Bridging Functional and Architectural Aspects of Controller Design in Cyber-Physical System

Liangpeng Guo

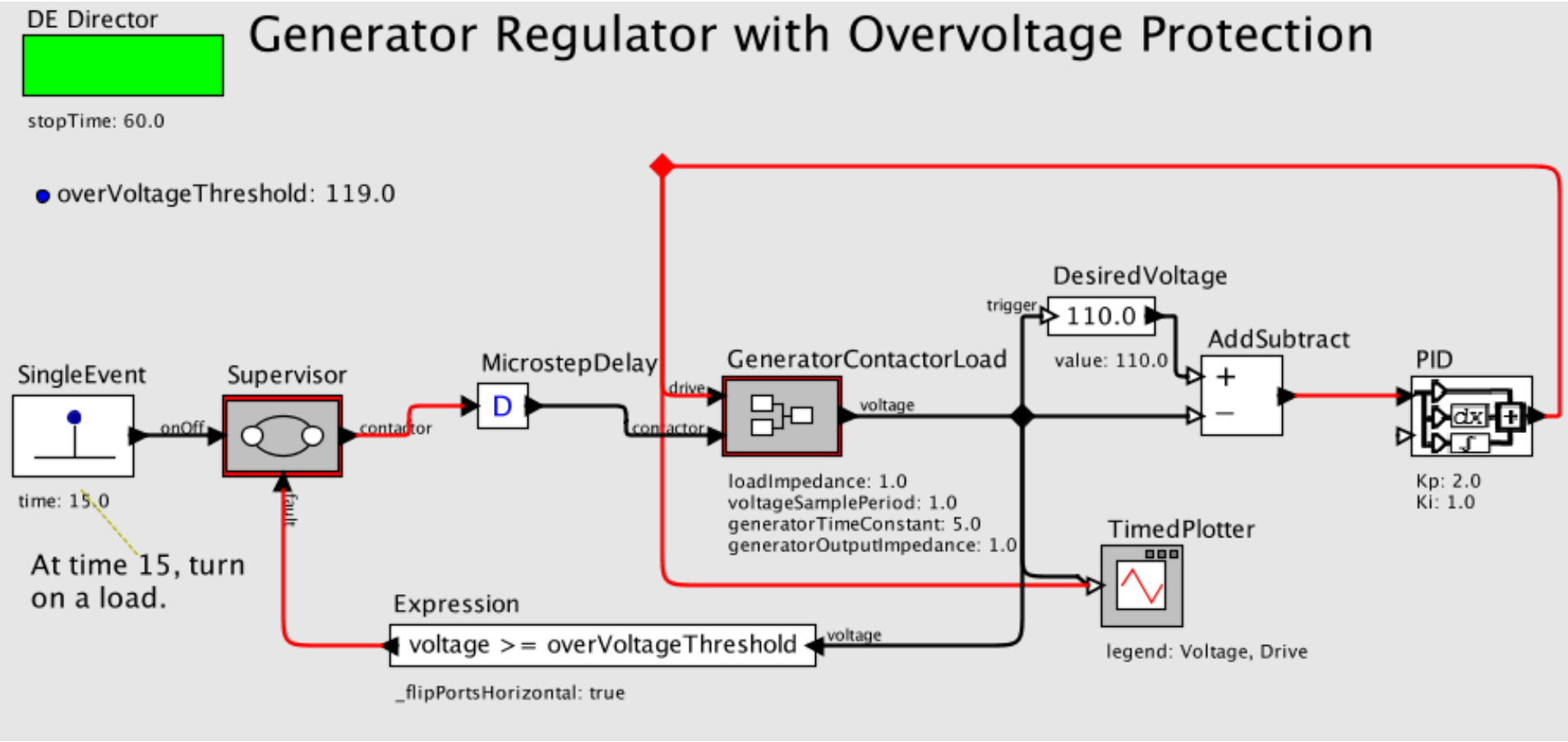
EECS Department, UC Berkeley

6/7/2013

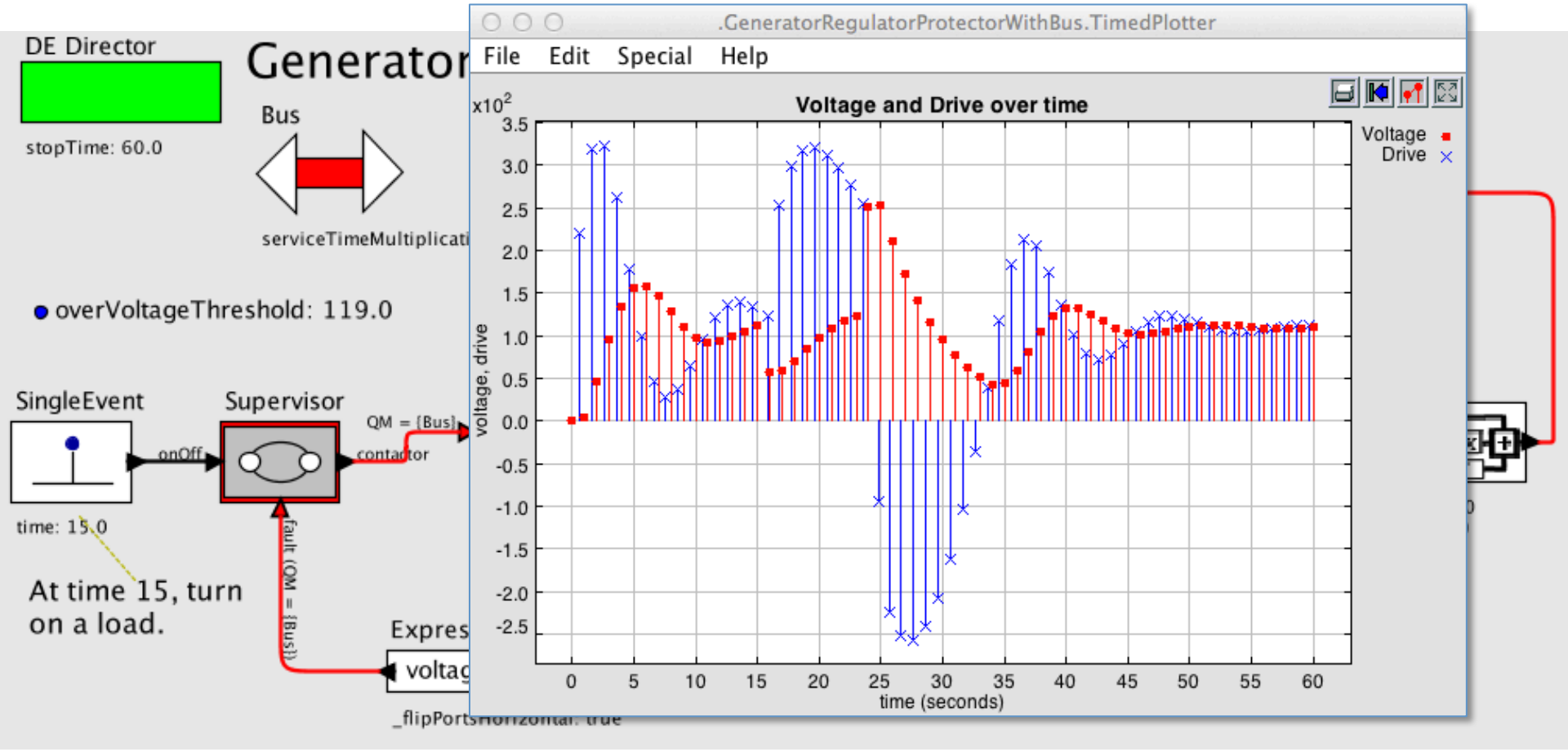
Gap between Function and Architecture

- Function: describe the functionality of the design
  - Behaviors that the system should implement
- Architecture: provides various platform targets (configurations of resources that implement certain functions)
- The functionality is modeled on high abstraction level with heterogeneous MoCs
  - High level abstraction without implementation details

A Functional Model without Implementation Details



Functional Behaviors are Affected by Implementation Details



Need New Methods & Tools for Bridging Functional & Architectural Aspects

- Approach 1: refine functional model by adding implementation details
  - E.g. add bus model
- Potential problems:
  - The model becomes complicated to understand and modify
  - Environment/language for modeling function is not suitable for architecture.
  - Semantic restriction: difficult to model complex architecture in functional environment
- Approach 2: construct the function using architectural primitives
  - E.g. programming language
- Potential problems:

Need New Methods & Tools for Bridging Functional & Architectural Aspects

- Approach 3: make assumptions (contracts) that decouple the function and architecture
  - E.g. assume a bounded communication delay
- Potential problems:
  - Making ‘appropriate’ assumptions is hard
  - Assumptions of the functional model about the architecture significantly impact the Software/ Hardware implementation
  - Inappropriate assumptions may restrict the design choices and lead to costly or infeasible architecture

