Embedded Software Synthesis of Heterogeneous Dataflow Models

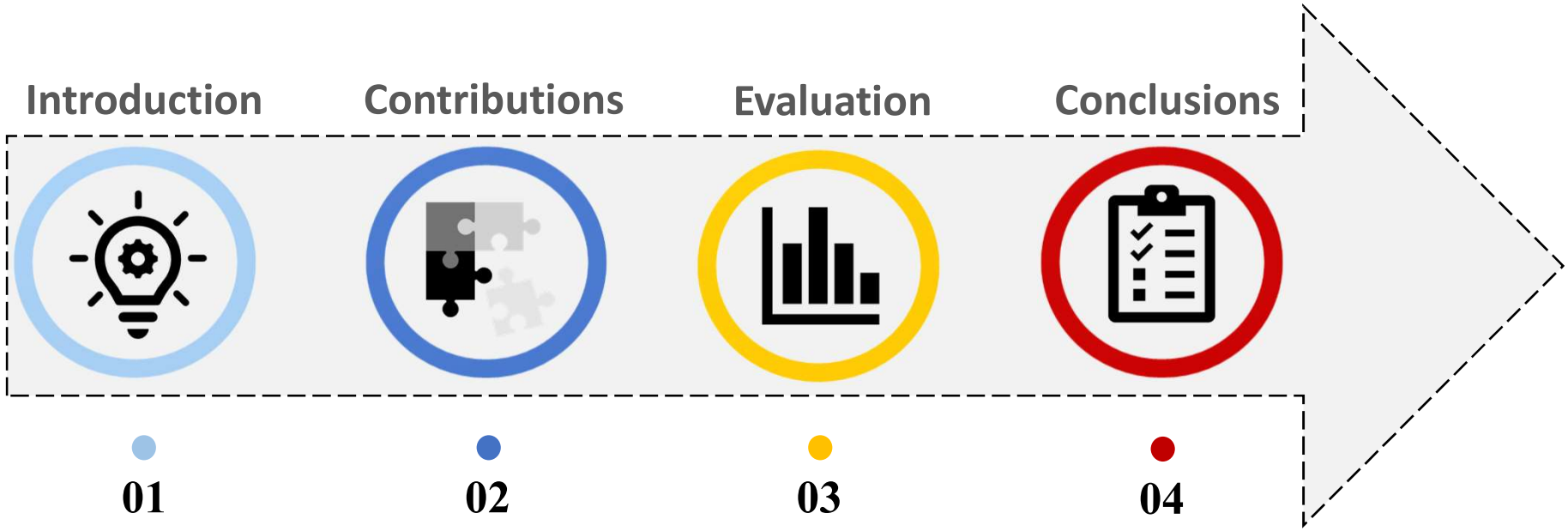
Omair Rafique

10.09.2021

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

(es) Embedded Systems Group
Department of Computer Science - University of Kaiserslautern

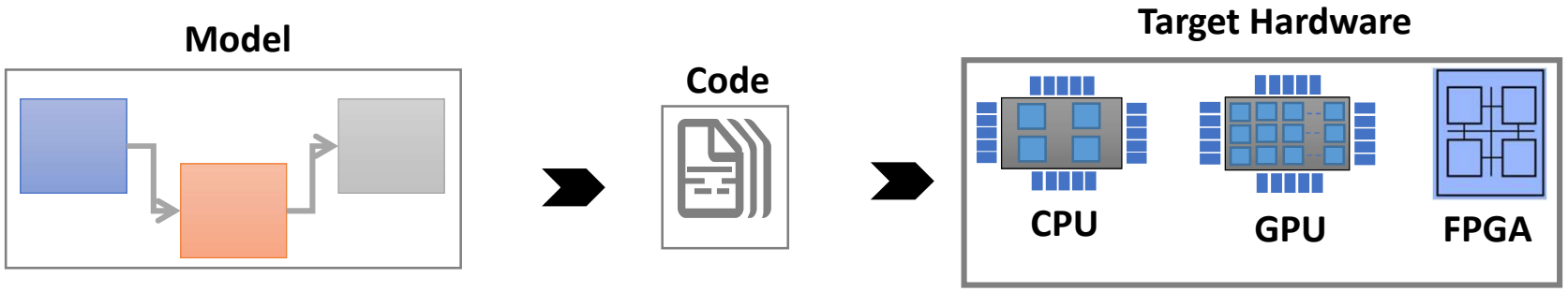
Contents


Introduction
Contributions
Evaluation
Conclusions
01
02
03
04

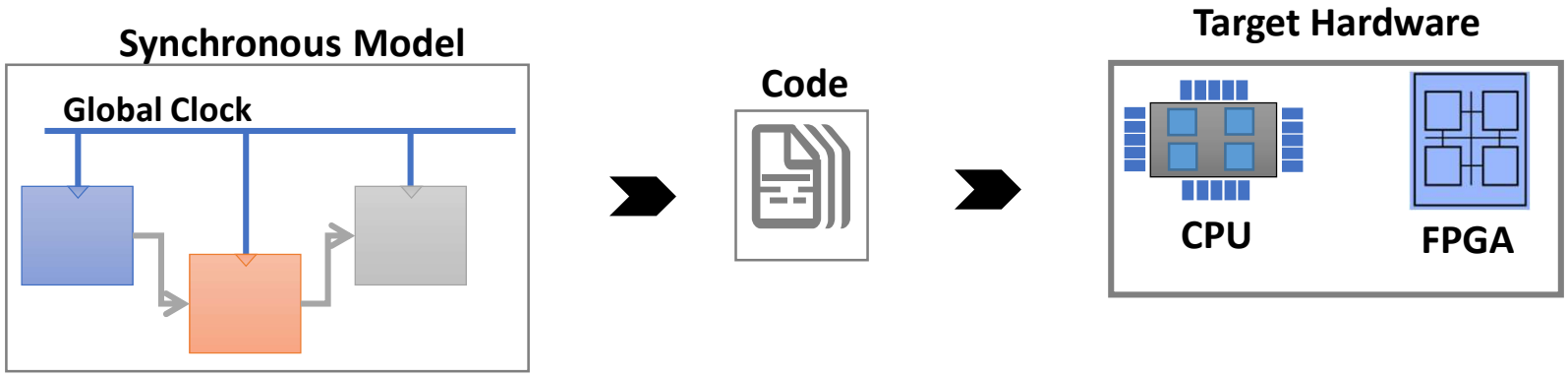
Model-based Design

- MoC precisely determines

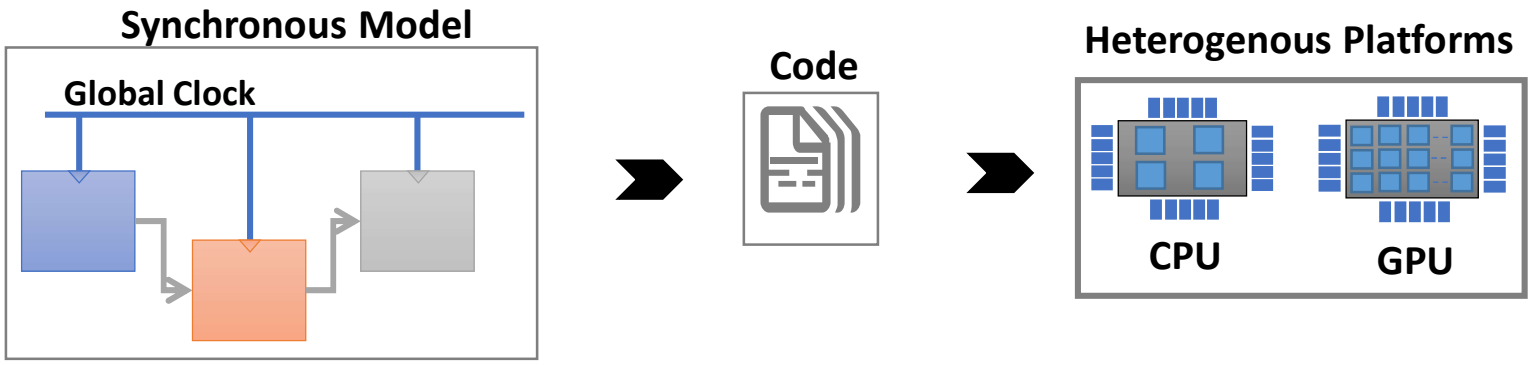
"why, when, which atomic action of a system is executed"



Model-based Design

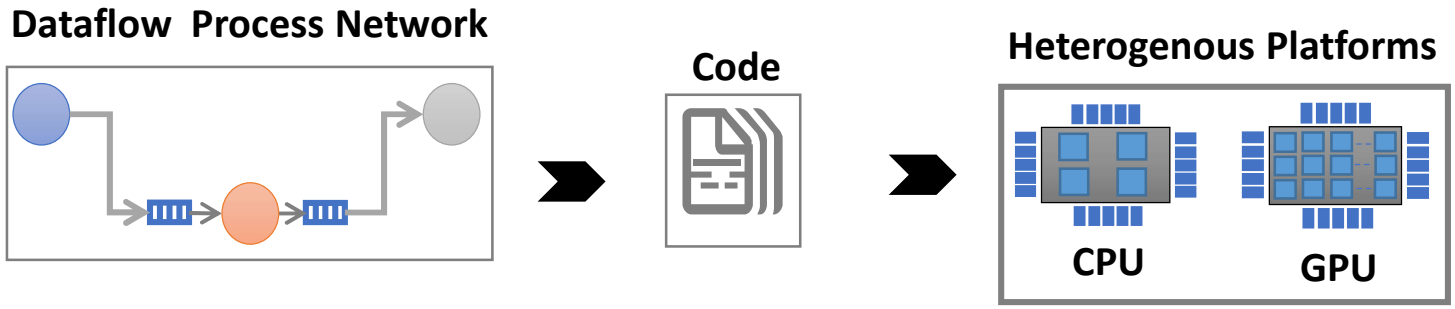


Model-based Design



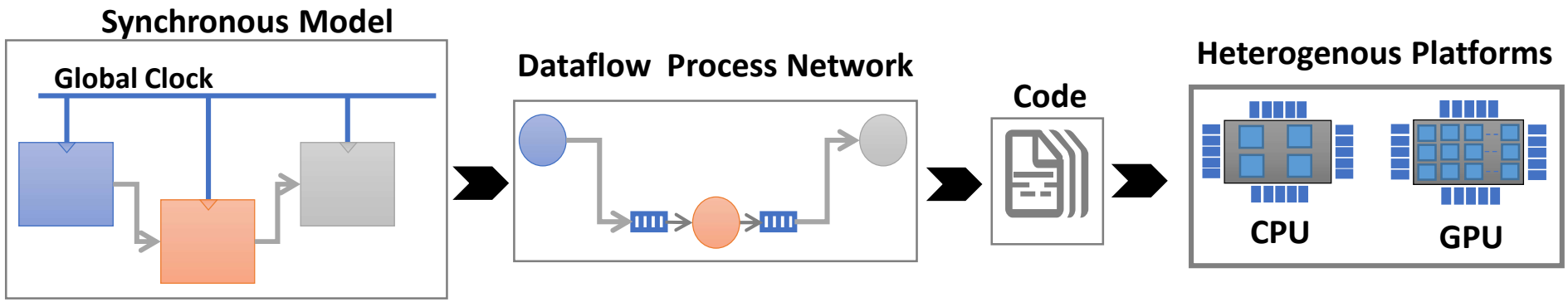
- synchronization overhead
- communication overhead

