CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

# Analysis and software synthesis of KPN applications

**Jeronimo Castrillon**
Chair for Compiler Construction
TU Dresden
jeronimo.castrillon@tu-dresden.de

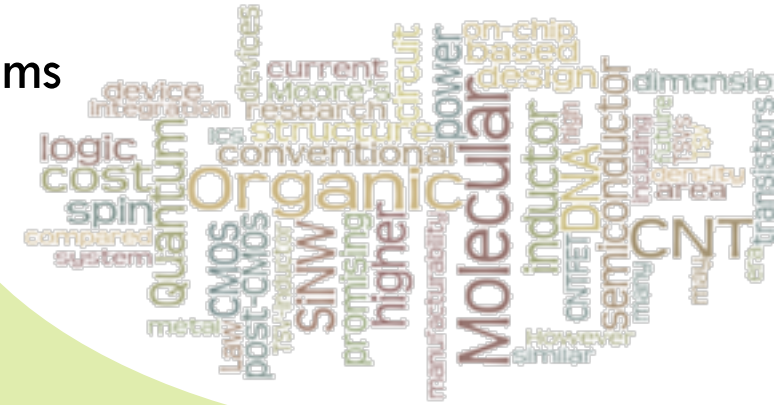
DREAMS Seminar
UC Berkeley, CA. October 22nd 2015

# Acknowledgements


- Institute for Communication Technologies and Embedded Systems (ICE), RWTH Aachen
- Silexica Software Solutions GmbH
- German Cluster of Excellence: Center for Advancing Electronics Dresden (www.cfaed.tu-dresden.de)

# Context: Cfaed

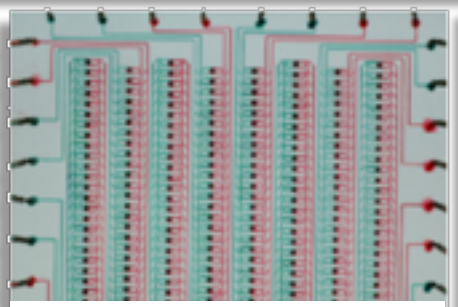
- Programming flows for future heterogeneous systems




**SW Layers and programming models**

**Hardware abstractions**

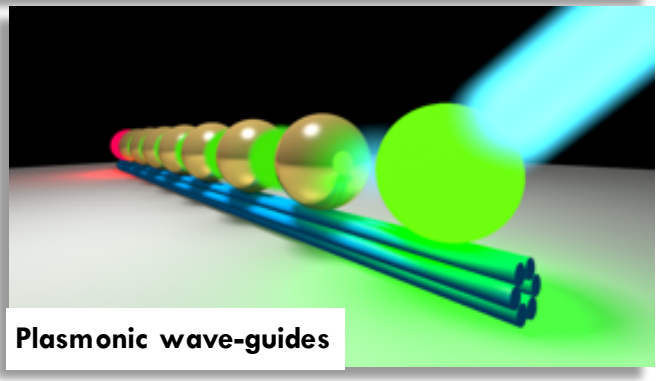
…



Chemical processing [Voigt14]


[Trommer15]

Reconfigurable HW with Si-Nanowires



Plasmonic wave-guides

Courtesy:
Thorsten
Lars-Schmidt

# Outline

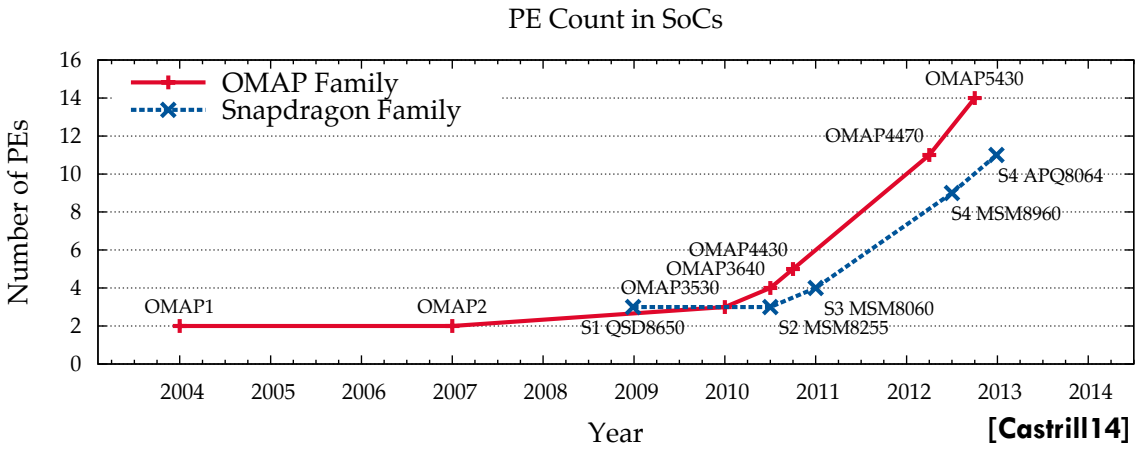
- Motivation
- Input specs
- Analysis and synthesis
- Code generation and evaluation
- Summary

# Outline

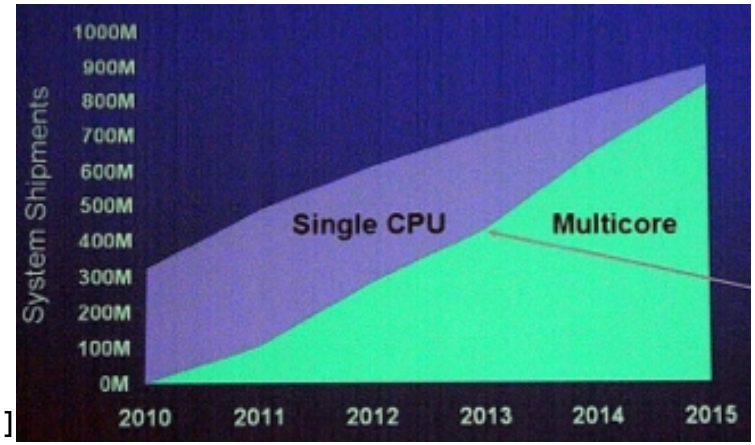
- **Motivation**
- Input specs
- Analysis and synthesis
- Code generation and evaluation
- Summary

# Multi-processor on systems and applications

- **HW complexity**
  - Increasing number of cores
  - Increasing heterogeneity
- Multi-cores everywhere
  - Ex.: Smartphones, tablets and e-readers
- **SW "complexity"**
  - Not anymore a simple control loop
  - Need expressive models