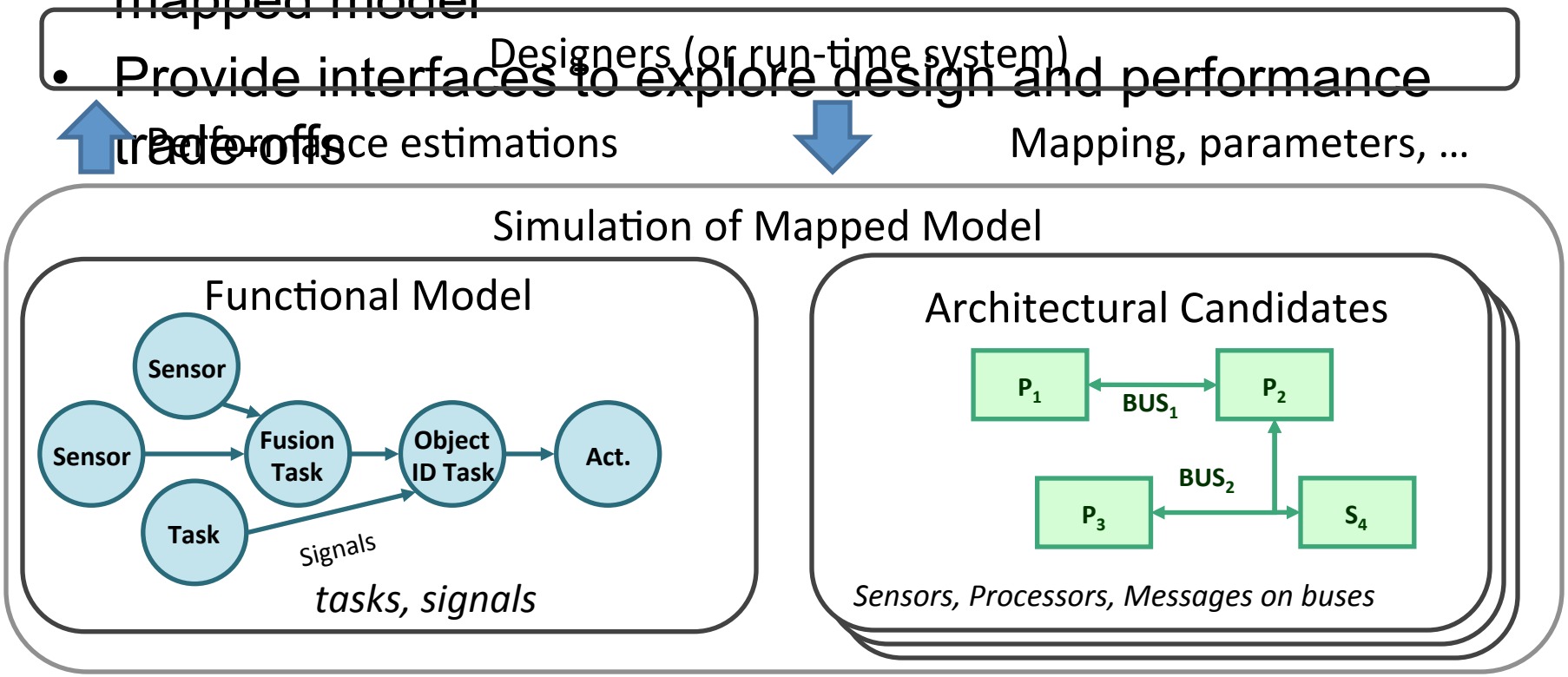
Aspect-Oriented Modeling

Joint modeling of implementation architecture and functional design to enable effective architecture exploration and assessment of the behavioral consequences of architectural choices.

- What is the performance? How this modifies the behavior? Does this satisfy the vertical contract?
- Investigate different design choices for the same functional model