Model-based Design



- buffer boundedness
- deadlock-freeness

Model-based Design



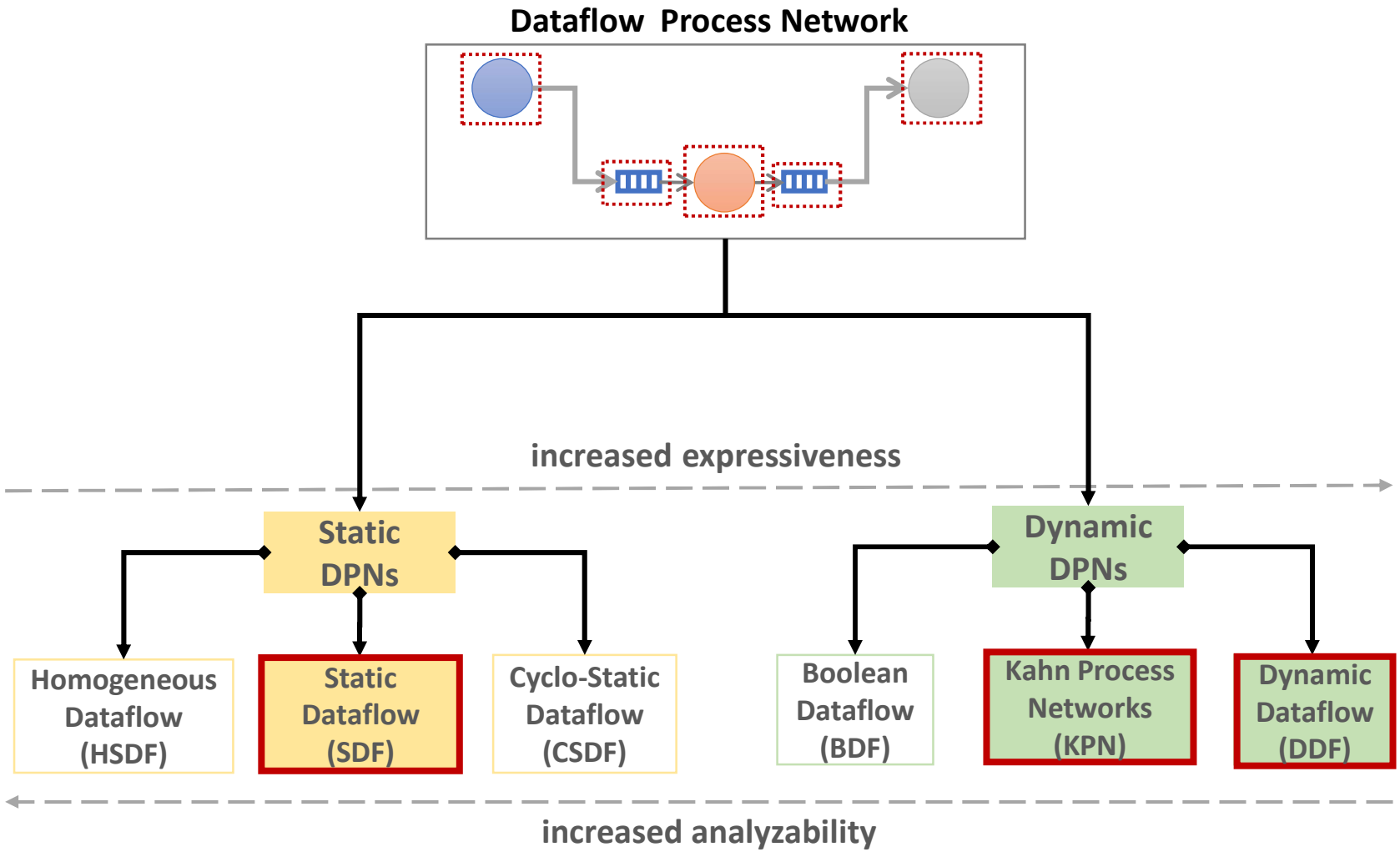
Model-based Design
‚the starting point‘

„Averest framework [http://www.averest.org/]“

„Model-based Design of Embedded Systems by Desynchronization [Yu Bai, 2016]“

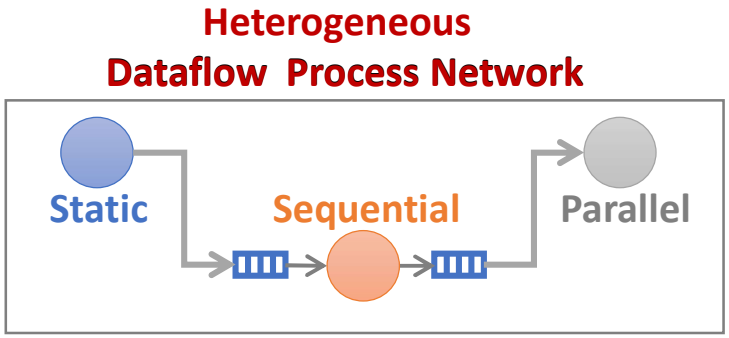
Motivation

‚dataflow process network (DPN)‘



Motivation

‚dataflow process network (DPN)‘



Static Dataflow (SDF)	Kahn Process Networks (KPN)	Dynamic Dataflow (DDF)
-static behaviors	-sequential behaviors	-parallel behaviors
-statically determined data	-dynamically determined data	-dynamically determined data
-static evaluation	-sequential evaluation	-independent evaluation

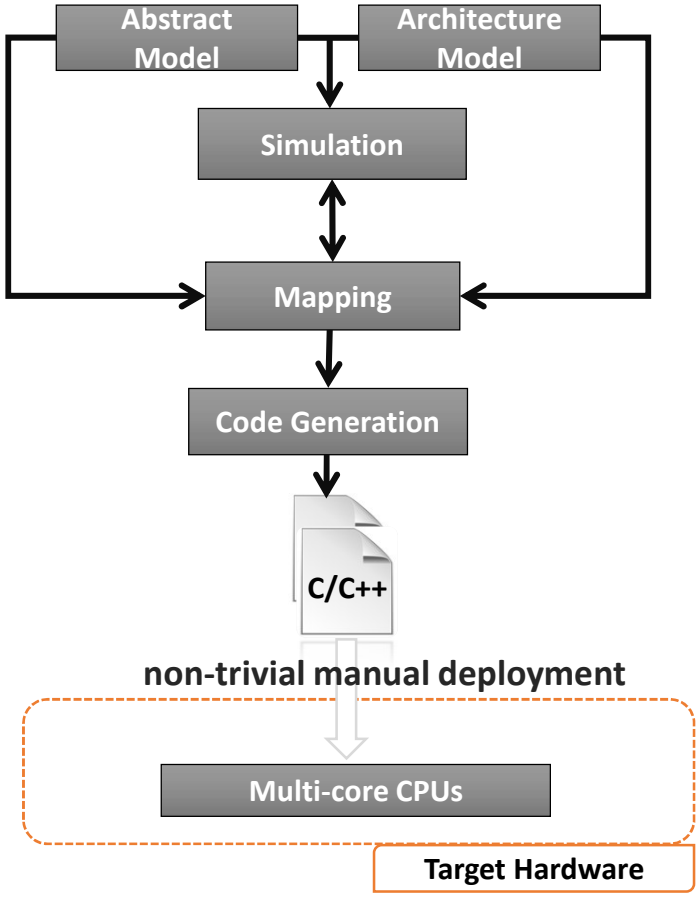
Motivation
‚related work‘

- state-of-the-art design tools for modeling
  - Ptolemy project (Eker et al., 2003)
  - SysteMoC (Haubelt et al., 2006)
  - FERAL (Kuhn et al., 2013)
  - emphasis on the design and analysis of systems
  - synthesis facility

„lack of automatic synthesis methods“

Motivation
‚related work‘

- state-of-the-art design tools for synthesis
  - (Yviquel et al., 2013)
  - (Lund et al., 2015)
  - (Boutellier et al., 2018)
  - based on a specific dataflow MoC
  - ORCC framework (Yviquel et al., 2013)
  - heterogeneous DPNs
  - performance degradation
  - manual deployment

