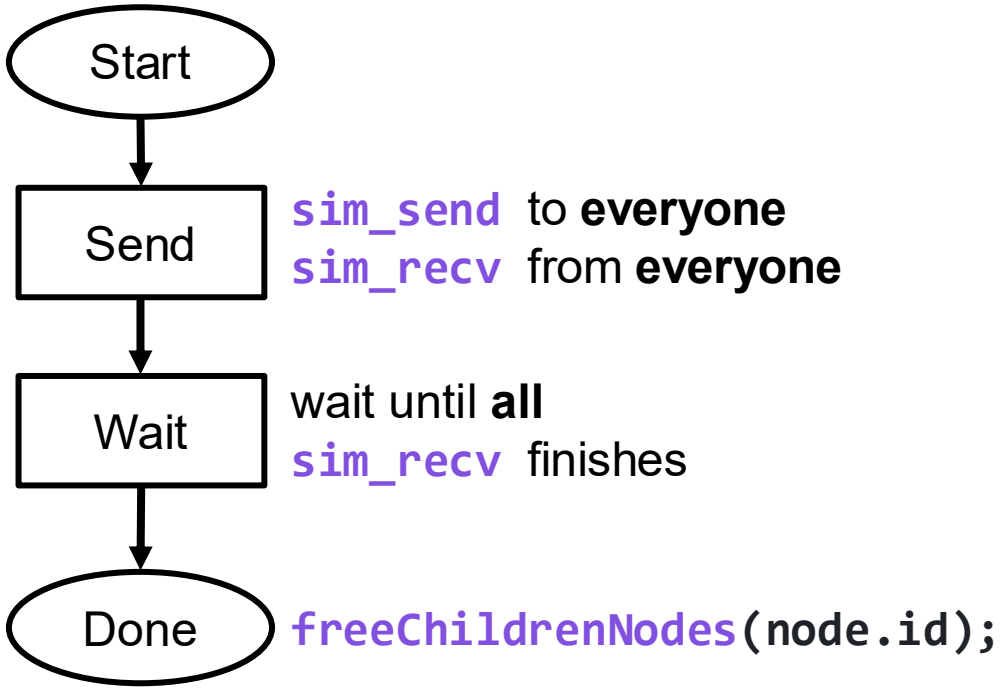

```
message Node {
  NodeType type = COMM_COLL_NODE;
  attr.comm_type = ALL_GATHER;
  uint64 comm_size = 16 MiB;
}
```

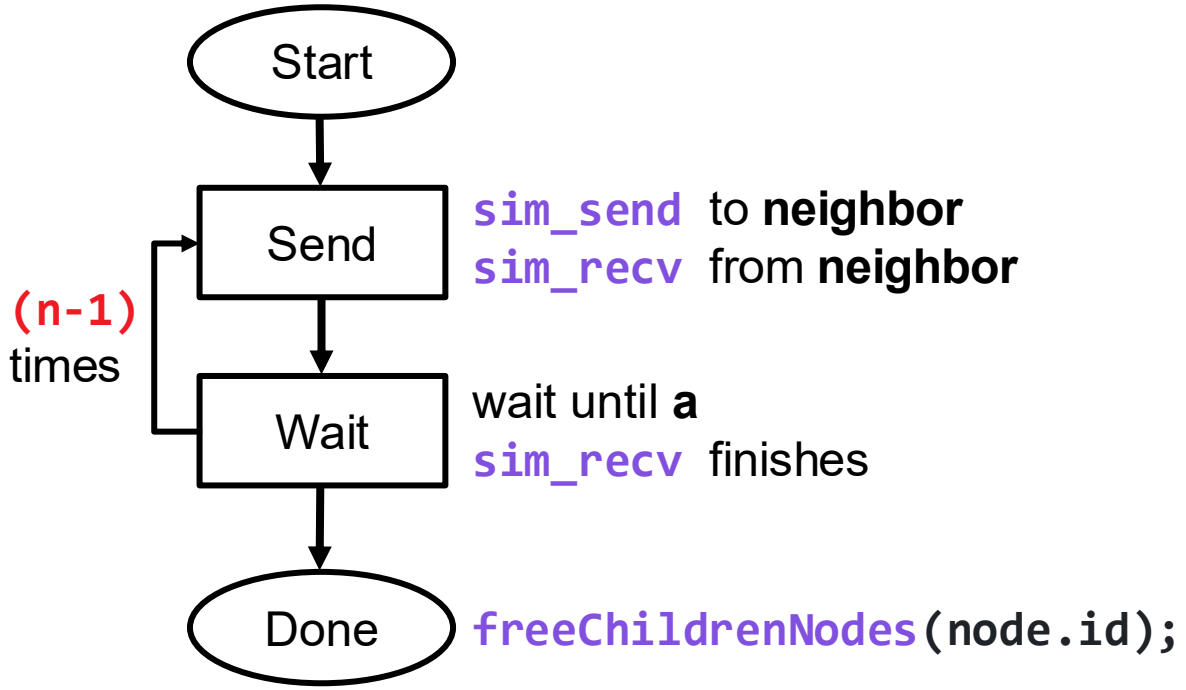
```
issue_comm(WrapperNode& node) {
    ...
    generate_all_gather(size: 16 MiB,
                        algorithm: "all-pairs");
}
```