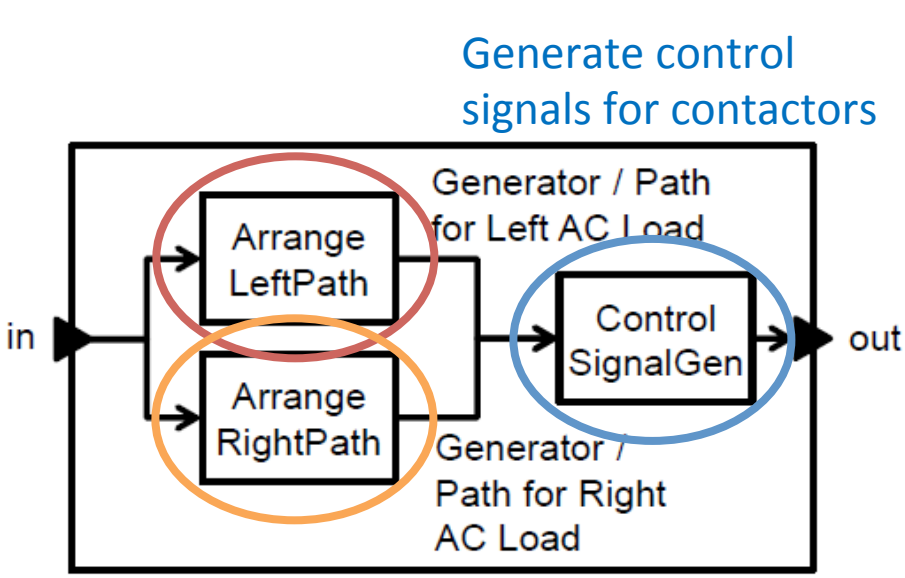
Aspect-Oriented Modeling

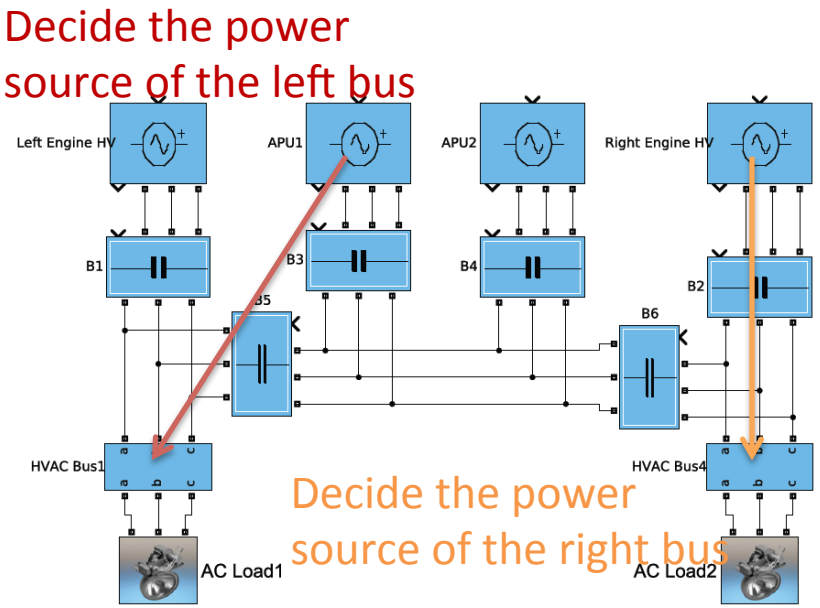
- Map functional and architectural models without significant changes
- Enable co-simulation and performance estimation of the mapped model
- Provide interfaces to explore design and performance trade-offs



Example 1: Simplified BCU Controller

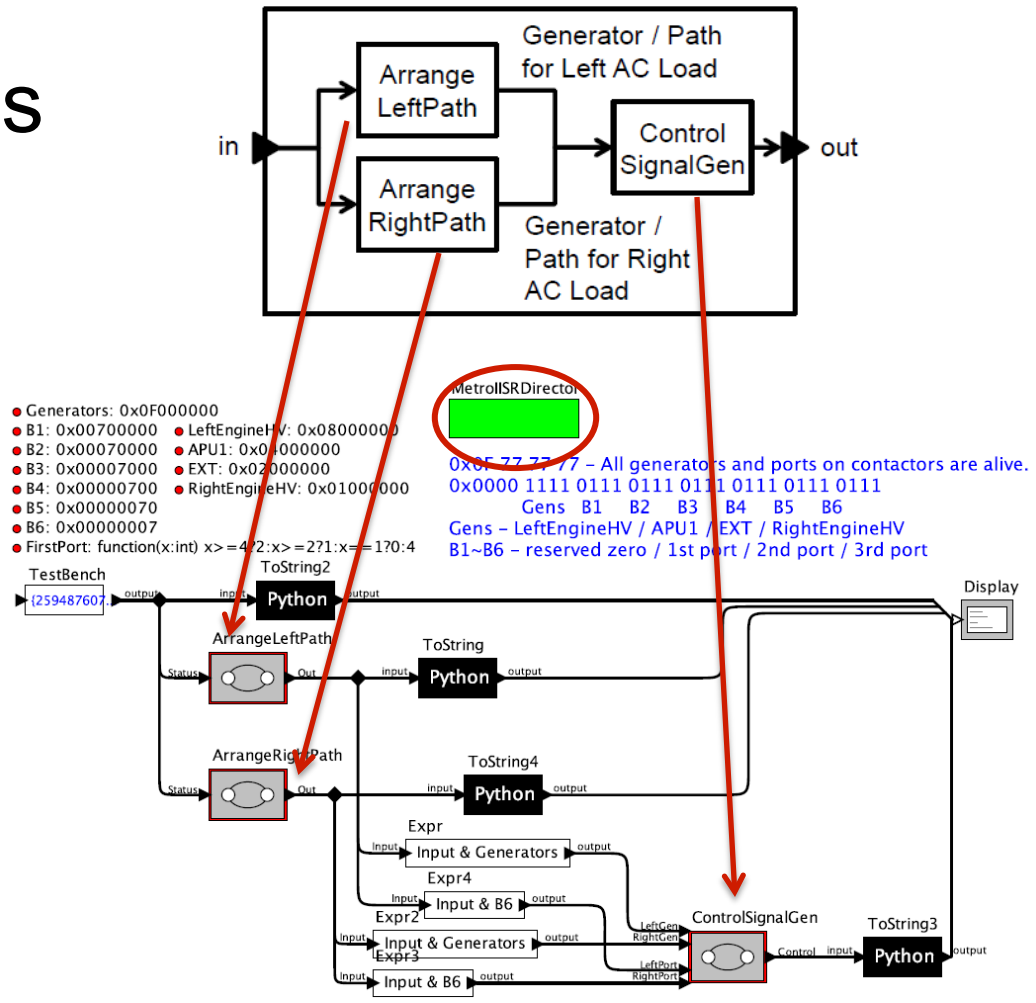
- Decide the power sources for AC buses
  - Input: fault signals of power sources and contactors
  - Output: control signals for contactors
  - Constraint: power sources are never paralleled