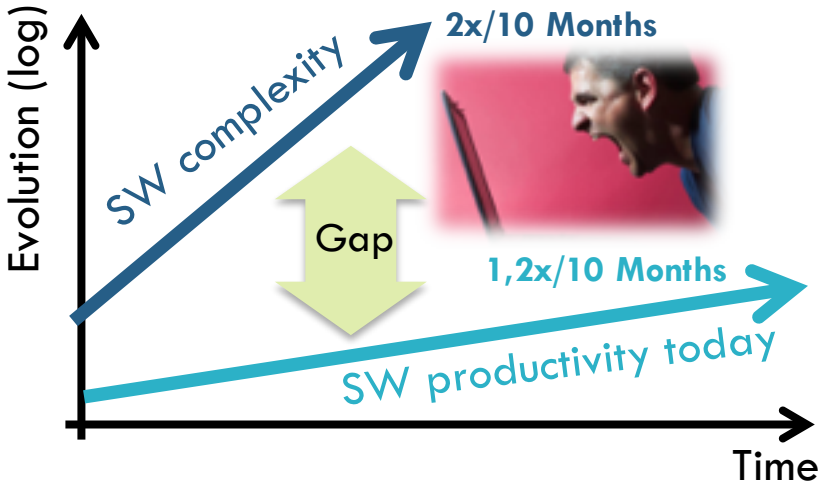

[Castrill14]




[EETimes11]

# SW productivity gap

- SW-productivity gap: complex SW for ever-increasing complex HW
  - ➔ Cannot keep pace with requirements
  - ➔ Cannot leverage available parallelism
- Difficult to reason about time constraints
  - Even more difficult about energy consumption

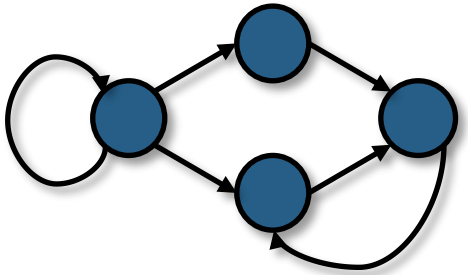



- Need domain-specific programming tools and methodologies!
  - ➔ Parallelizing sequential codes
  - ➔ Parallel abstractions and mapping methods

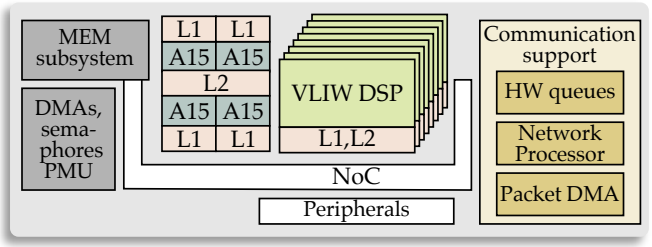
In this talk: One such a flow for **KPN applications** (multimedia & signal processing domains)

# Programming flow: Overview

KPN Application



Architecture model




Non-functional specification

Analysis

Synthesis

Code generation

Property models (timing, energy, error, …)

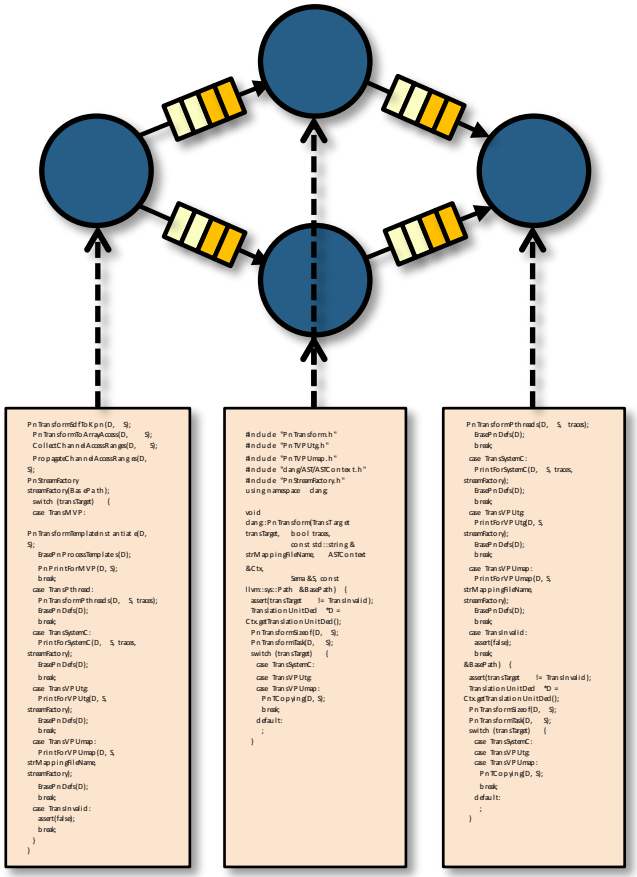
```
PNargs_ifft_r.ID = 6U;
    PNargs_ifft_r.PNchannel_freq_coef = f
    PNargs_ifft_r.PNnum_freq_coef = 0U;
    PNargs_ifft_r.PNchannel_time_coef = s
    PNargs_ifft_r.channel = 1;
    sink_left = IPCllmrf_open(3, 1, 1);
    sink_right = IPCllmrf_open(7, 1, 1);
    PNargs_sink.ID = 7U;
    PNargs_sink.PNchannel_in_left = sink_
    PNargs_sink.PNnum_in_left = 0U;
    PNargs_sink.PNchannel_in_right = sink
    PNargs_sink.PNnum_in_right = 0U;
    taskParams.arg0 = (xdc_UArg)&PNargs_s
    taskParams.priority = 1;

ti_sysbios_knl_Task_create((ti_sysbios_kr
&taskParams, &eb);
    glob_proc_cnt++;
    hasProcess = 1;
    taskParams.arg0 = (xdc_UArg)&PNargs_f
    taskParams.priority = 1;

ti_sysbios_knl_Task_create((ti_sysbios_kr
ft_Templ, &taskParams, &eb);
    glob_proc_cnt++;
    hasProcess = 1;
    taskParams.arg0 = (xdc_UArg)&PNargs_i
    taskParams.priority = 1;

ti_sysbios_knl_Task_create((ti_sysbios_kr
fft_Templ, &taskParams, &eb);
    glob_proc_cnt++;
    hasProcess = 1;
    taskParams.arg0 = (xdc_UArg)&PNargs_s
    taskParams.priority = 1;
```

# Outline

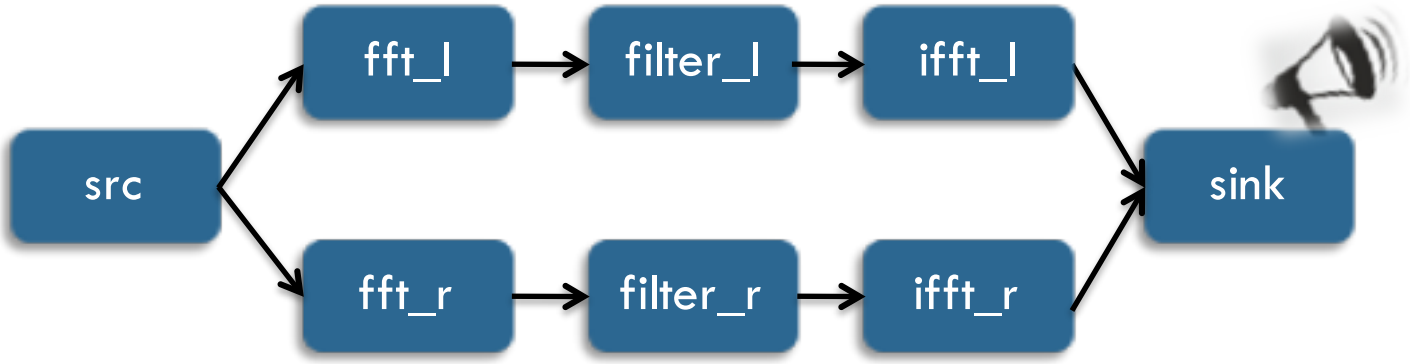
- Motivation
- **Input specs**
- Analysis and synthesis
- Code generation and evaluation
- Summary

