The background features large, overlapping geometric shapes in shades of green and teal. A large green shape starts from the top left and extends towards the center. A teal shape is positioned below it, overlapping the green one. Another green shape is at the bottom left, overlapping the teal one. A small teal triangle is located at the end of the title text.

UNIFIED AND FLEXIBLE
METHODOLOGY FOR SYSTEM
SPECIFICATION AND EVALUATION

ZHIFENG LIN, YASUKO ECKERT, GABRIEL H. LOH
AMD RESEARCH
AUGUST 10, 2016

SOC DESIGN AND OPTIMIZATION IS COMPLEX

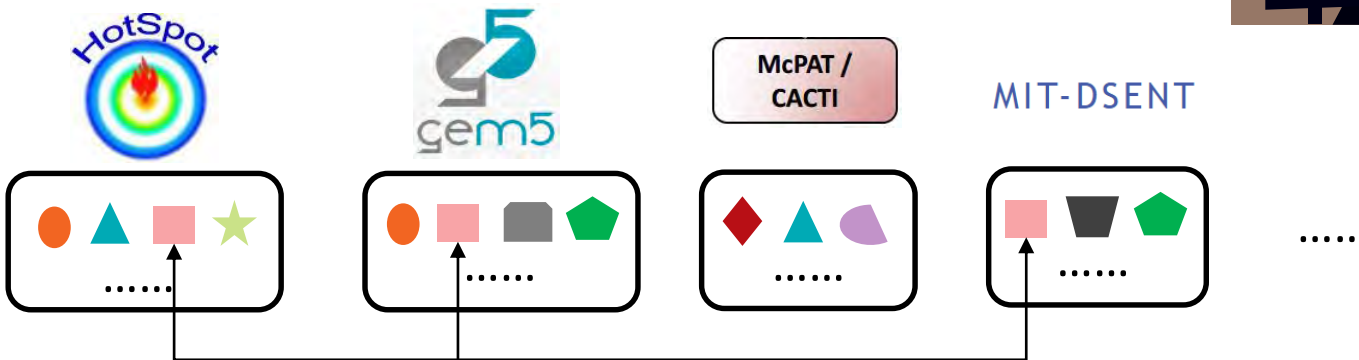


INTRODUCTION

Explore design space for complex SoC using multiple simulators

- Floor planning for components
- Yield/cost analysis
- Thermal projection and evaluation
- Power modeling
- On-chip interconnect simulation
- Etc.





Hundreds of input parameters for each of these simulators!!!