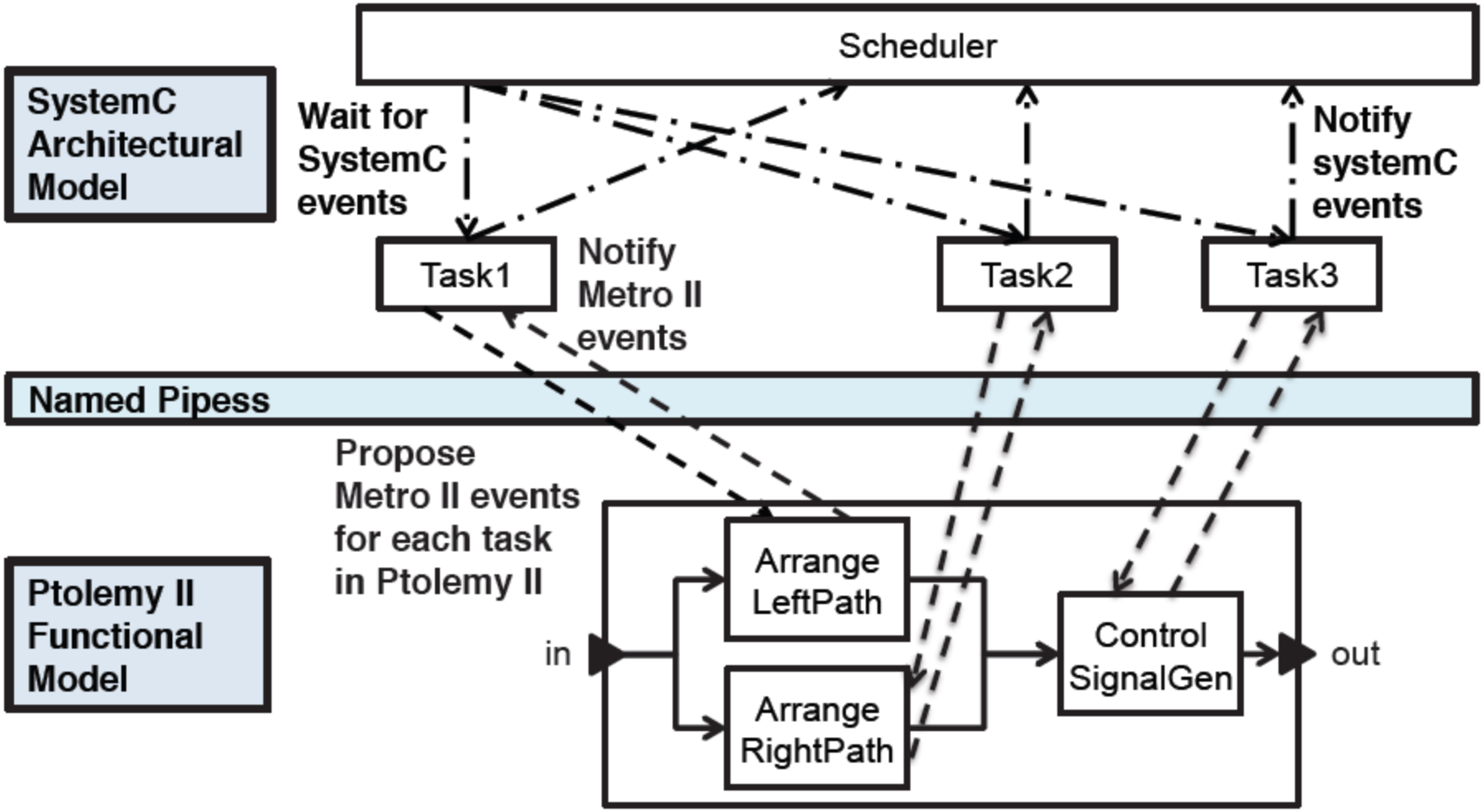


Functional Model in Ptolemy

- Model each part as a FSM
- Compose FSMs into a Synchronou Reactive (SR) Model
- SR Director is modified to make