System layer: simple communication model

System Layer: Collective Communication (cont'd)

- Each collective algorithm is an FSM
  - **Dispatches** simulation request to the network layer
  - Finally, **free the Chakra node** when FSM finishes



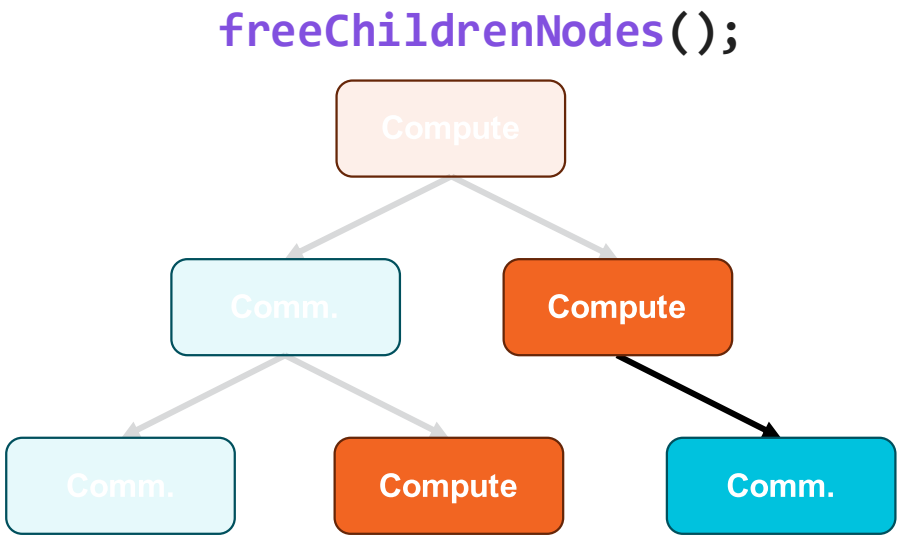



```
generate_all_gather(algorithm: "all-pairs");
```

```
generate_all_gather(algorithm: "ring");
```

Workload Layer: Freeing a Node

- ASTRA-sim simulation runs until all Chakra nodes are executed
  - When `hasNodesToIssue()` returns `false`, simulation ends.



Summary: Workload and System Layers

```
while (hasNodesToIssue()) {
    node = getNextIssuableNode();
    if (node.getNodeType() == COMP_NODE) {
        issue_comp(node);
    } else if (node.getNodeType() == COMM_NODE) {
        issue_comm(node);
    }
}
```

Workload layer

```
issue_comp(WrapperNode& node) {
    ...
    runtime = node.getRuntime();
    register_event(runtime, freeChildrenNodes(node.id));
}
```

System layer: simple compute model

```
issue_comm(WrapperNode& node) {
    ...
    generate_all_gather(size: 16 MiB, alg: "all-pairs");
}

generate_all_gather() {
    ...
    // finite state machine dispatching sim_send/recv
    freeChildrenNodes(node.id));
}
```

System layer: simple communication model

COPYRIGHT AND DISCLAIMER

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED "AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.


AMD