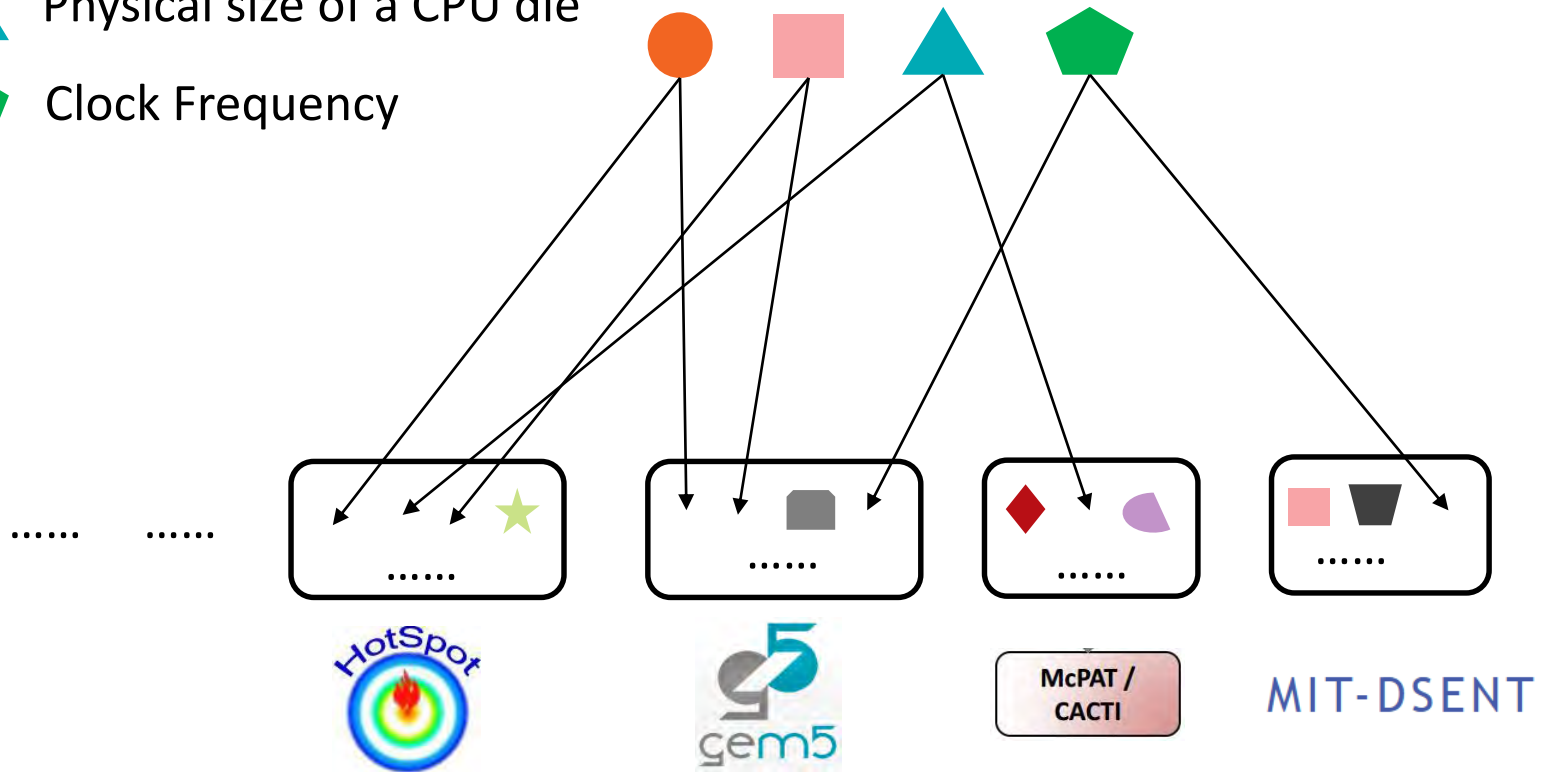


Same input to multiple different simulators

SPECIFY ONCE, USE EVERYWHERE



-  # of CPU cores
-  # of GPU compute units
-  Physical size of a CPU die
-  Clock Frequency



▲ **Consistency – Specify Once, Use Everywhere**

- Provide a method that can drive multiple simulators with consistent common input parameters

▲ **Efficient and predictable design processes**

- Spot constraints early by having common input parameters shared in “locked step”
- Avoid cross validation of configurations among different simulations to the extent possible

- ▲ Built a framework with different modules to drive their own simulations
- ▲ All common information is delivered as meta-data to the various simulation modules
- ▲ Provide tools to help auto-generate configuration to feed a simulator

DESIGN METHODOLOGY

“Specify Once, Use Everywhere”

A series of overlapping, semi-transparent green geometric shapes at the bottom of the slide, including a large trapezoid and a smaller parallelogram.

A HETEROGENEOUS SOC WITH GPU AND CPU



WORKING EXAMPLE:

SoC with two interposers on a MCM

Each with a CPU and two 3D die-stacked GPU and memory

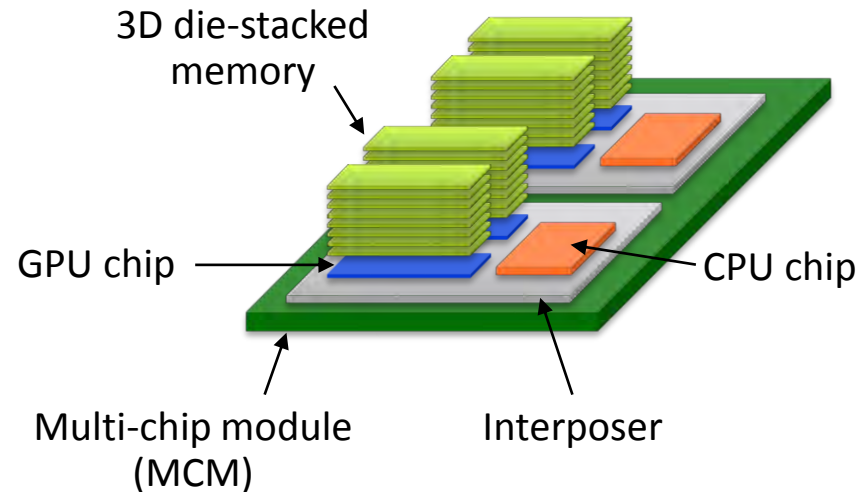
Evaluate design trade-offs through a variety of simulations