# Kahn Process Networks (KPNs)

- Graph representation of applications
  - Processes **communicate only** over FIFO buffers
  - Good model for streaming applications
  - Good match for signal processing & multi-media



- Stereo digital audio filter



# Language: C for process networks

- FIFO Channels

```
typedef struct { int i; double d; } my_struct_t;
__PNchannel my_struct_t S;
__PNchannel int A = {1, 2, 3}; /* Initialization */
__PNchannel short C[2], D[2], F[2], G[2];
```

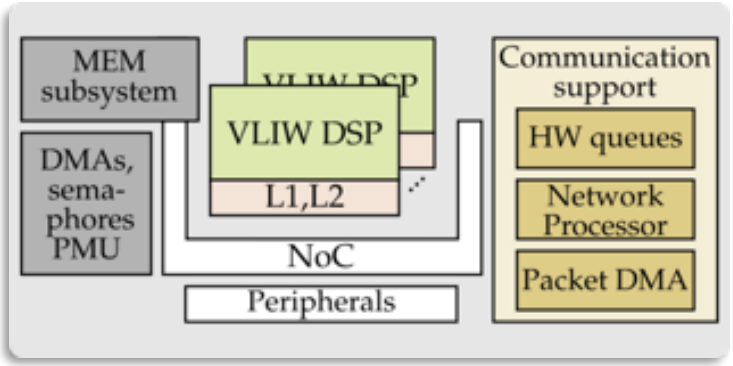
- Processes & networks

```
__PNkpn AudioAmp __PNin(short A[2]) __PNout(short B[2])
                 __PNparam(short boost){
  while (1)
    __PNin(A) __PNout(B) {
      for (int i = 0; i < 2; i++)
        B[i] = A[i]*boost;
    }}
__PNprocess Amp1 = AudioAmp __PNin(C) __PNout(F) __PNparam(3);
__PNprocess Amp2 = AudioAmp __PNin(D) __PNout(G) __PNparam(10);
```

[Sheng14]

# Architecture model

- System model including:
  - Topology, interconnect, memories
  - Computation: cost tables (as backup)
  - Communication: cost function (no contention)
- Example: Texas Instruments Keystone




```
-<Platform>
   <Processors List="dsp0 dsp1 dsp2 dsp3 dsp4 dsp5 dsp6 dsp7"/>
   <Memories List="local_mem_dsp0_L2 local_mem_dsp1_L2 local_mem_dsp2_L2
   local_smem_dsp1_L2 local_smem_dsp2_L2 local_smem_dsp3_L2 local_smem_dsp
   local_mem_dsp3_DDR local_mem_dsp4_DDR local_mem_dsp5_DDR local_mem_ds
   <CommPrimitives List="IPCll_SL2 IPCll_DDR EDMA3_SL2 EDMA3_DDR EDMA
 </Platform>
 <Processor Name="dsp0" CoreRef="DSPC66"/>
 <Processor Name="dsp1" CoreRef="DSPC66"/>
     ...
 <Processor Name="dsp7" CoreRef="DSPC66"/>
-<Memory>
   <LocalMemory Name="local_mem_dsp0_L2" Size="524288" BaseAddress_hex="00800000" ProcessorRef="dsp0"/>
 </Memory>
     ...
```

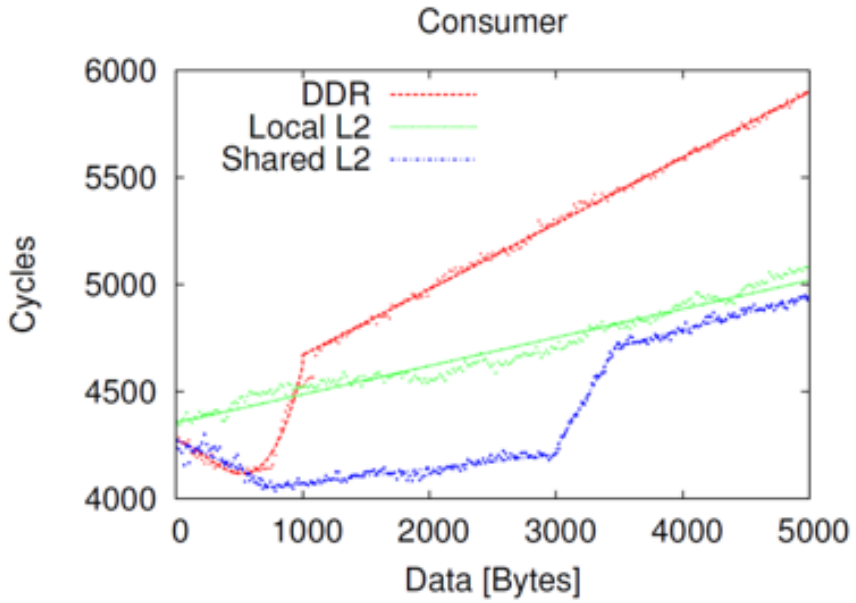
```
-<Core Name="DSPC66" CoreType="DSPC66" Category="DSP">
 -<MultiTaskingInfo MaxNumberOfTasks="-1">
   <ContextSwitchInfo StoreTime="1000" LoadTime="1000"/>
   <SchedulingPolicies List="FIFO PriorityBased"/>
  </MultiTaskingInfo>
 -<CostTable>
  -<Operation Name="Load">
   -<VariableType Name="Char">
     <Cost>1</Cost>
    </VariableType>
   -<VariableType Name="Double">
```

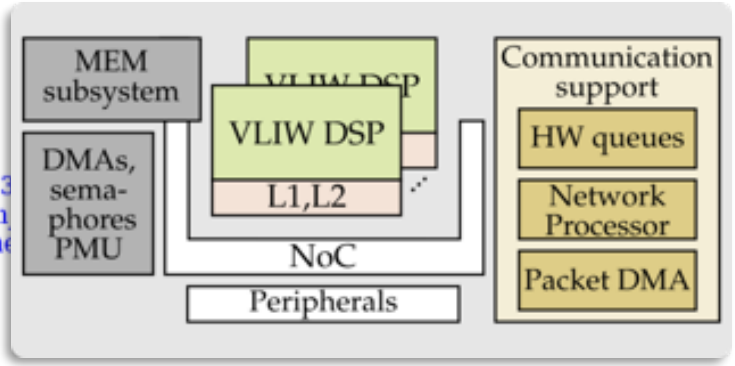
[Oden13]