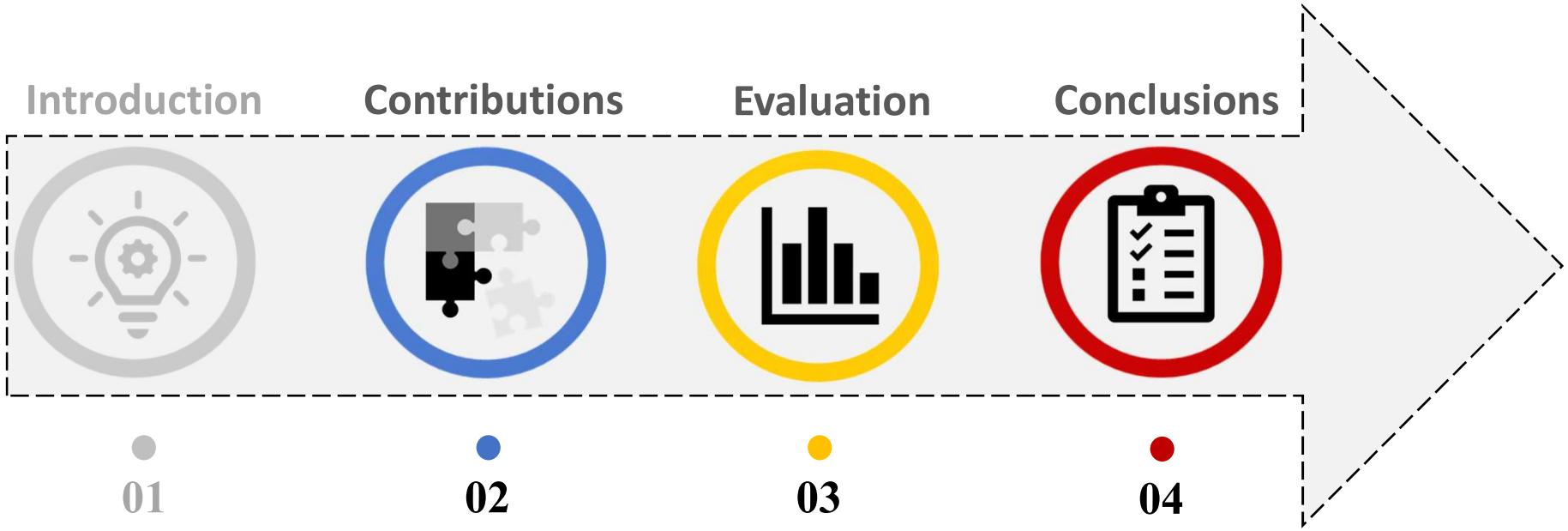


„simply employing DPNs not enough“

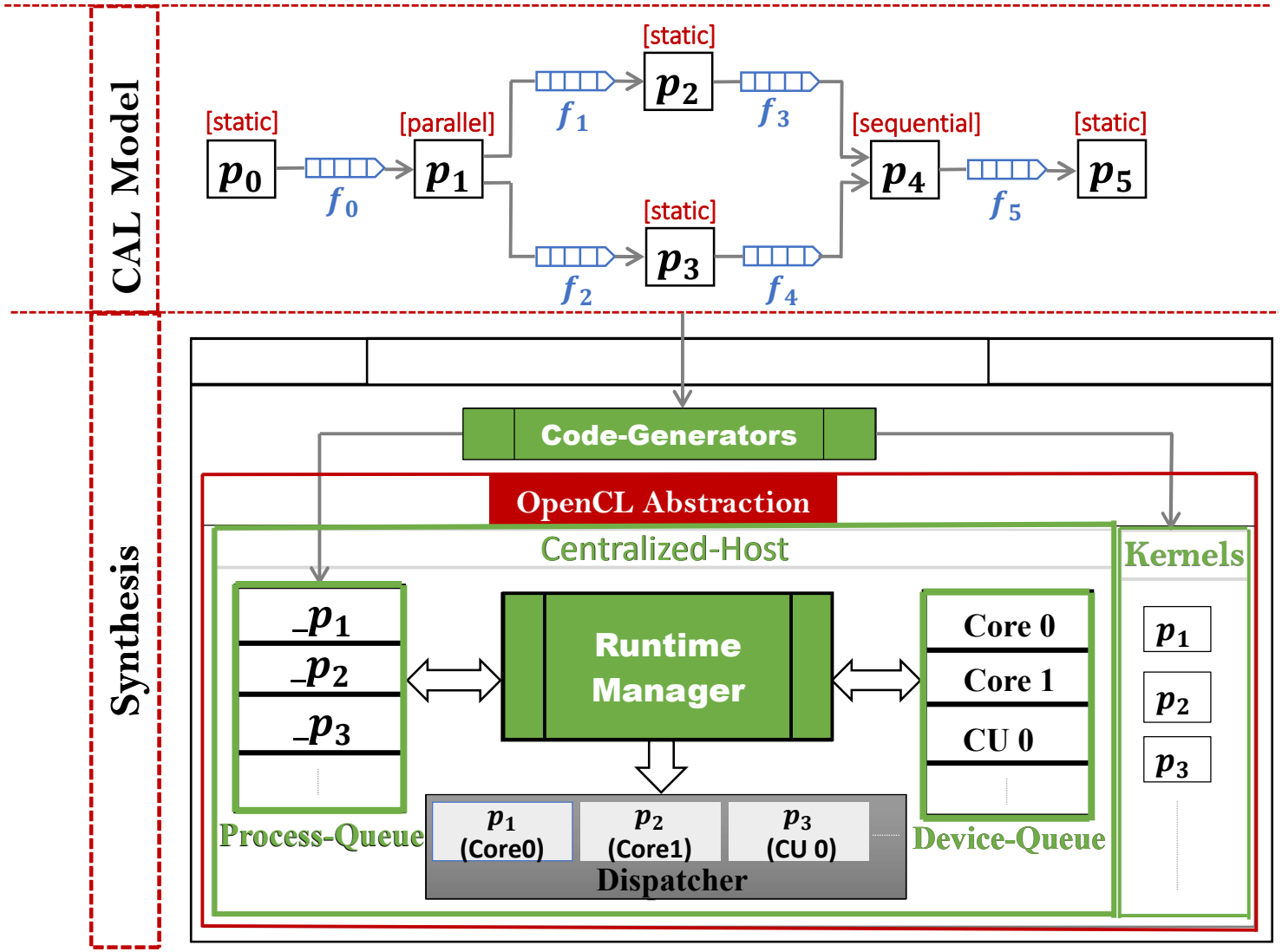
Motivation
‚research goals‘

- a common model-based design tool
  - focuses on the software synthesis of DPNs
  - enables automatic implementation of different dataflow MoCs
    - SDF, KPN and DDF
    - systematic exploration of the tradeoffs
- a smarter synthesis method
  - exploits heterogeneity in the network
  - scheduling/execution precisely based on the kinds of processes
- further automating the design process
  - mapping models on cross-vendor COTS target hardware
  - integral part of the synthesis method

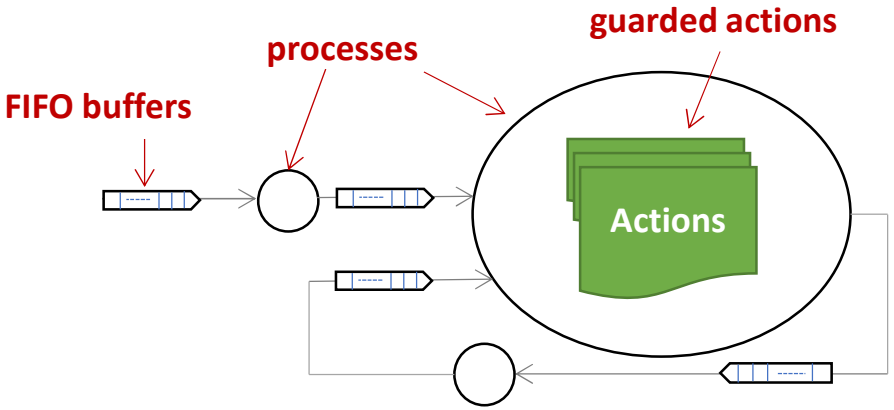
Contents


Introduction
Contributions
Evaluation
Conclusions
01
02
03
04

The Design Flow
‚overview‘



The Design Flow
‚the general CAL model‘

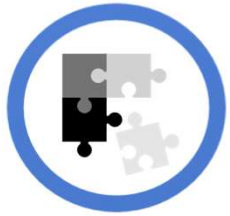



```
actor merge() int X1, int X2 ==> int Y1
act1: action X1: [x1] ==> Y1: [y1]
      guard true
      do
        y1:= x1;
      end
act2: action X2:[x2] ==> Y1: [y1]
      guard true
      do
        y1:= x2;
      end
end
```

Definition 1 (Input Constraint):
each input channel must have sufficient input tokens

Definition 2 (Output Constraint):
each output channel must contain space for output tokens

Definition 3 (Guard Constraint):
required values on the inputs are available i.e., the guard must be true



The Design Flow
‚the supported dataflow models‘

Static (SDF)

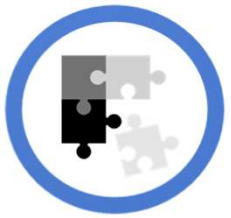
```
actor ITE() int X1, int X2, int X3 ==> int Y
 act1: action X1: [x1], X2:[x2], X3:[x3] ==>
Y: [y]
      guard x1 >= 0
      do
         y:= x2;
      end
 act2: action X1:[x1], X2:[x2], X3:[x3] ==>
Y: [y]
      guard x1 < 0
      do
         y:= x3;
      end
end
```

Sequential (KPN)

```
actor split() int X1, int X2 ==> int Y1,
int Y2
 act1: action X1: [x1], X2:[x2] ==>
Y1: [y1]
      guard x1 = 1
      do
         y1:= x2;
      end
 act2: action X1:[x1], X2:[x2a, x2b] ==>
Y1: [y1], Y2:[y2]
      guard x1 = 2
      do
         y1:= x2a;
         y2:= x2b;
      end
end
```

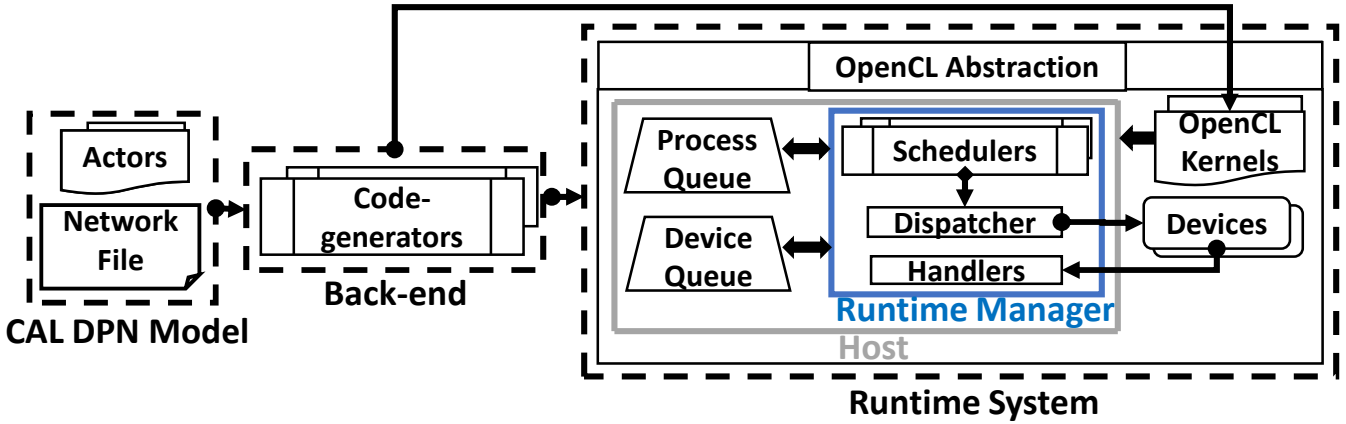
Parallel (DDF)

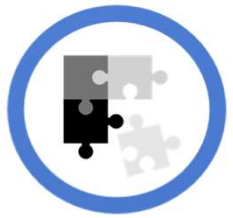
```
actor POR() bool X1, bool X2 ==>
bool Y
act1: action X1:[x1] ==> Y: [y]
      guard x1 = true
      do
         y:= 1;
      end
 act2: action X2:[x2] ==> Y: [y]
      guard x2 = true
      do
         y:= 1;
      end
 act3: action X1:[x1], X2:[x2] ==>
Y: [y]
      guard x1 = false and x2 = false
      do
         y:= 0;
      end
 end
```