Thermal Simulation

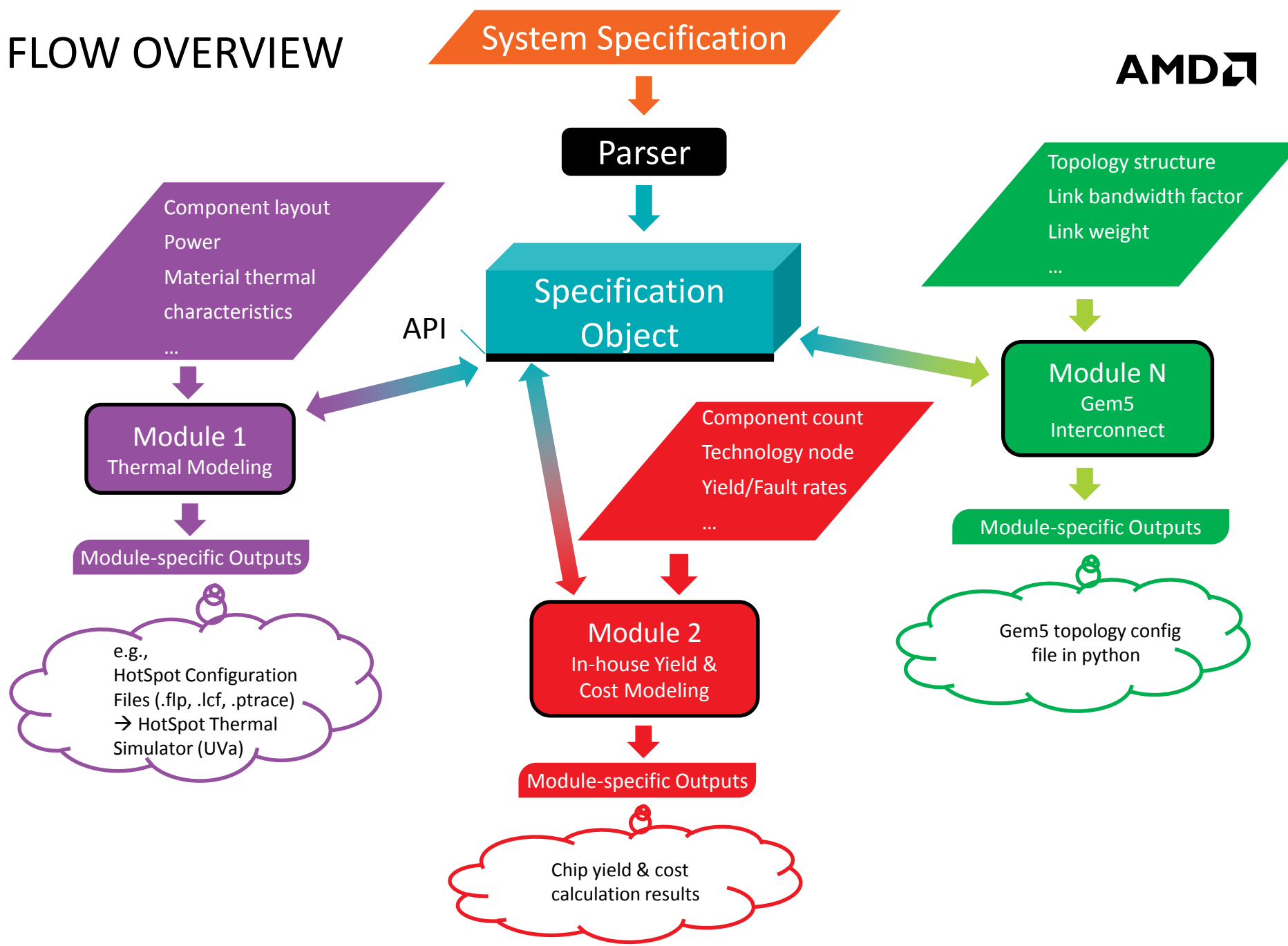
Yield/Cost Analysis

On-Chip Network Topology

Power Projection



FLOW OVERVIEW



SYSTEM DESCRIPTION

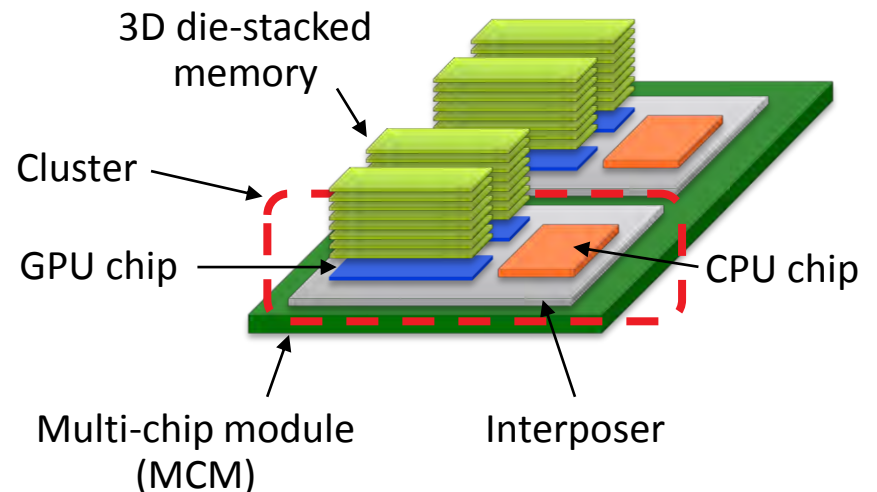


Use an xml file to describe the system with hierarchical structure

- Easy to read
- Easy to create

Example system configuration file

```
<system>
<components>
  <component componentName="MCM" count="1">
    <component componentName="Cluster" count="2">
      <items>
        <item itemName="Interposer" count="1" FloorPlan="interposer.flp" LayerNo="0" elements="interposer"/>