# Architecture model: Communication

- Piecewise curve-fitting from measurements

```
-<Platform>
  <Processors List="dsp0 dsp1 dsp2 dsp3 dsp4 dsp5 dsp6 dsp7"/>
  <Memories List="local_mem_dsp0_L2 local_mem_dsp1_L2 local_mem_dsp2_L2 local_mem_dsp3
  local_smem_dsp1_L2 local_smem_dsp2_L2 local_smem_dsp3_L2 local_smem_dsp4_L2 local_smem
  local_mem_dsp3_DDR local_mem_dsp4_DDR local_mem_dsp5_DDR local_mem_dsp6_DDR local_me
  <CommPrimitives List="IPCII_SL2 IPCII_DDR EDMA3_SL2 EDMA3_DDR EDMA3_L2L2"/>
```



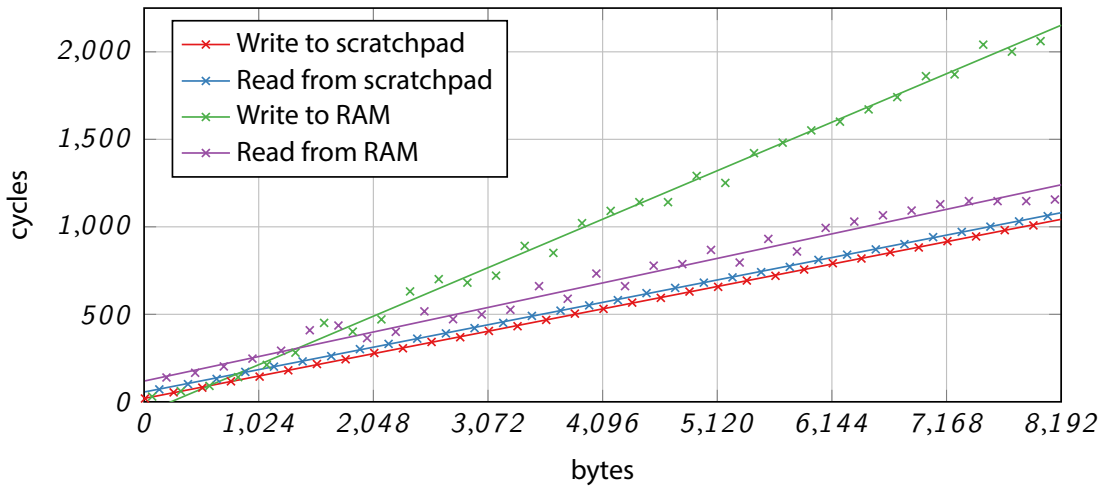

```
-<CommPrimitive>
 -<CPDMA Name="EDMA3_DDR">
    <Description>EDMA over DDR</Description>
   -<Costs>
      <Cost End="800" Function="11442.60163-0.15775*x"/>
      <Cost Start="801" Function="11204.94186+0.316143*x"/>
    </Costs>
    <DMAs List="local_mem_dsp0_DDR local_mem_dsp1_DDR local
  </CPDMA>
 </CommPrimitive>
```

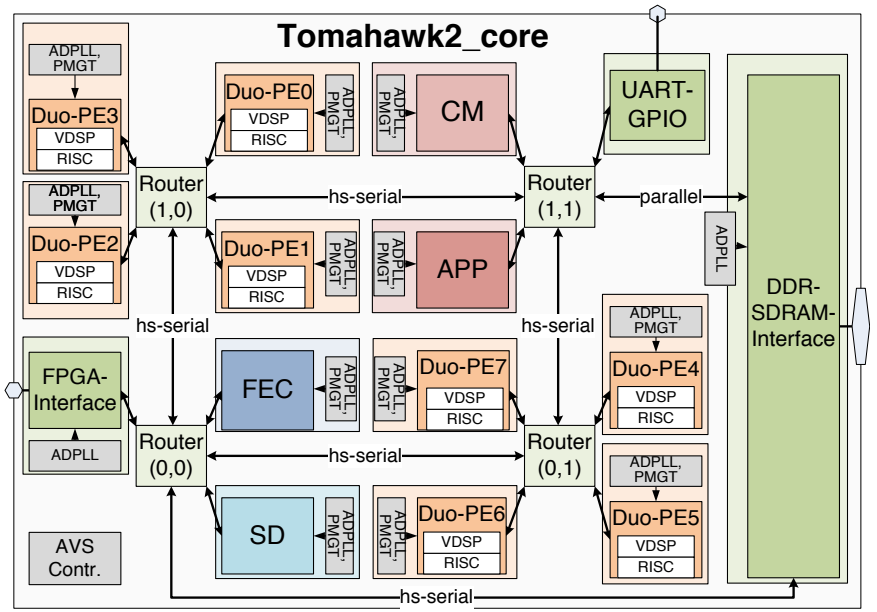
[Oden13]

# Architecture model: Communication (2)

- Models for Network on Chips (NoC)
- Channels can be mapped to
  - Local scratchpad (producer or consumer)
  - Global SDRAM