The Design Flow

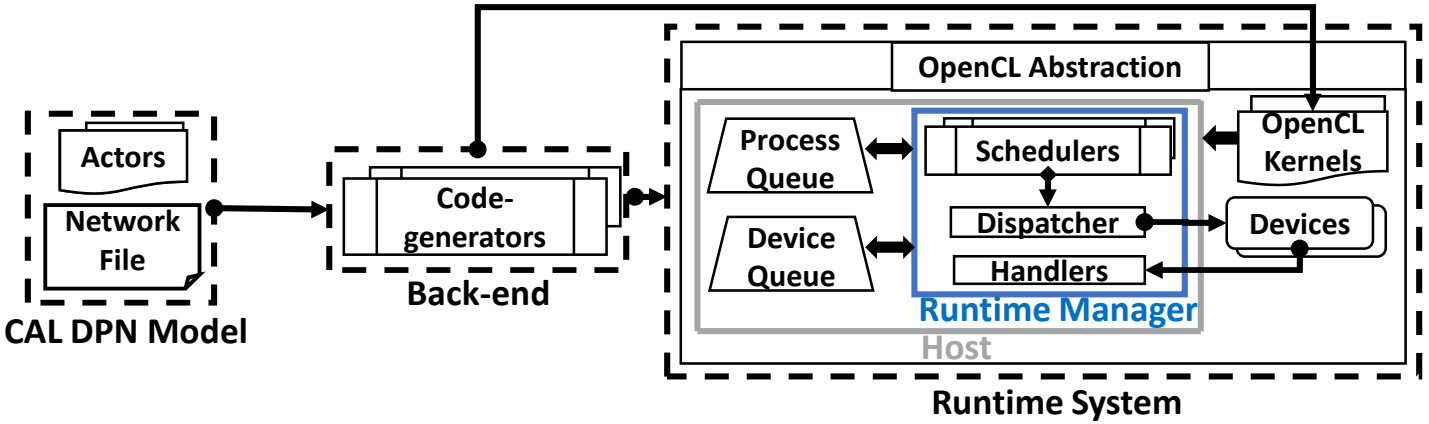
‚the synthesis tool chain‘



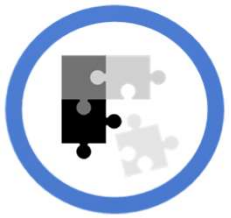


The Design Flow

‚the synthesis tool chain‘

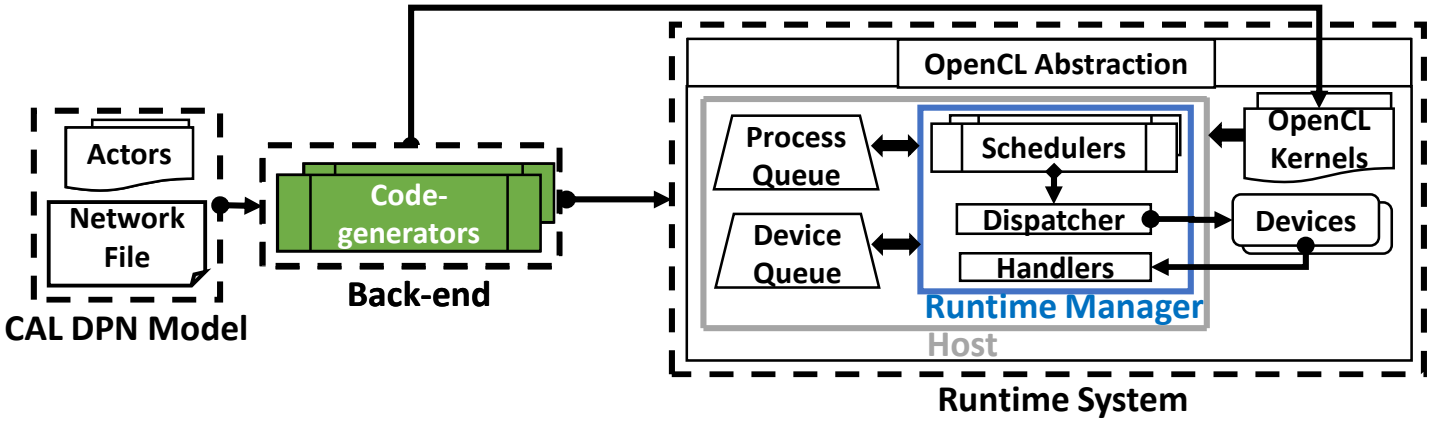


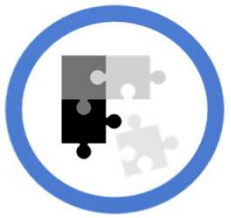

```
1  <?xml version="1.0" encoding="UTF-8"?><XDF name="seqStITE">
2      <Instance id="site">
3          <Class name="cal.site" moc="S"/>
4      </Instance>
5      <Instance id="printer1">
6          <Class name="cal.printer1" moc="S"/>
7      </Instance>
8      <Connection dst="site" dst-port="X1" src="" src-port="i1"/>
9      <Connection dst="site" dst-port="X2" src="" src-port="i2"/>
10     <Connection dst="site" dst-port="X3" src="" src-port="i3"/>
11     <Connection dst="printer1" dst-port="X" src="site" src-port="Y"/>
12 </XDF>
```