        <item itemName="CpuChip" count="1" FloorPlan="Cpu8Cores.flp" LayerNo="1" elements="BPred,Dec,FP,Sched,Exe,L1D,L1I,L2,L3"/>
        <item itemName="GpuChip" count="2" FloorPlan="Gpu8CU.flp" LayerNo="1" elements="CU,TCC,TSV"/>
        <item itemName="HBM" count="2" FloorPlan="HBM.flp" LayerNo="2,3,4,5,6,7,8,9" elements="HBM"/>
      </items>
    </component>
  </component>
</components>
</system>
```



“SPECIFY ONCE, USE EVERYWHERE”

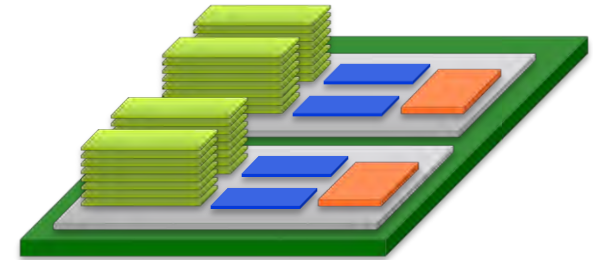


- ▲ For example, most of the component-level resources are reusable
- ▲ Can be used across SoCs with different system-level organizations
 - Fast design space exploration!