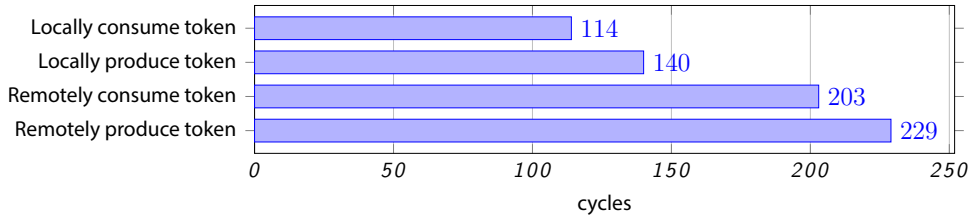
Cost model & measurement



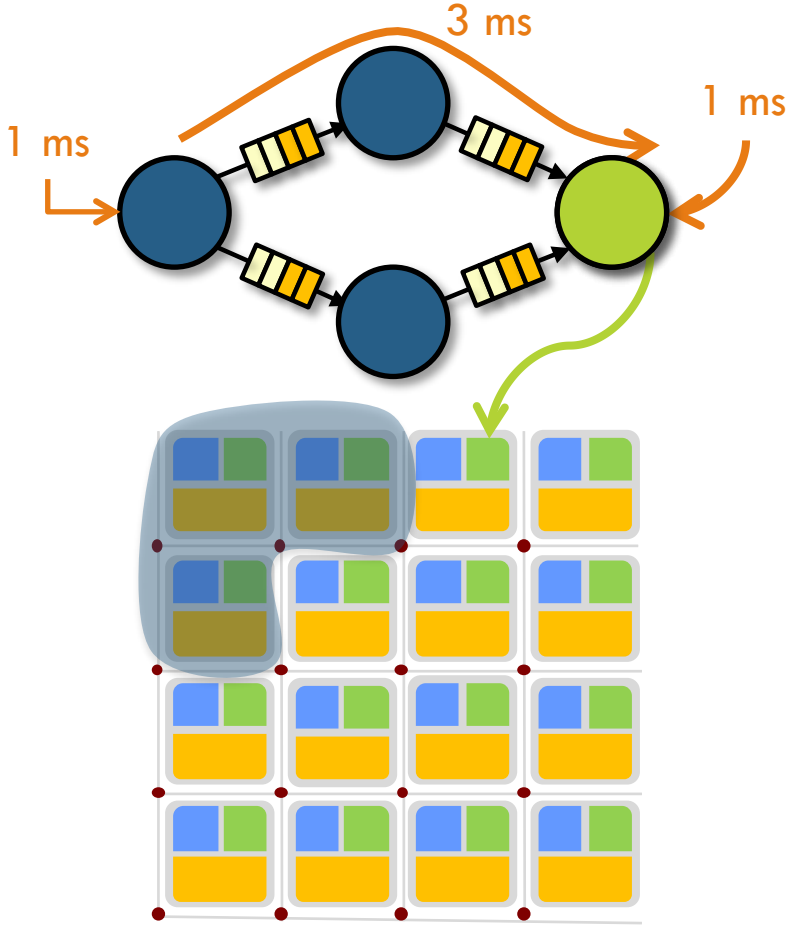

[Arnold13]

Static access costs



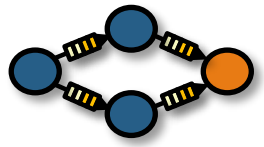
# Constraints

- Timing constraints
  - Process throughput
  - Latencies along paths
  - Time triggering
- Mapping constraints
  - Processes to processors
  - Channels to primitives
- Platform constraints
  - Subset of resources (processors or memories)
  - Utilization



# Algorithmic description

- Extended application specification
  - Selected processes are algorithmic kernels with **algorithmic parameters**



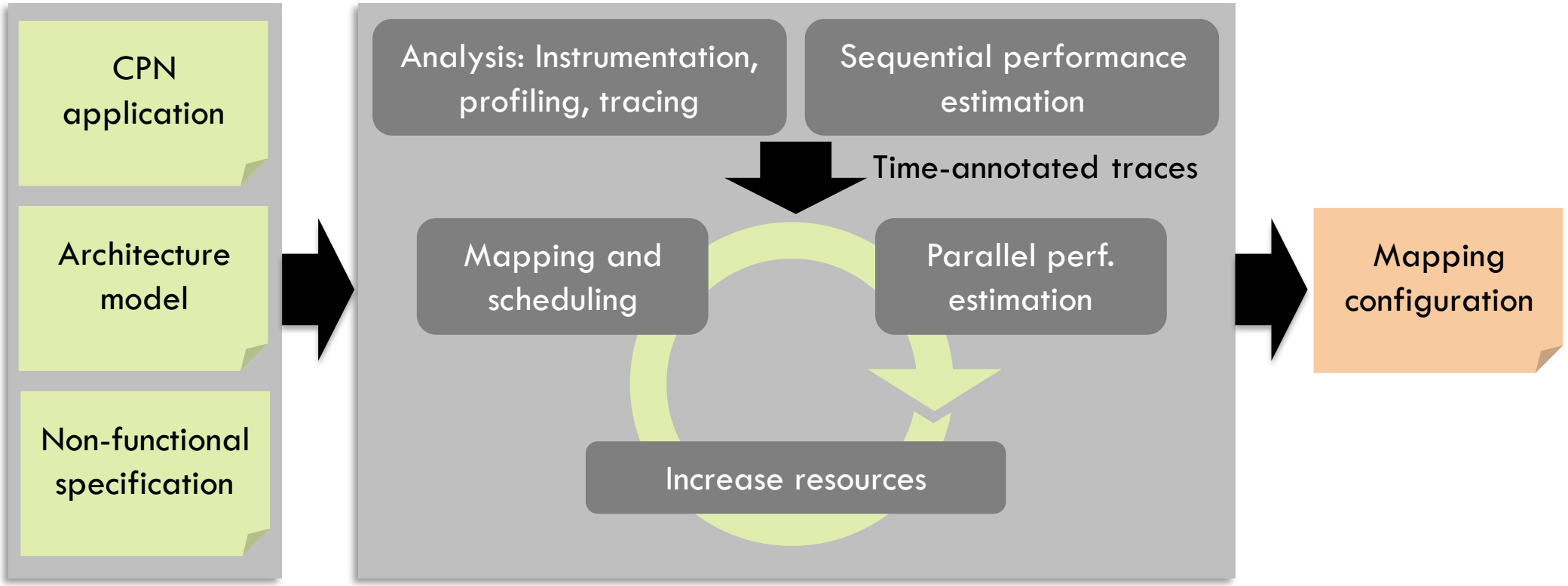
- Extended platform model
  - SW/HW accelerated kernels and their **implementation parameters**


[Castrill10, Castrill11]

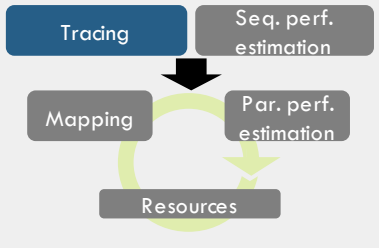
# Outline

- Motivation
- Input specs
- **Analysis and synthesis**
- Code generation and evaluation
- Summary