Map to Arch Model in SystemC



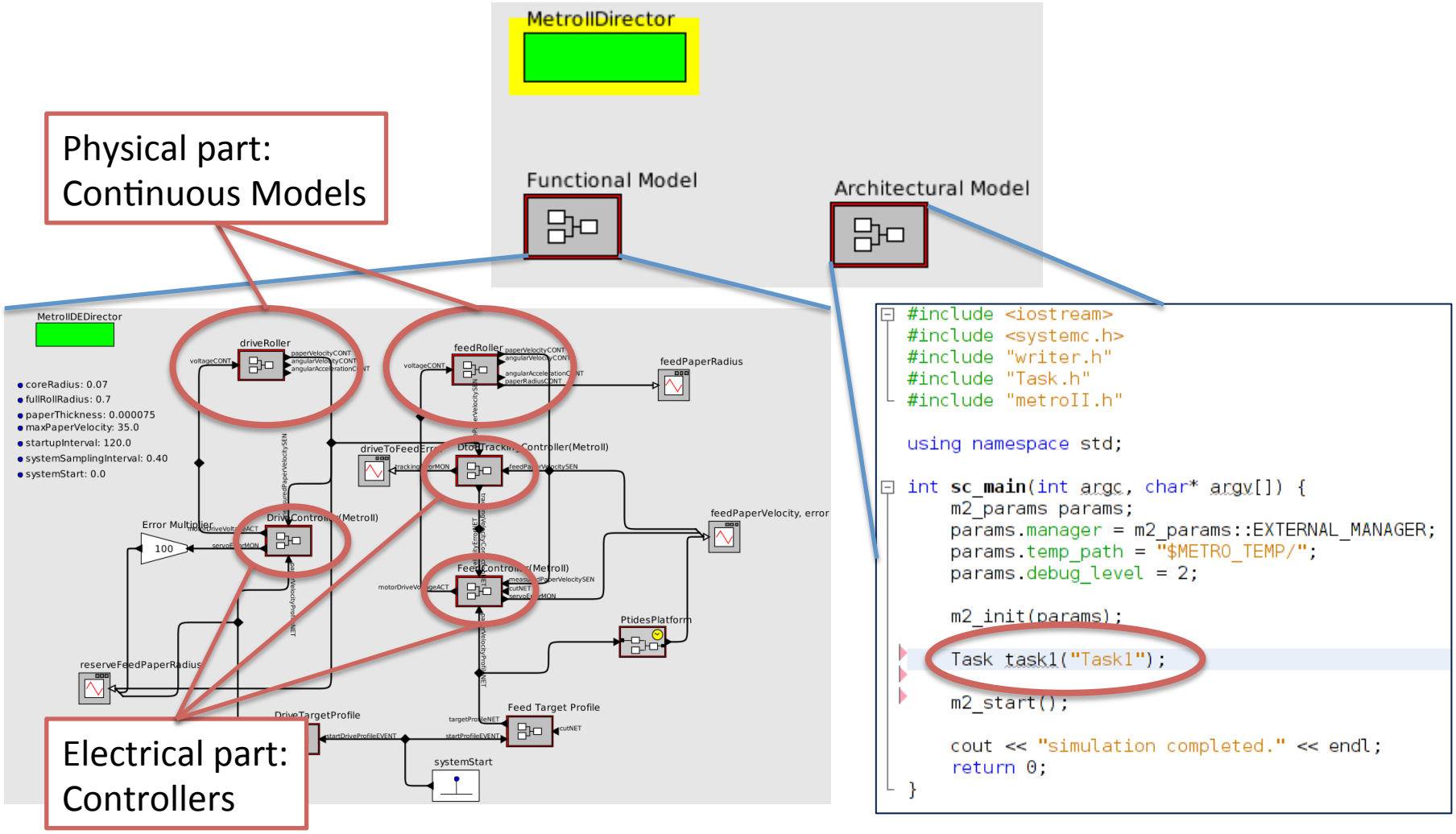
Preliminary Co-simulation Results

- Candidate 1
  - High-speed single processor
- Candidate 2
  - Slow-speed dual processors
  - low synchronization overhead
- Candidate 3
  - Medium speed dual processors
  - high synchronization

Parameters		Candidate		
		1	2	3
Scheduling overhead (ns)		10	10	10
Execution Time (ns)	ALP	40	60	50
	ARP	45	65	55
	CSG	25	35	30
Synchronization overhead (ns)		0	5	15
Parallelization of ALP and ARP		No	Yes	Yes
Total execution time (ns)		775	800	750

ALP: left controller
ARP: right controller
CSG: control signal generator

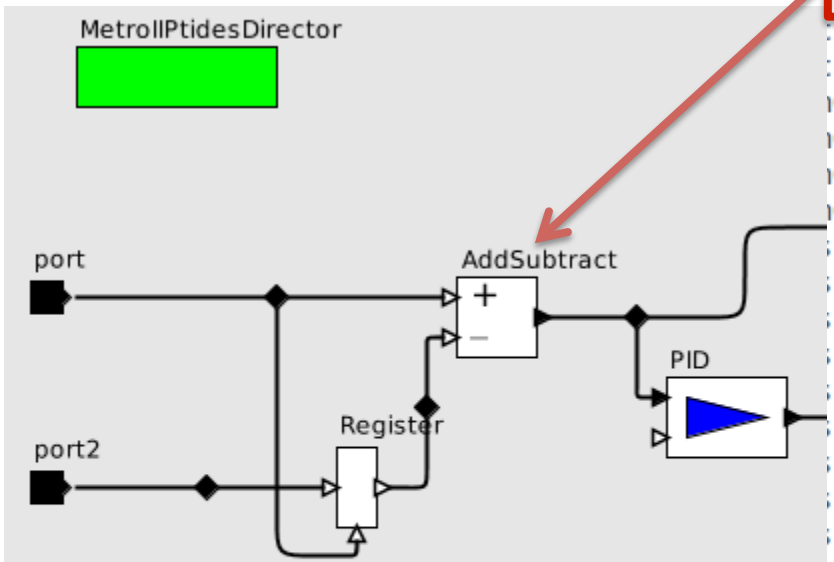
Example 2: printingpress controller




```
#include <iostream>
#include <systemc.h>
#include "writer.h"
#include "Task.h"
#include "metroII.h"

using namespace std;

int sc_main(int argc, char* argv[]) {
    m2_params params;
    params.manager = m2_params::EXTERNAL_MANAGER;
    params.temp_path = "$METRO_TEMP/";
    params.debug_level = 2;

    m2_init(params);

    Task task1("Task1");

    m2_start();

    cout << "simulation completed." << endl;
    return 0;
}
```