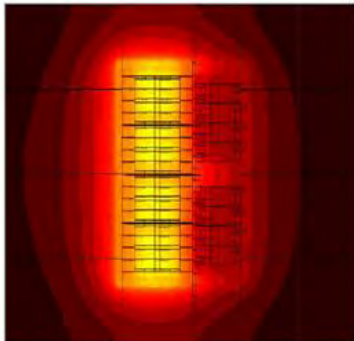


Memory-on-logic 3D stacking?	Yes	Yes	No
# of interposers	2	1	2
MCM	Yes	No	Yes

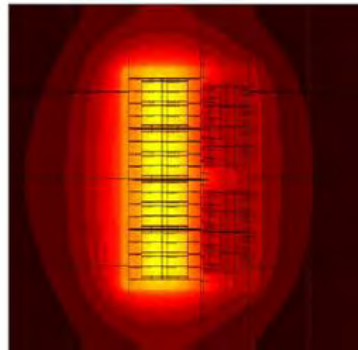
FAST DESIGN EXPLORATION



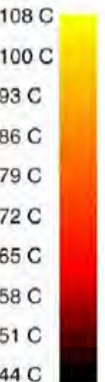
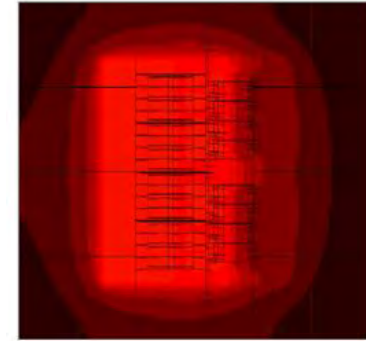
```
<system>
<components>
<component componentName="MCM" count="1">
<component componentName="Cluster" count="2">
<items>
<item itemName="Interposer" count="1" FloorPlan="interposer.flp"
LayerNo="0" elements="interposer"/>
<item itemName="GpuChip" count="2" FloorPlan="Gpu8CU.flp"
LayerNo="1" elements="CU,TCC,TSV"/>
<item itemName="CpuChip" count="1" FloorPlan="Cpu8Cores.flp"
LayerNo="1" elements="BPred,Dec,FP,Sched,Exe,L1D,L1,L2,L3"/>
<item itemName="HBM" count="2" FloorPlan="HBM.flp"
LayerNo="2,3,4,5,6,7,8,9" elements="HBM"/>
</items>
</component>
</components>
</system>
```



```
<system>
<components>
<component componentName="Cluster" count="1">
<items>
<item itemName="Interposer" count="1"
FloorPlan="interposerB.flp" LayerNo="0"
elements="interposer"/>
<item itemName="GpuChip" count="4" FloorPlan="Gpu8CU.flp"
LayerNo="1" elements="CU,TCC,TSV"/>
<item itemName="CpuChip" count="2" FloorPlan="Cpu8Cores.flp"
LayerNo="1" elements="BPred,Dec,FP,Sched,Exe,L1D,L1,L2,L3"/>
<item itemName="HBM" count="4" FloorPlan="HBM.flp"
LayerNo="2,3,4,5,6,7,8,9" elements="HBM"/>
</items>
</component>
</components>
</system>
```



```
<system>
<components>
<component componentName="MCM" count="1">
<component componentName="Cluster" count="2">
<items>
<item itemName="Interposer" count="1"
FloorPlan="interposerC.flp" LayerNo="0"
elements="interposer"/>
<item itemName="GpuChip" count="2" FloorPlan="Gpu8CU.flp" LayerNo="1"
elements="CU,TCC,TSV"/>
<item itemName="CpuChip" count="1" FloorPlan="Cpu8Cores.flp" LayerNo="1"
elements="BPred,Dec,FP,Sched,Exe,L1D,L1,L2,L3"/>
<item itemName="HBM" count="2" FloorPlan="HBM.desc"
LayerNo="1,2,3,4,5,6,7,8" elements="HBM"/>
</items>
</component>
</components>
</system>
```



SUMMARY



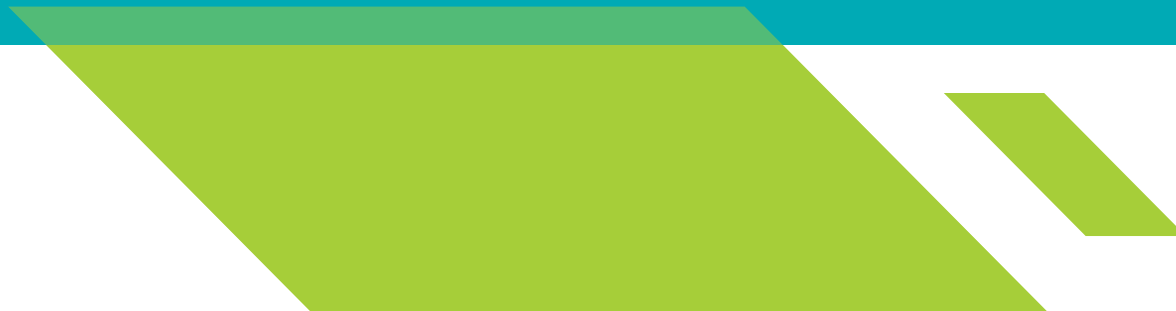
SPECIFY ONCE USE EVERYWHERE



▲ Our framework enables:

- Evaluating different aspects of system with consistent inputs across simulators
- Avoiding unnecessary cross validations of input configurations
- Generating configuration files much faster without worrying about careless typos
- Exploring more system design variations utilizing many reusable components

Questions? 



DISCLAIMER & ATTRIBUTION



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 for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. 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.

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 MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY 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.