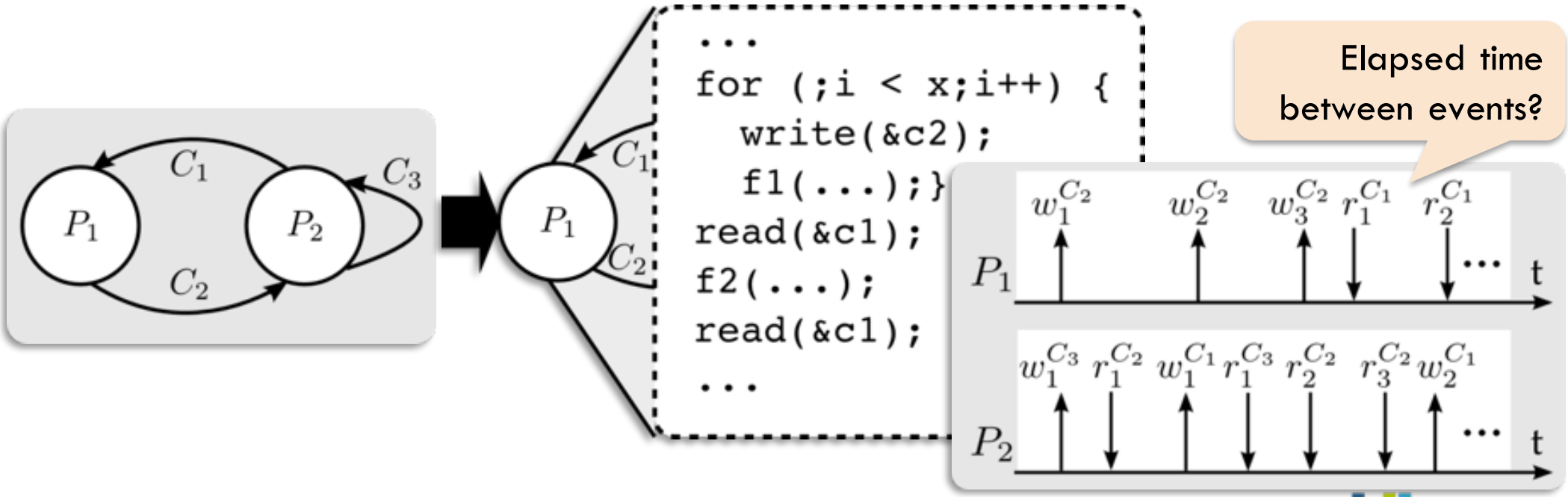
# Analysis and synthesis: Overview



# Tracing: Dealing with dynamic behavior

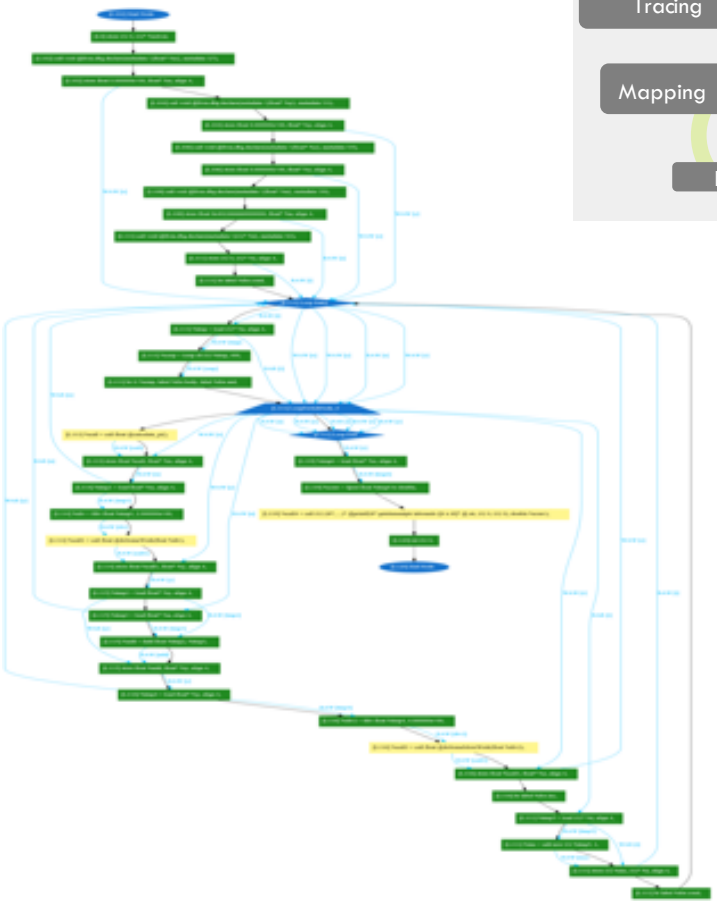


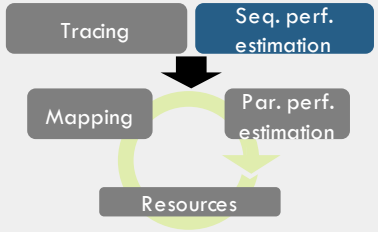

- KPNs do not have firing semantics
- **White model of processes**: source code analysis and tracing
- Tracing: instrumentation, token logging and event recording



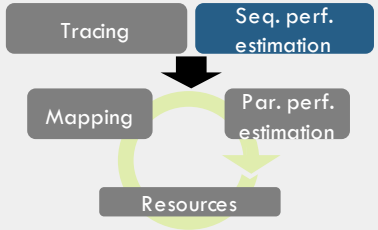
# Sequential performance estimation

- Fine-grained: Sometimes within code basic-blocks
- IR-level instrumentation
  - Cost tables for different architectures
  - Execution count in between events
- Advanced: Emulate effect of target compilers and back annotate to IR