The Design Flow

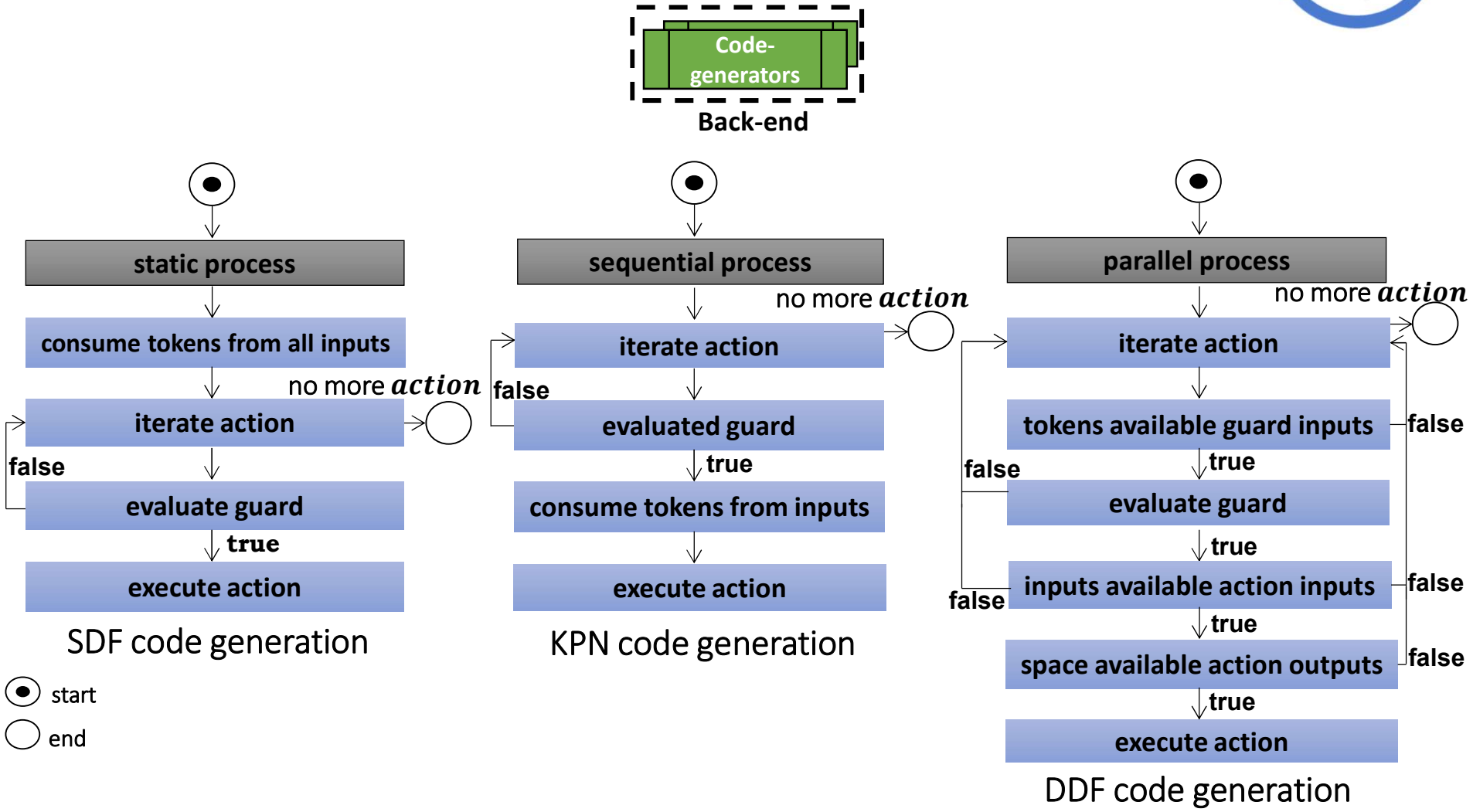
‚the synthesis tool chain‘

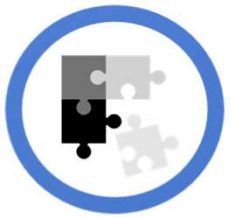




The Design Flow

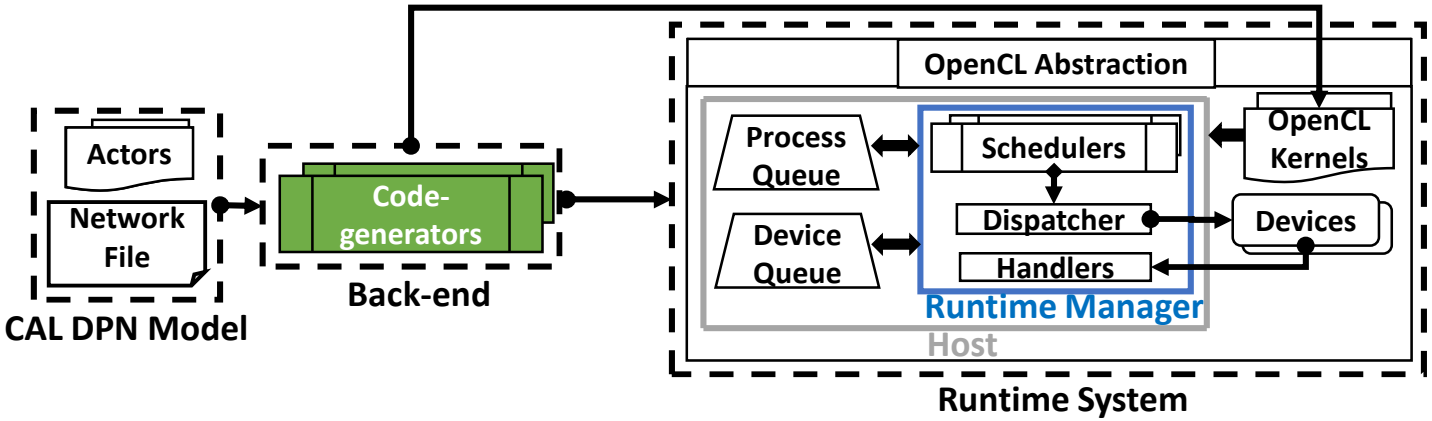
, the synthesis tool chain‘





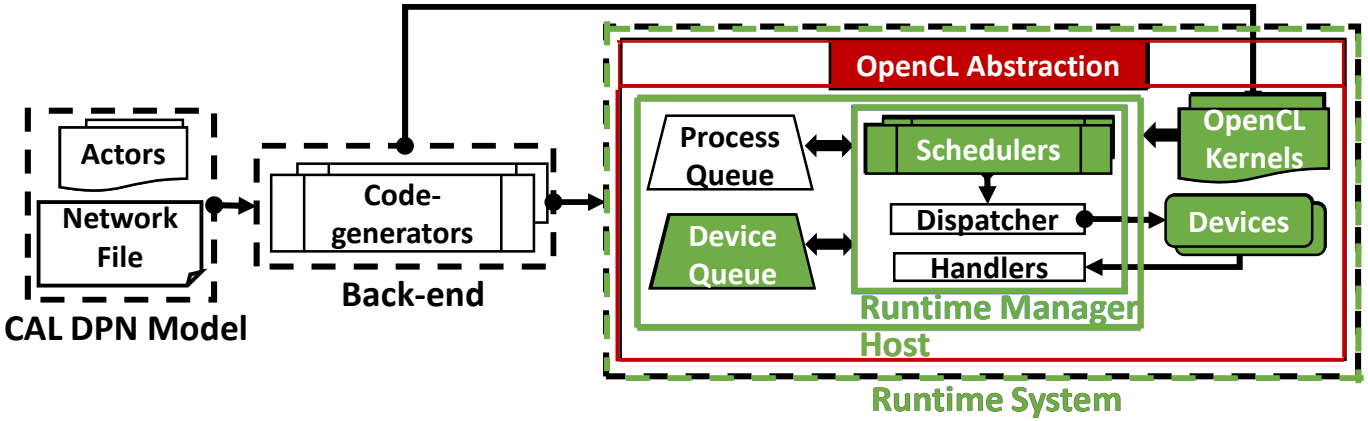
The Design Flow

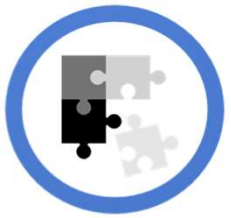
‚the synthesis tool chain‘



The Design Flow

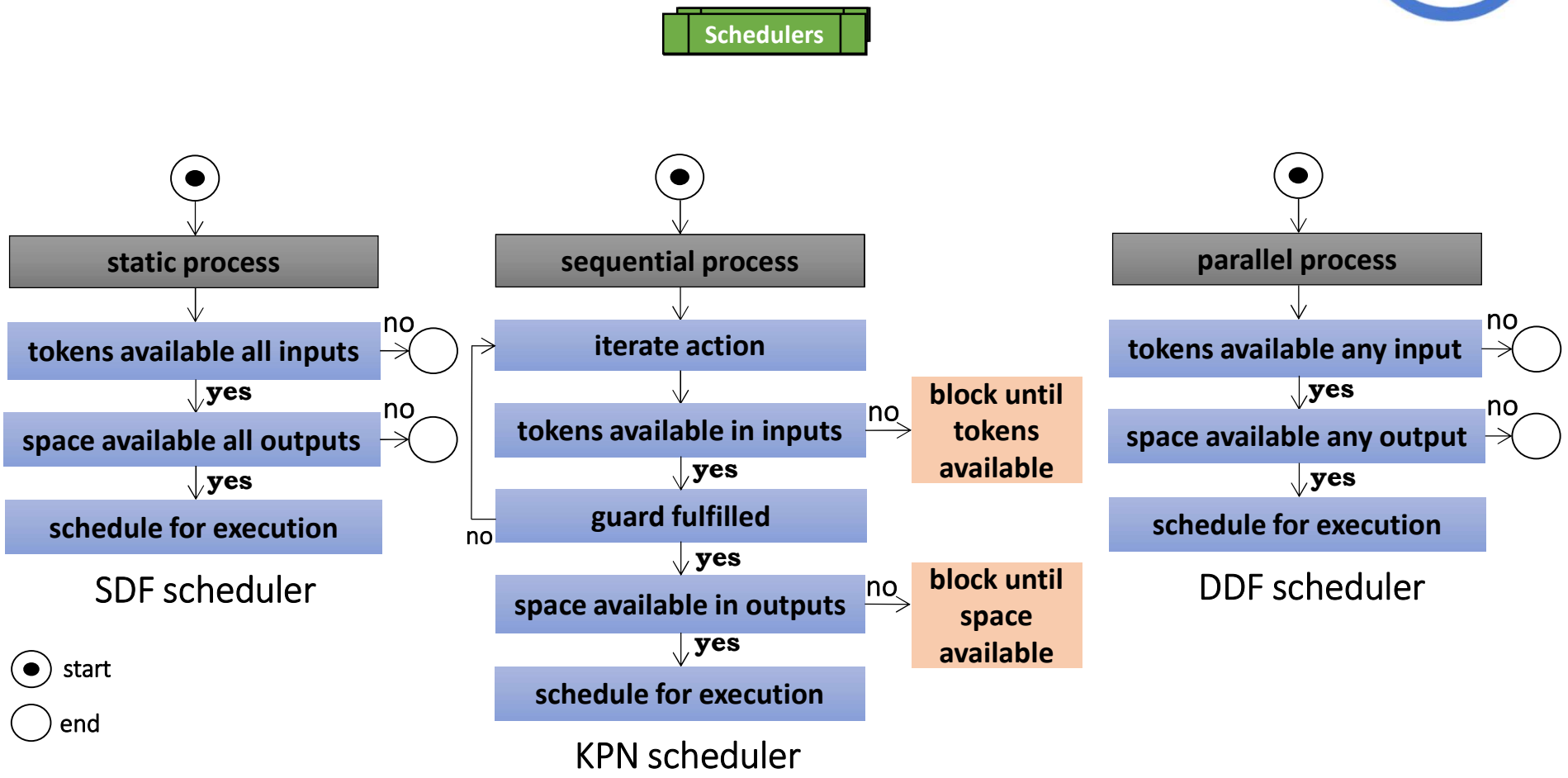
‚the synthesis tool chain‘

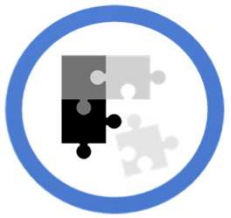




The Design Flow

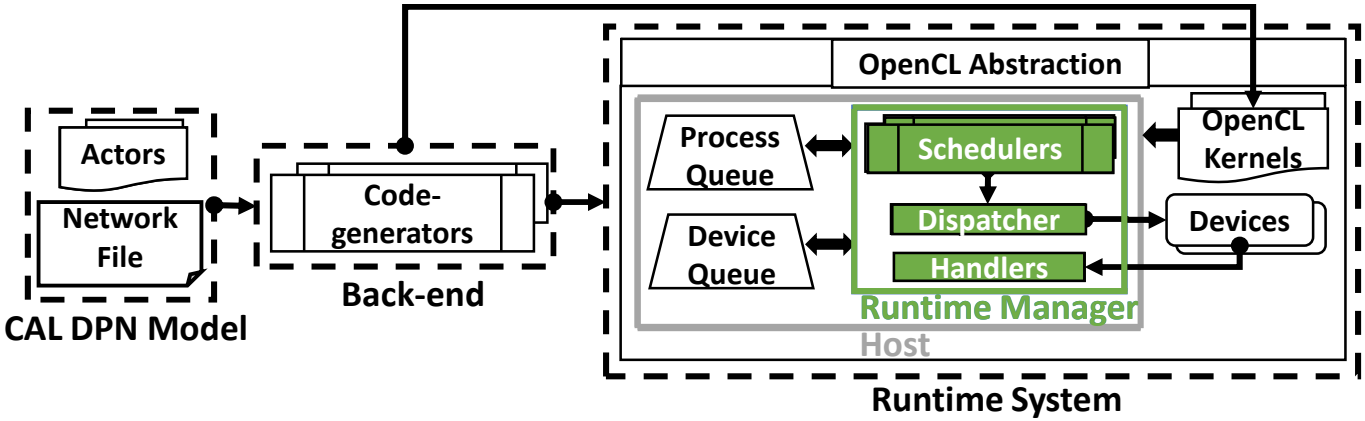
, the synthesis tool chain‘

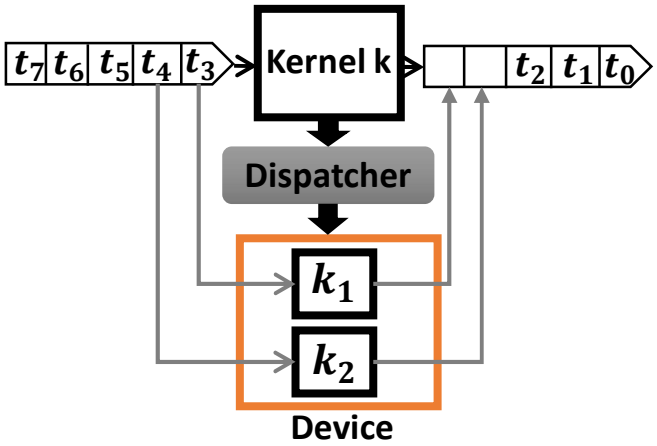




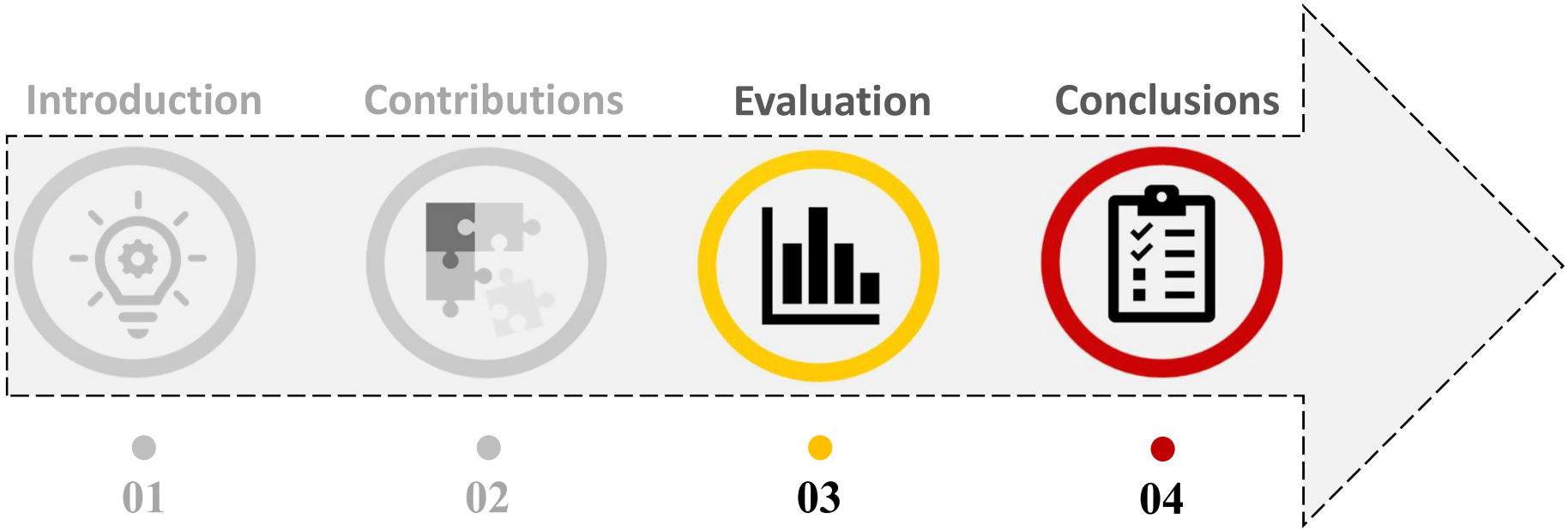
The Design Flow