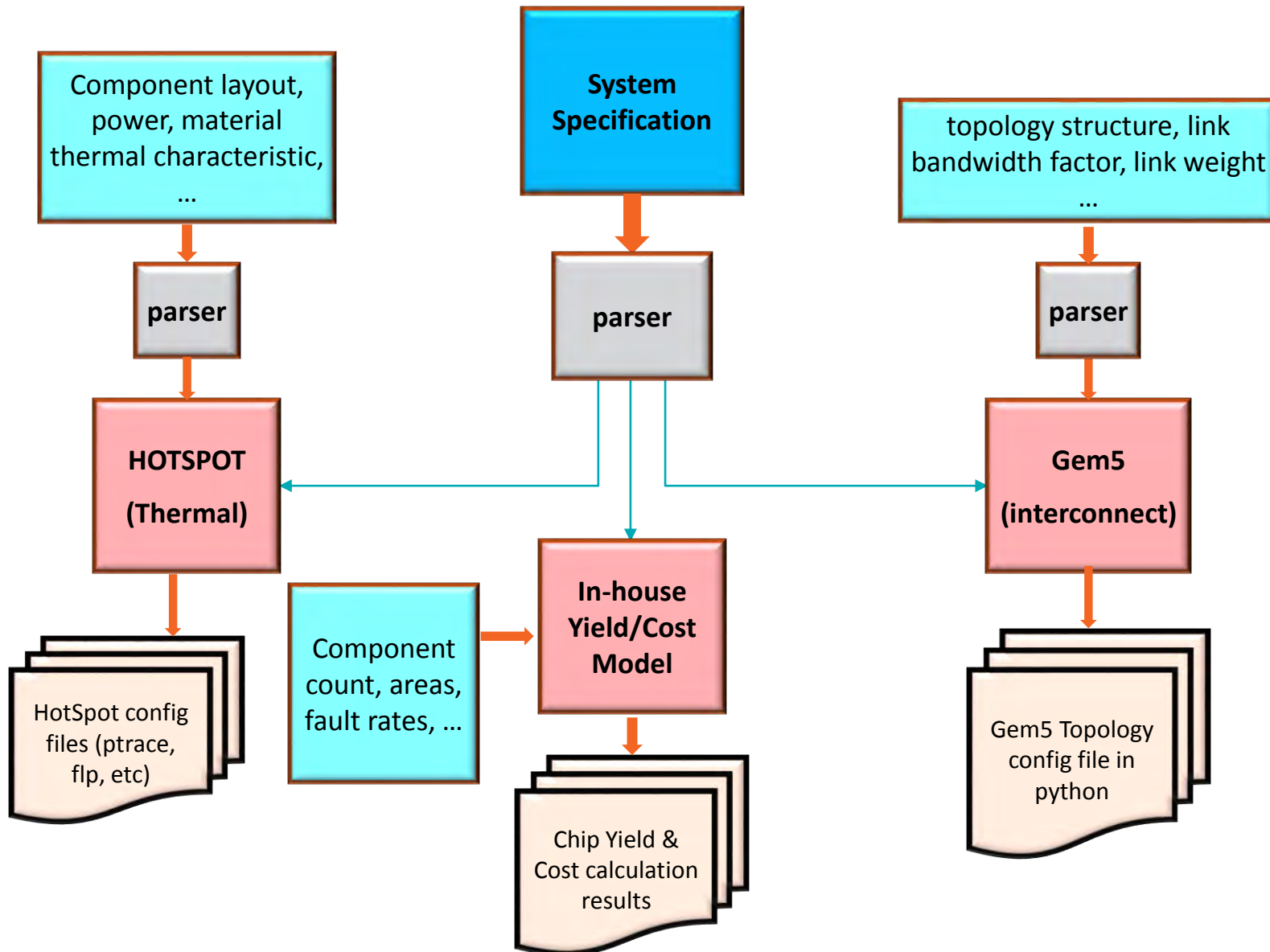
ATTRIBUTION

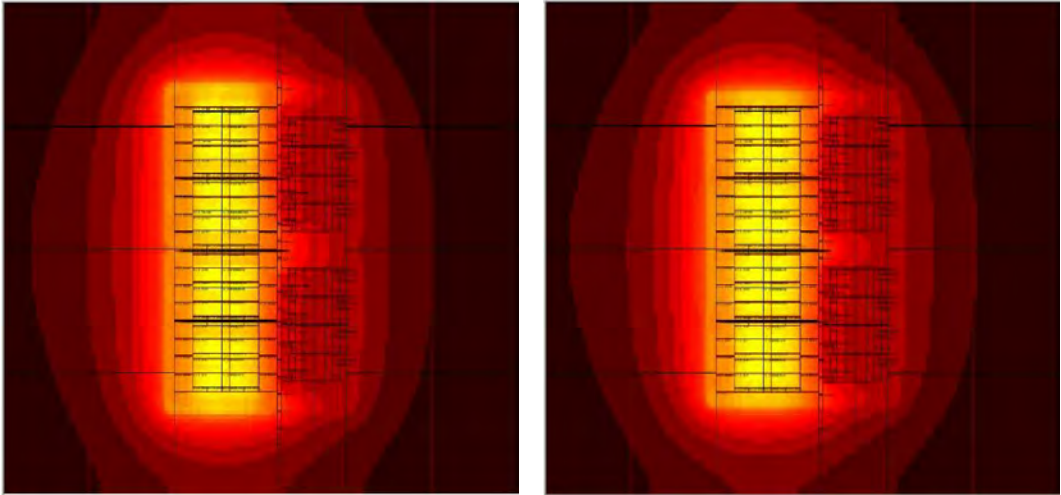
© 2016 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

BACKUP.



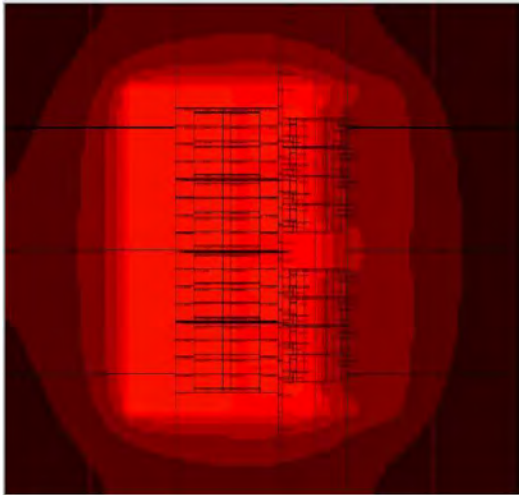
FLOW OVERVIEW





(a)

(b)



(c)



Explore logical interconnect configurations -- Auto-generate Topology(in python) for Gem5 Simulation

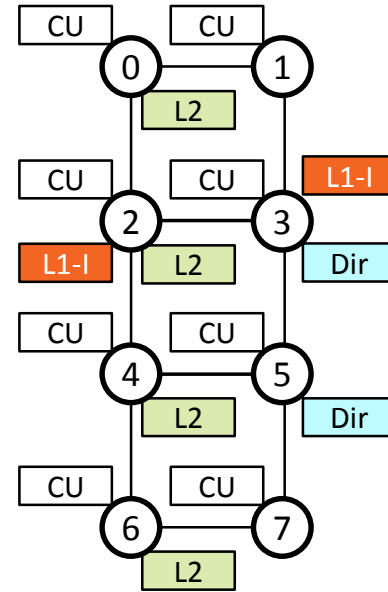


```
# gem5Topology.ini
topologyType::GpuComplex mesh

meshRows::GpuComplex      4
meshCols::GpuComplex      2

L1-IRouters                2 3
L2Routers                  0 2 4 6
DirRouters                  3 5
```

(a)



(b)

Explore logical interconnect configurations --

Auto-generate Topology(in python) for Gem5 Simulation

▲ Goals

- Make sure Gem5 Simulation uses the same set of system-level input parameters that are shared among other simulators and modeling tools
 - For example, the interconnect topology must assume 4 GpuComplex, each with 8 CUs
- Ease Gem5 users' burden of manually creating every single link (especially internal links) in a huge SoC system.
 - This can be done automatically for subcomponents (such as a GpuComplex in the example) that have “regular” topology structures (mesh, torus, ring, etc)
 - Also provide some APIs for gluing these “regular” topologies together → instead of dealing at the link-level, users can now create topology at the chip-level, or cluster-level

▲ Improvements

- Extend the framework to have more modules to drive other simulations and modeling analysis; add/modify attributes in system.xml to describe a system better
- Couple the logical design more tightly with the physical implementation of the system
 - Ex. Make sure a topology configuration is feasible on a specific layout
 - Or the other way around, make sure a layout can support a specific type of topology
- Provide a GUI instead of a text-based interface