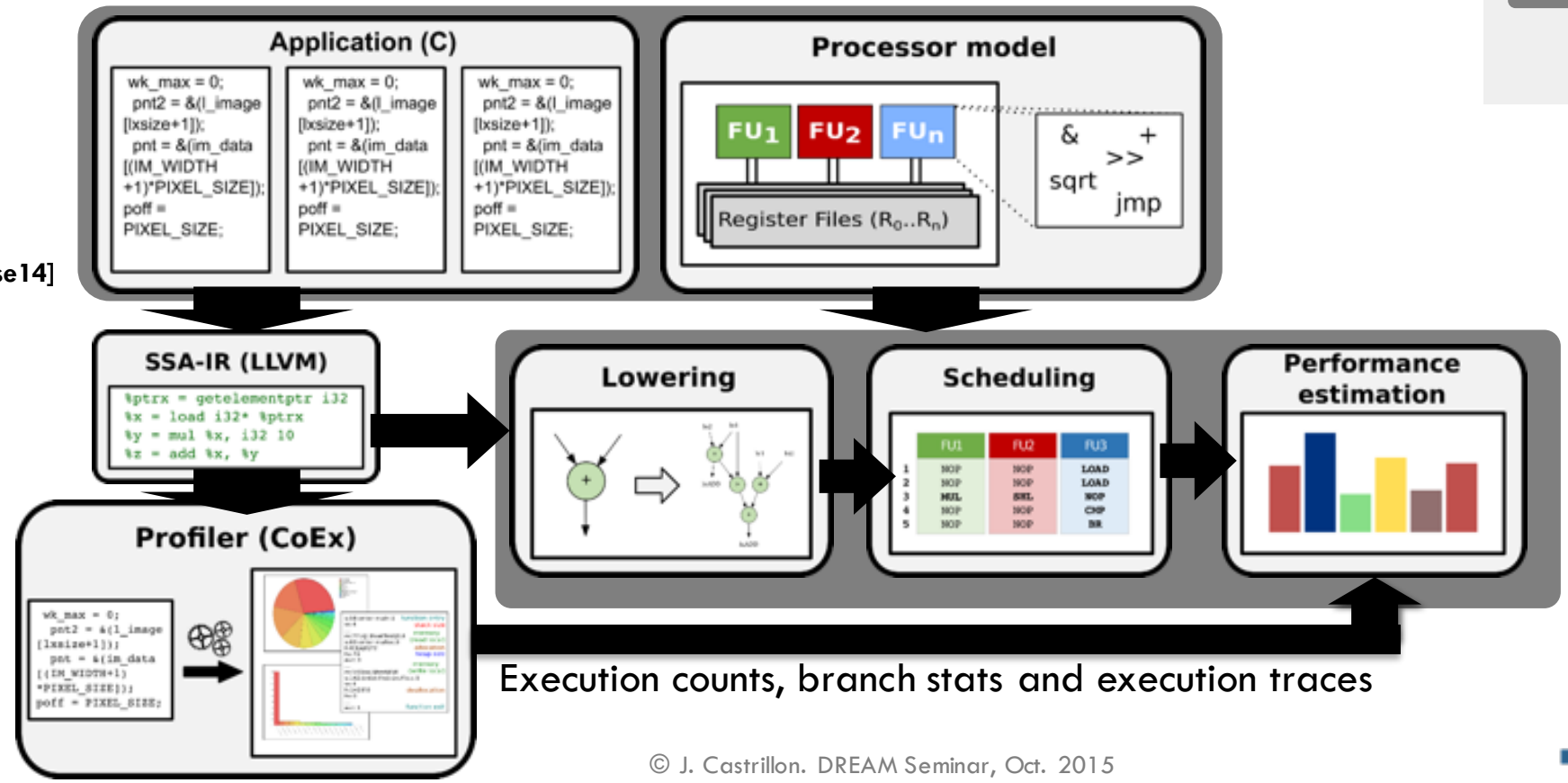


# Sequential performance estimation (2)

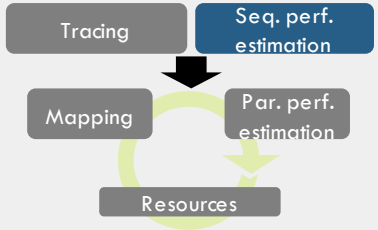



- Processor models

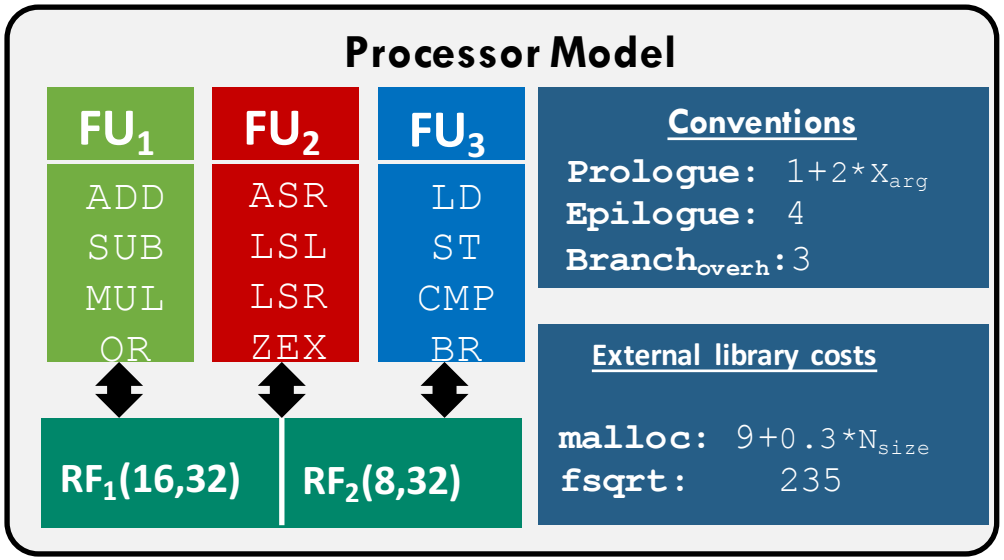
[Eusse14]



# Sequential performance estimation (3)

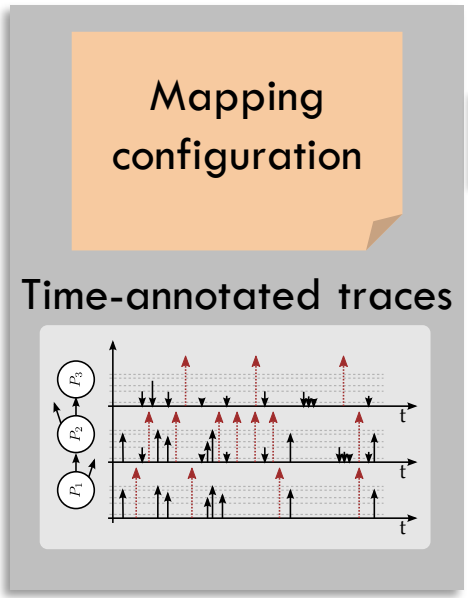


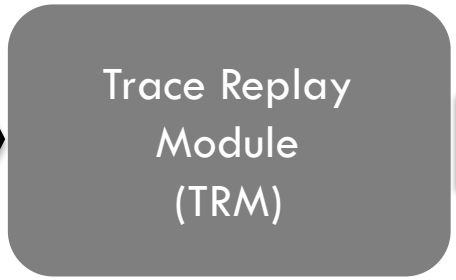

- Abstract models for compiler emulation
  - Resources (functional units, register banks)
  - Operations (pipeline effects, SIMD, addressing, predicated exec.)
  - SW-related costs (calling convention, register spilling, C-lib calls)

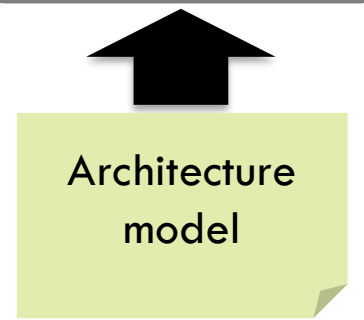


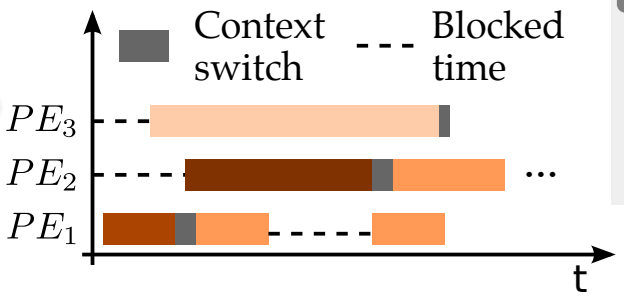

```
<Model name="c674x"> ...
  <FunctionalUnit BW="32" Ctrl="0" name="FU1">
    <Operation bw="32" Pipe="1" lat="2" name="MUL"
           Imm="16" SIMD="1"/>
    <Operation bw="16" Pipe="0" lat="2" name="ADD"
           Imm="16" SIMD="2"/>
  </FunctionalUnit>
  ...
  <RegisterFile name="RF1" size="16" BW="32"/>
  <RegisterFile name="RF2" size="8"  BW="32"/>
  <Perilogue proBase="1" proLin="2" epBase="4" epiLin="0"/>
  <LibraryCosts >
    <Cost name="malloc" base="9"   lin="0.3"/>
    <Cost name="fsqrt"  base="235" lin="0" />
    ...
  </LibraryCosts>
</Model>
```