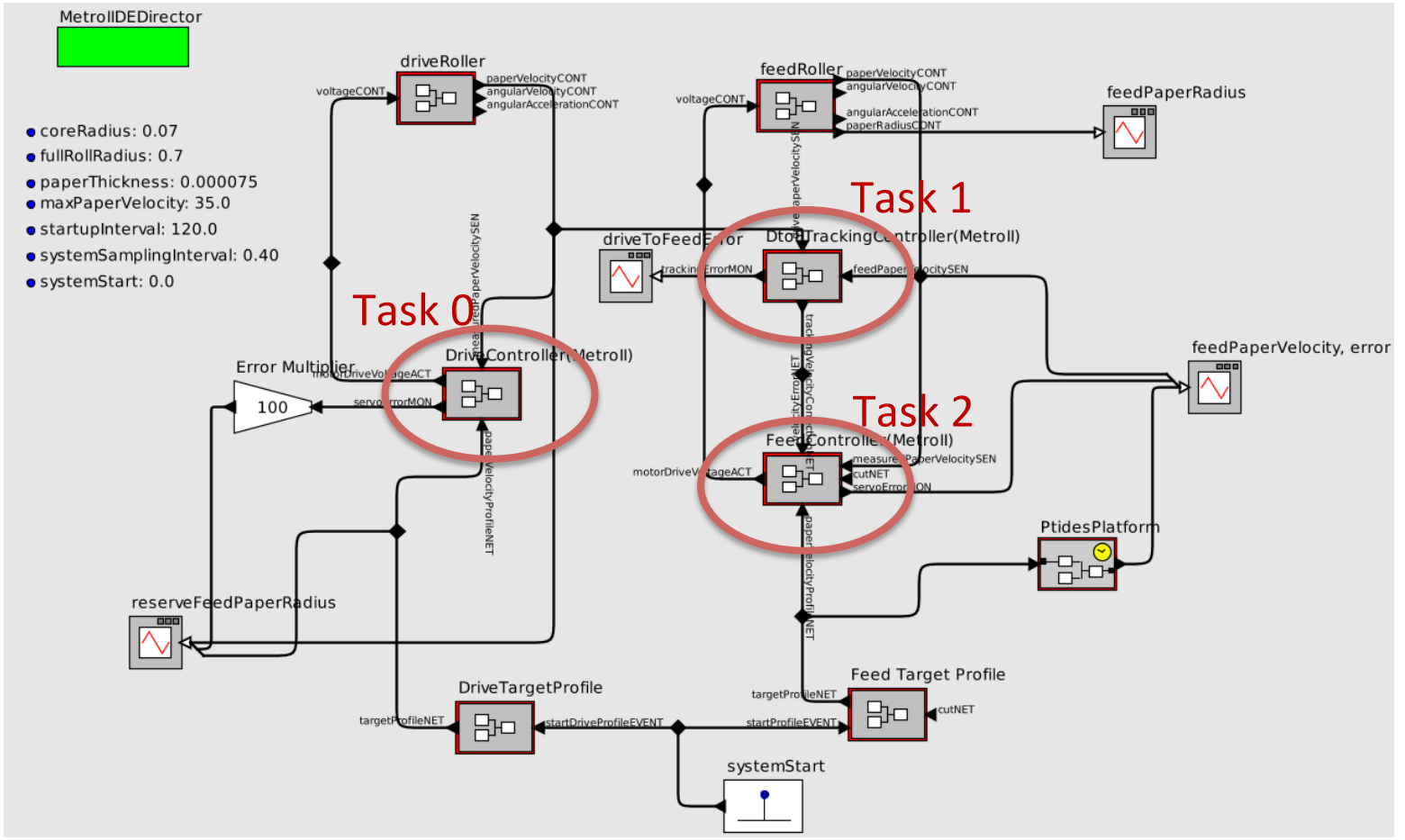
Mapping Configuration

Ptides MoC




```
s.Register.FIRE_BEGIN,Task0.Register.Begin
s.Register.FIRE_END,Task0.Register.End
s.AddSubtract.FIRE_BEGIN,Task0.AddSubtract.Begin
s.AddSubtract.FIRE_END,Task0.AddSubtract.End
s.PID.FIRE_BEGIN,Task0.PID.Begin
s.PID.FIRE_END,Task0.PID.End
Contents.AddSubtract.FIRE_BEGIN,Task1.AddSubtract.Begin
Contents.AddSubtract.FIRE_END,Task1.AddSubtract.End
Contents.PID.FIRE_BEGIN,Task1.PID.Begin
Contents.PID.FIRE_END,Task1.PID.End
.Register.FIRE_BEGIN,Task2.Register.Begin
.Register.FIRE_END,Task2.Register.End
.Register2.FIRE_BEGIN,Task2.Register.Begin
.Register2.FIRE_END,Task2.Register.End
.AddSubtract.FIRE_BEGIN,Task2.AddSubtract.Begin
.AddSubtract.FIRE_END,Task2.AddSubtract.End
.AddSubtract2.FIRE_BEGIN,Task2.AddSubtract.Begin
.AddSubtract2.FIRE_END,Task2.AddSubtract.End
.Const.FIRE_BEGIN,Task2.Const.Begin
.Const.FIRE_END,Task2.Const.End
.Const2.FIRE_BEGIN,Task2.Const.Begin
.Const2.FIRE_END,Task2.Const.End
.PID.FIRE_BEGIN,Task2.PID.Begin
.PID.FIRE_END,Task2.PID.End
```