‚the synthesis tool chain‘





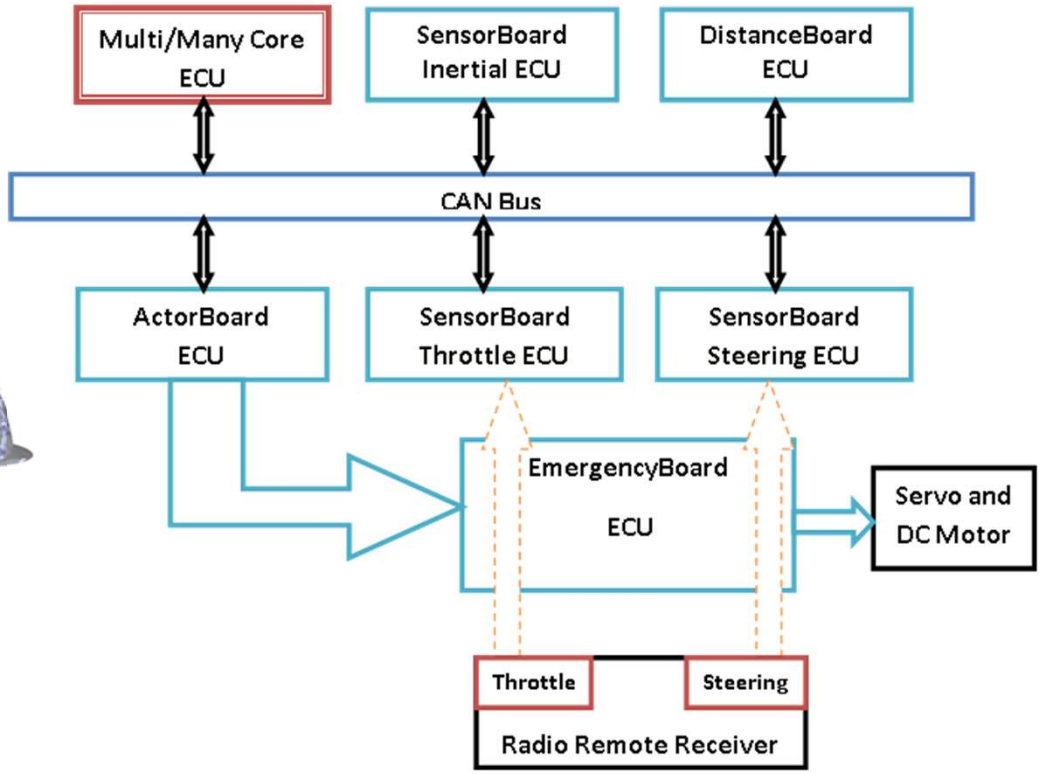
Contents


Introduction
Contributions
Evaluation
Conclusions
01
02
03
04

Experimental Setup
‚the target devices‘

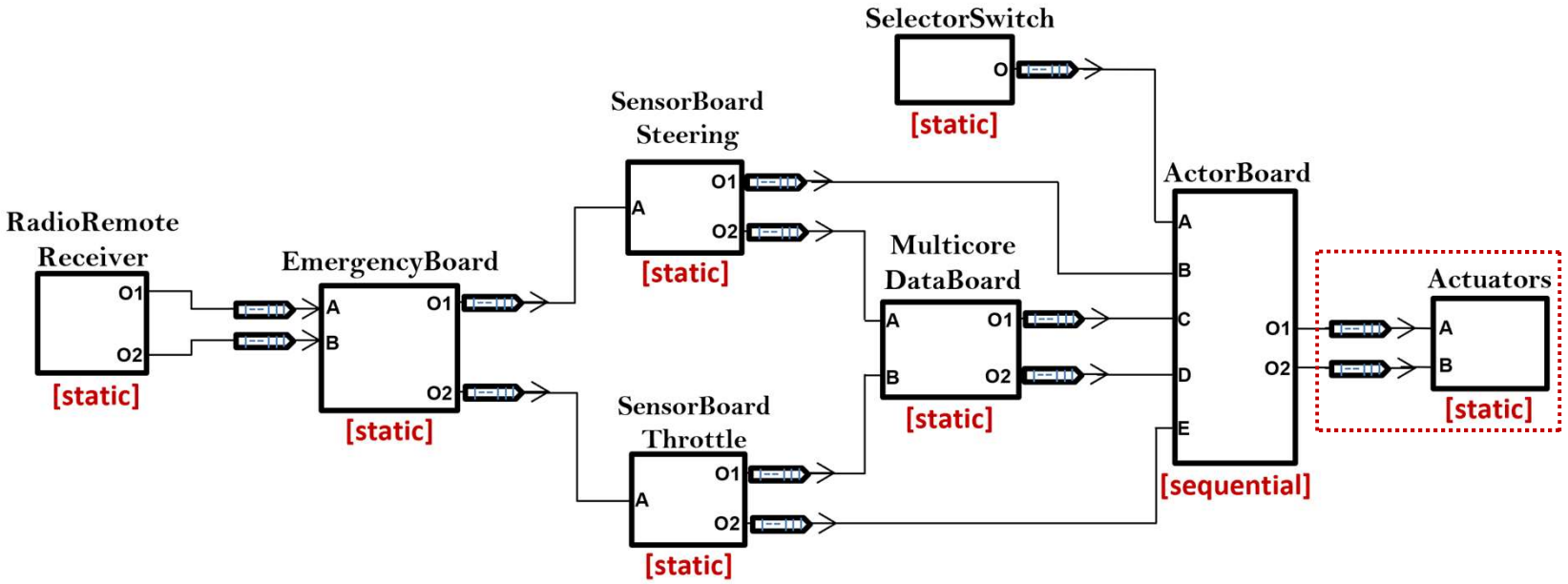
CPU1: Intel i5-8460U			
#physical cores	#logical cores	base frequency	RAM
4	4	3.2 GHz	8 GB
CPU2: Intel i7-8650U			
#physical cores	#logical cores	base frequency	RAM
4	8	1.9 GHz	8 GB
GPU1: Intel UHD Graphics 620			
#shaders	#compute units	base clock	RAM
192	24	300 MHz	-
GPU2: AMD Radeon HD 5450 GPU			
#shaders	#compute units	base clock	RAM
80	2	650 MHz	512 MB
GPU3: NVIDIA GeForce GTX 1050			
#shaders	#compute units	base clock	RAM
640	5	1354 MHz	2 GB