Mapping Configuration



Round-Robin Scheduling

- Architectural model
  - Designed in a language and environment that are comfortable for architecture designer
- Example 1: RR scheduling

```
Task* OS::next_task_to_run() {
    if (_ready_task_list.empty()) {
        return NULL;
    }
    Task* next_task = _ready_task_list.front();
    _ready_task_list.pop_front();
    return next_task;
}
```

```
   500 ms: Task0.Register begins
500005 us: Task0.Register ends
500005 us: Task2.Register begins
500010 us: Task2.Register ends
500010 us: Task0.AddSubtract begins
500012 us: Task0.AddSubtract ends
500012 us: Task2.AddSubtract begins
500014 us: Task2.AddSubtract ends
500014 us: Task0.PID begins
500029 us: Task0.PID ends
500029 us: Task2.Register begins
500034 us: Task2.Register ends
500034 us: Task2.AddSubtract begins
500036 us: Task2.AddSubtract ends
500036 us: Task2.PID begins
500051 us: Task2.PID ends
500051 us: Task2.Const begins
500053 us: Task2.Const ends
```

```
   800 ms: Task1.AddSubtract begins
800002 us: Task1.AddSubtract ends
800002 us: Task2.Register begins
800007 us: Task2.Register ends
800007 us: Task0.Register begins
800012 us: Task0.Register ends
800012 us: Task1.PID begins
800027 us: Task1.PID ends
800100 us: Task2.Register begins
800105 us: Task2.Register ends
```

Priority-based Scheduling

- Example 1: priority-bas

```
Task* OS::next_task_to_run() {
    if (_ready_task_list.empty()) {
        return NULL;
    }
    // Task* next_task = _ready_task_list.front();
    // _ready_task_list.pop_front();

    list<Task *>::iterator it_highest_priority = _re
    for (list<Task *>::iterator it=_ready_task_list                    it++) {
        if ((*it)->priority() > (*it_highest_priorit
            it_highest_priority = it;
        }
    }
    Task * next_task = (*it_highest_priority);
    _ready_task_list.erase(it_highest_priority);
    return next_task;
}
```

```
   500 ms: Task0.Register begins
500005 us: Task0.Register ends
500005 us: Task0.AddSubtract begins
500007 us: Task0.AddSubtract ends
500007 us: Task0.PID begins
500022 us: Task0.PID ends
500022 us: Task2.Register begins
500027 us: Task2.Register ends
500027 us: Task2.AddSubtract begins
500029 us: Task2.AddSubtract ends
500029 us: Task2.Register begins
500034 us: Task2.Register ends
500034 us: Task2.AddSubtract begins
500036 us: Task2.AddSubtract ends
500036 us: Task2.PID begins
500051 us: Task2.PID ends
500051 us: Task2.Const begins
500053 us: Task2.Const ends
```

```
   800 ms: Task0.Register begins
800005 us: Task0.Register ends
800005 us: Task1.AddSubtract begins
800007 us: Task1.AddSubtract ends
800007 us: Task1.PID begins
800022 us: Task1.PID ends
800022 us: Task2.Register begins
800027 us: Task2.Register ends
800100 us: Task2.Register begins
800105 us: Task2.Register ends
```

Performance Annotation

- The basic performance annotation can be customized in a csv file

Register	5	SC_US
DiscreteClock	1	SC_US
CurrentTime	1	SC_US
Scale	5	SC_US
TrigFunction	10	SC_US
Const	2	SC_US
AddSubtract	2	SC_US
BooleanSwitch	5	SC_US
Merge	5	SC_US
LogicalNot	2	SC_US
PID	15	SC_US
Repeat	20	SC_US
Ramp	5	SC_US

```
   500 ms: Task0.Register begins
500005 us: Task0.Register ends
500005 us: Task0.AddSubtract begins
500007 us: Task0.AddSubtract ends
500007 us: Task0.PID begins
500022 us: Task0.PID ends
500022 us: Task2.Register begins
500027 us: Task2.Register ends
500027 us: Task2.AddSubtract begins
500029 us: Task2.AddSubtract ends
500029 us: Task2.Register begins
500034 us: Task2.Register ends
500034 us: Task2.AddSubtract begins
500036 us: Task2.AddSubtract ends
500036 us: Task2.PID begins
500051 us: Task2.PID ends
500051 us: Task2.Const begins
500053 us: Task2.Const ends
```

Summary

- A simulation-based architecture exploration framework
  - Separate function/architecture models
  - Change mapping without significantly modifying models
  - Evaluate performance,
  - Explore architectural candidates and ‘discover’ new architectures

THANK YOU