Case-Study I
‚the ConceptCar‘



Case-Study I
‚dataflow emulation of the ConceptCar (open-loop setting)‘

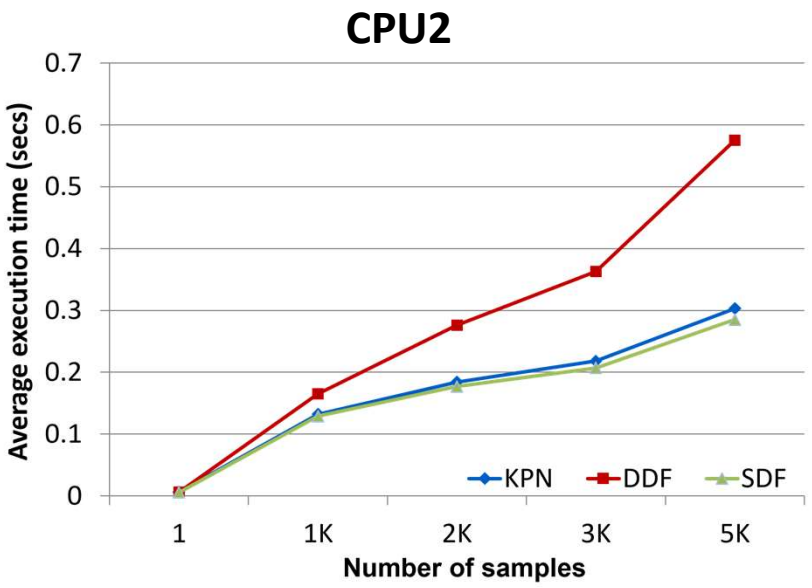
Evaluation

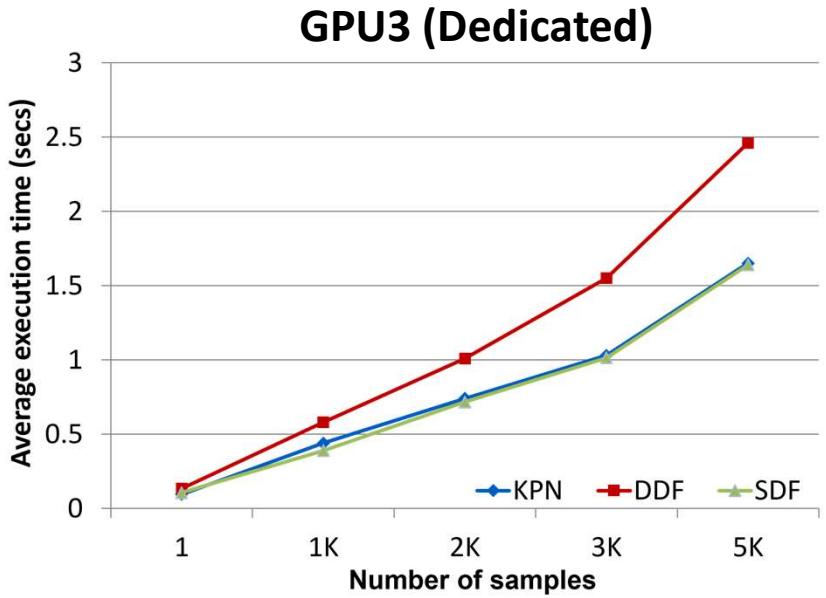


Case-Study I
‚results: open-loop setting‘

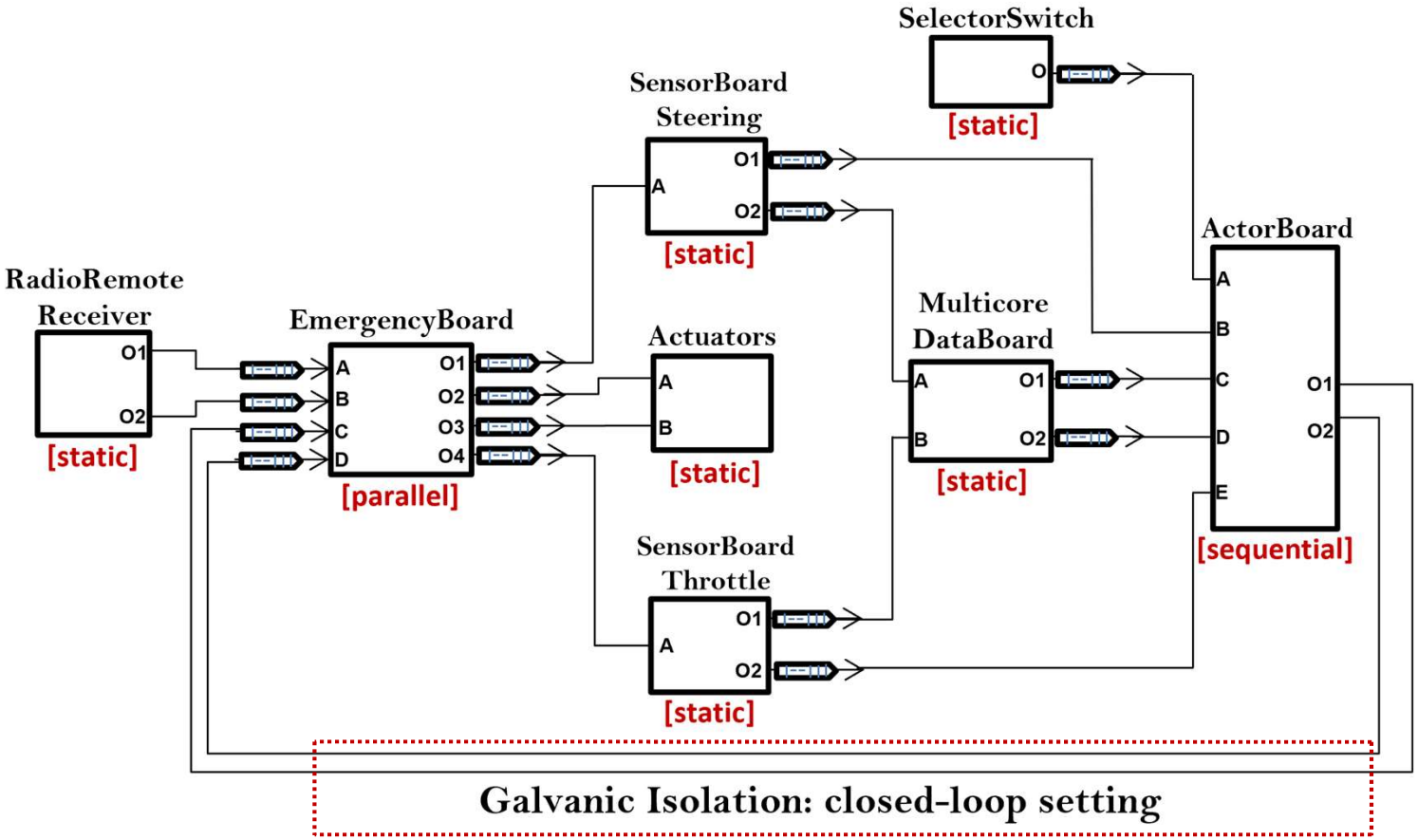
Evaluation

- generate implementations based on the individual dataflow MoCs
  - does using a more generalized dataflow MoC than needed affect performance?





Case-Study I
‚dataflow emulation of the ConceptCar (closed-loop setting)‘

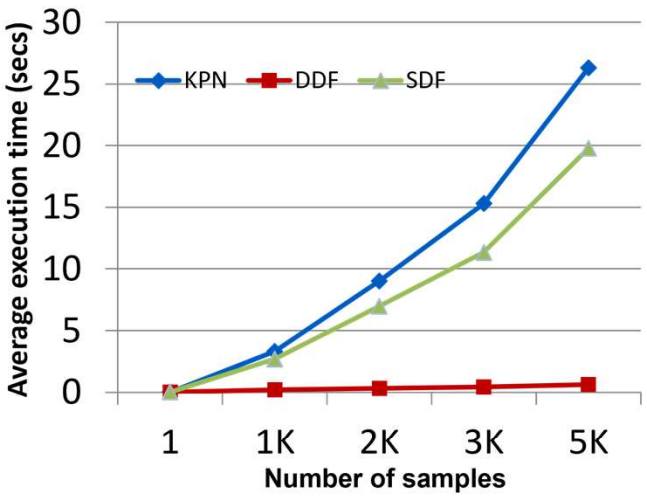


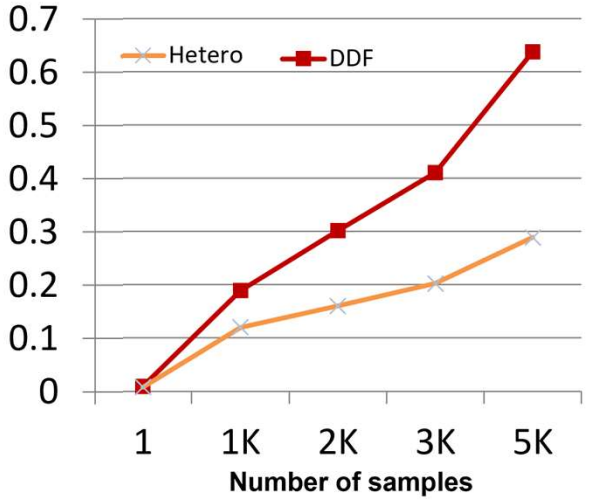
Case-Study I

‚results: closed-loop setting‘

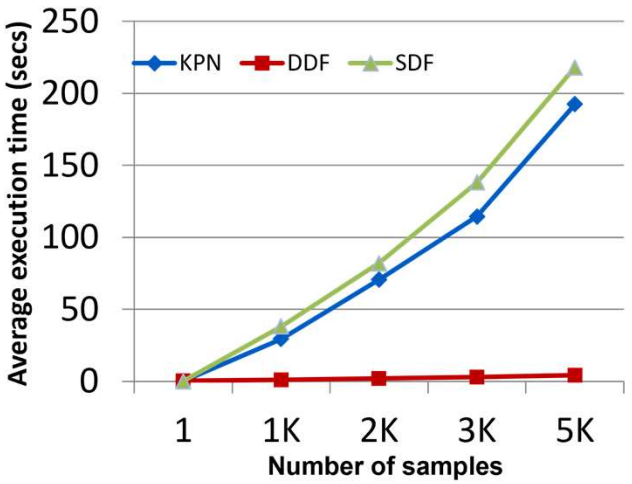
- how does a feedback loop in the network affect performance?
- does exploiting heterogeneity improve performance?

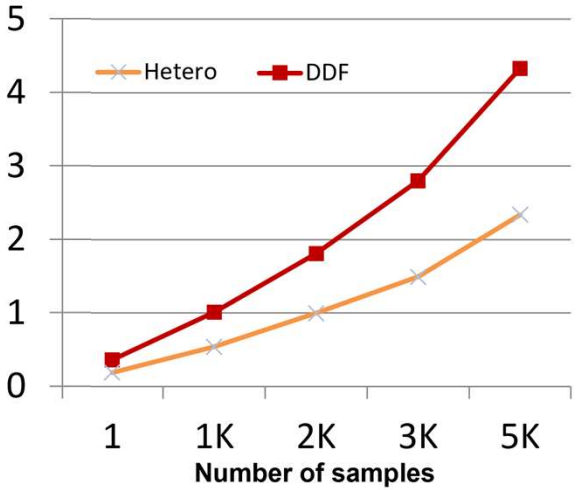
CPU2





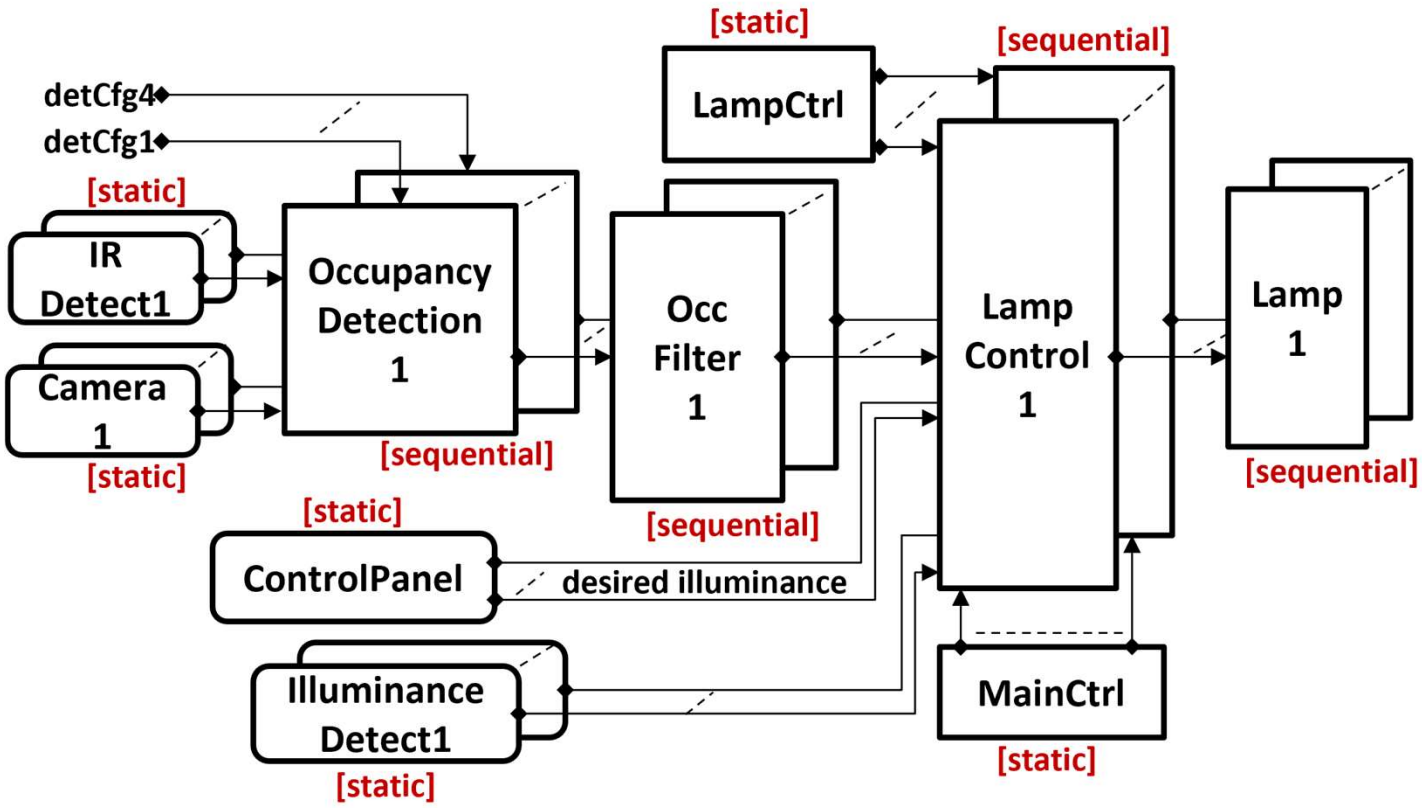
GPU3 (Dedicated)





Case-Study II

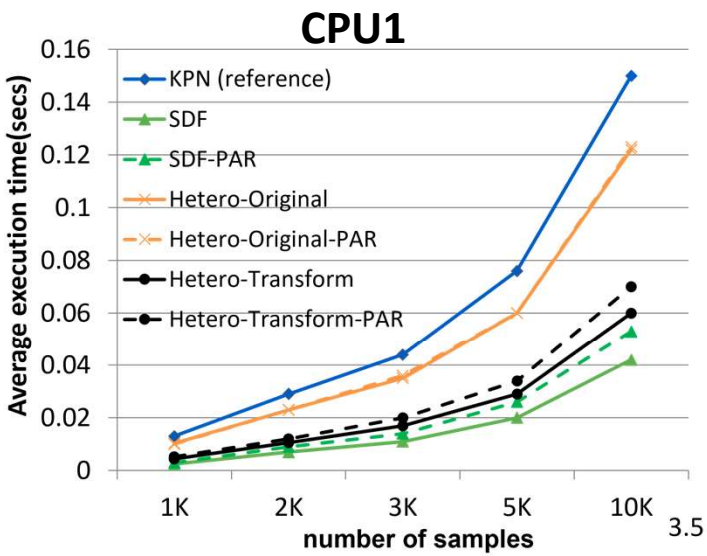
‚the smart building automation system‘

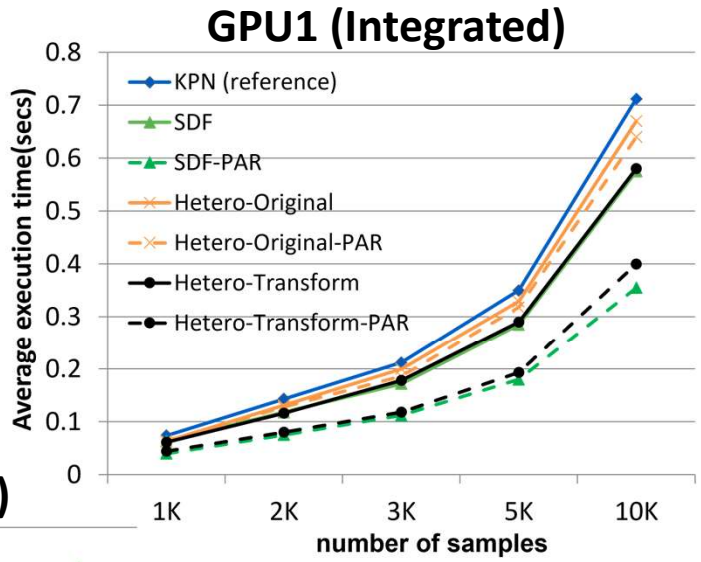


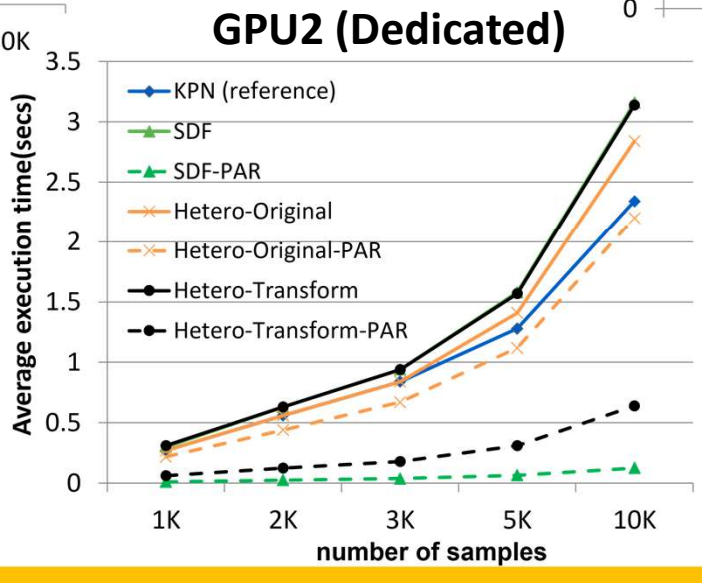
Case-Study II
‚results: end-to-end performance‘

Evaluation

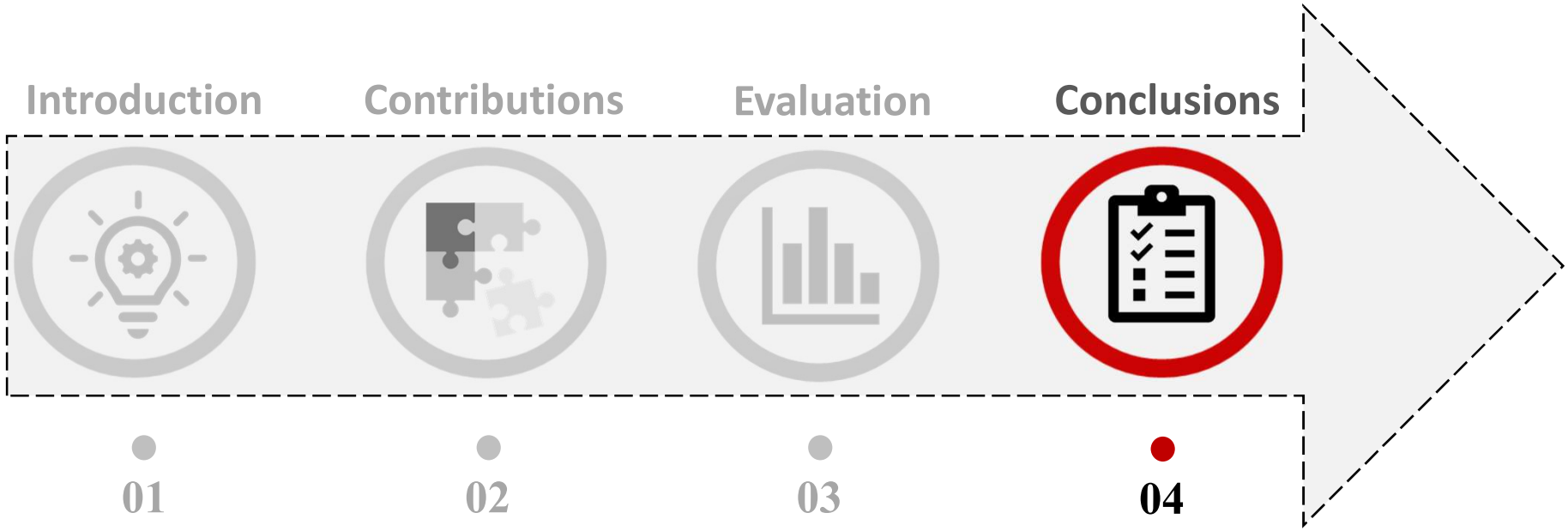
- does exploiting heterogeneity improve performance?





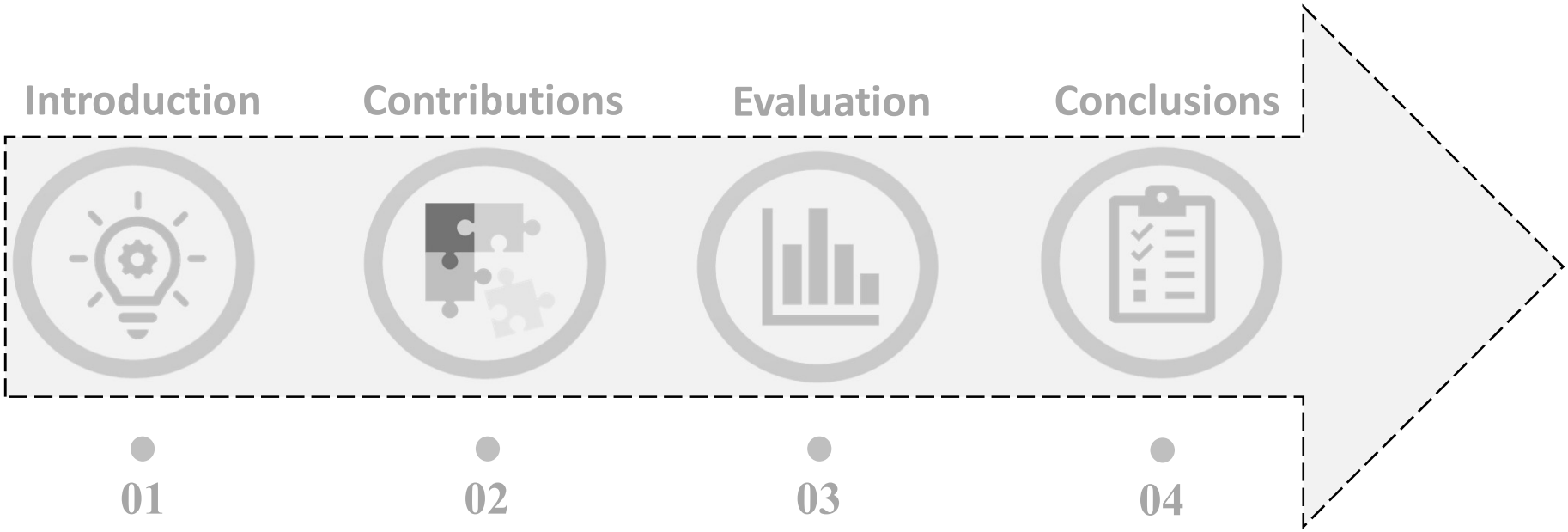


Contents


Introduction
Contributions
Evaluation
Conclusions
01
02
03
04

Summary

- a common dynamic environment for synthesis
  - goal 1: systematic exploration of the tradeoffs
  - implementations based on the precise dataflow MoCs
  - goal 2: a smarter synthesis method that exploits heterogeneity
  - ability to exploit heterogeneity significantly improves performance
  - goal 3: a design process for cross-vendor portability
  - employing OpenCL as an integral part of the synthesis process
    - systematic deployment of generated versions on various COTS hardware



THANK YOU FOR LISTENING!!