# Parallel performance estimation

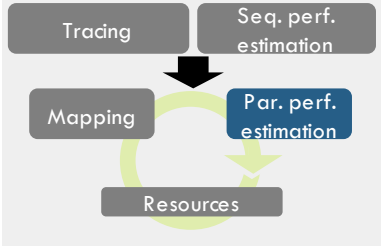


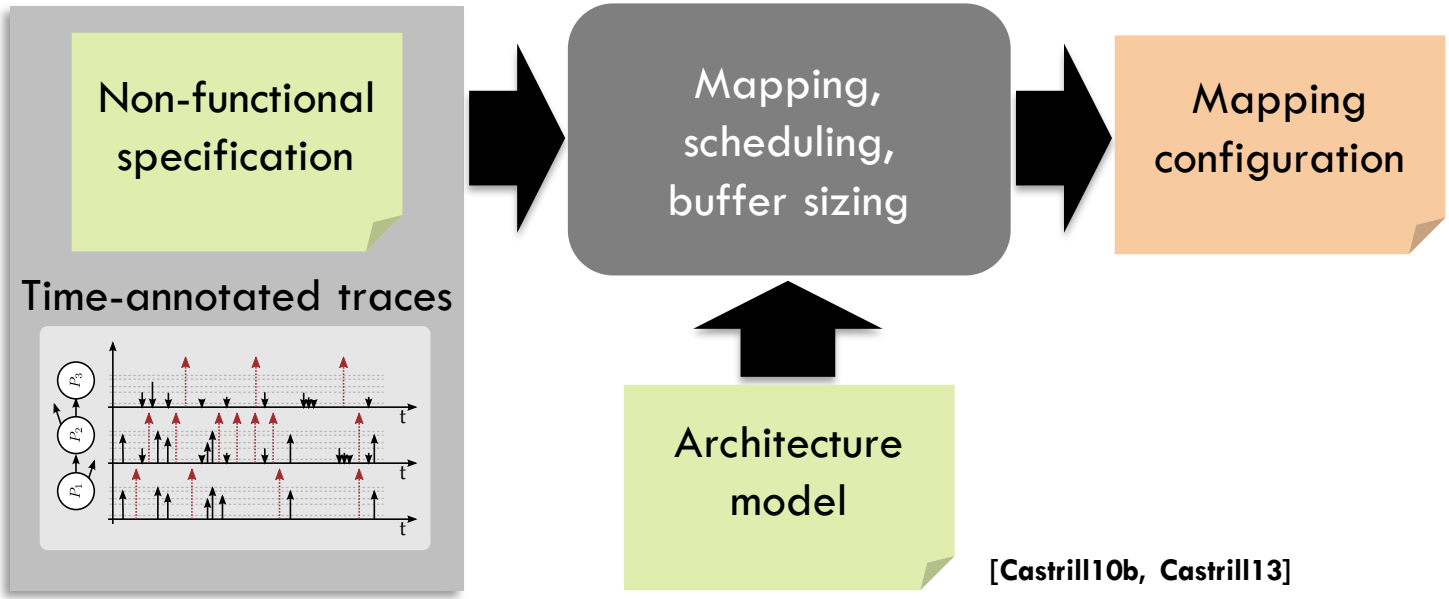




Grantt Chart, platform utilization, channel profiles, ...




- Discrete event simulator to evaluate a solution
  - Replay traces according to mapping
  - Extract costs from architecture file (NoC modeling, context switches, communication)

# Trace-based synthesis



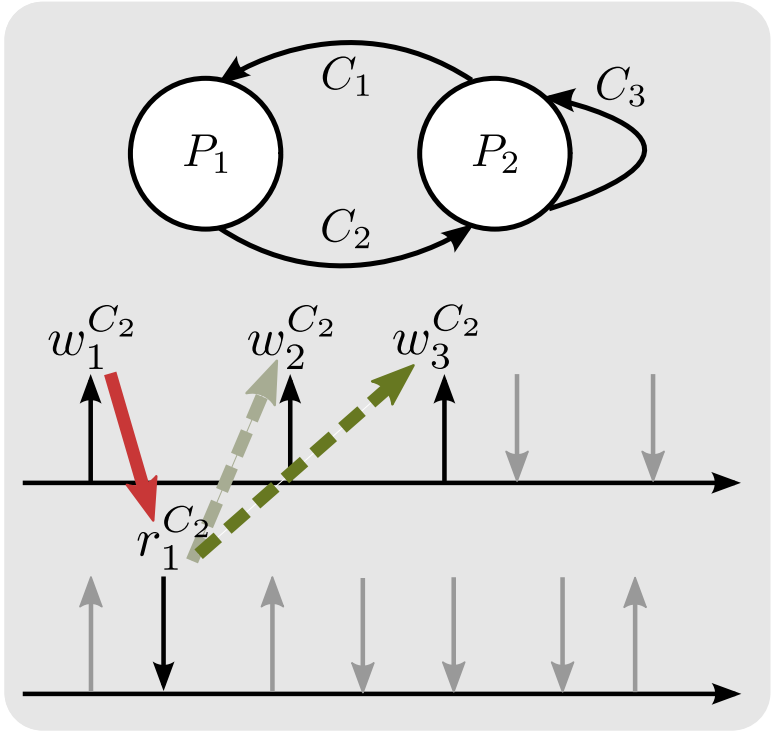


[Castrill10b, Castrill13]

- Synthesis based on code and trace analysis (using simple heuristics)
  - Mapping of processes and channels
  - Scheduling policies
  - Buffer sizing

# Trace-based algorithms


- Event traces can be represented as large dependence graphs



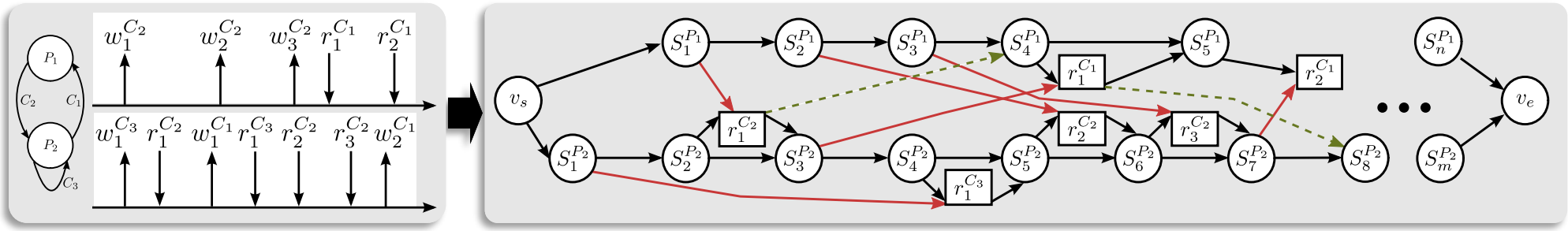

Blocking read

Blocking write, size(chan. 2) = 1

Blocking write, size(chan. 2) = 2

# Trace-based algorithms (2)


- Event traces can be represented as large dependence graphs
- Possible to reason about
  - Channel sizes and memory allocation
  - Mapping and scheduling onto heterogeneous processors




size(chan. 2) = 2, size(chan. 1,3) = 1

# Dealing with heterogeneity: group-based mapping (GBM)

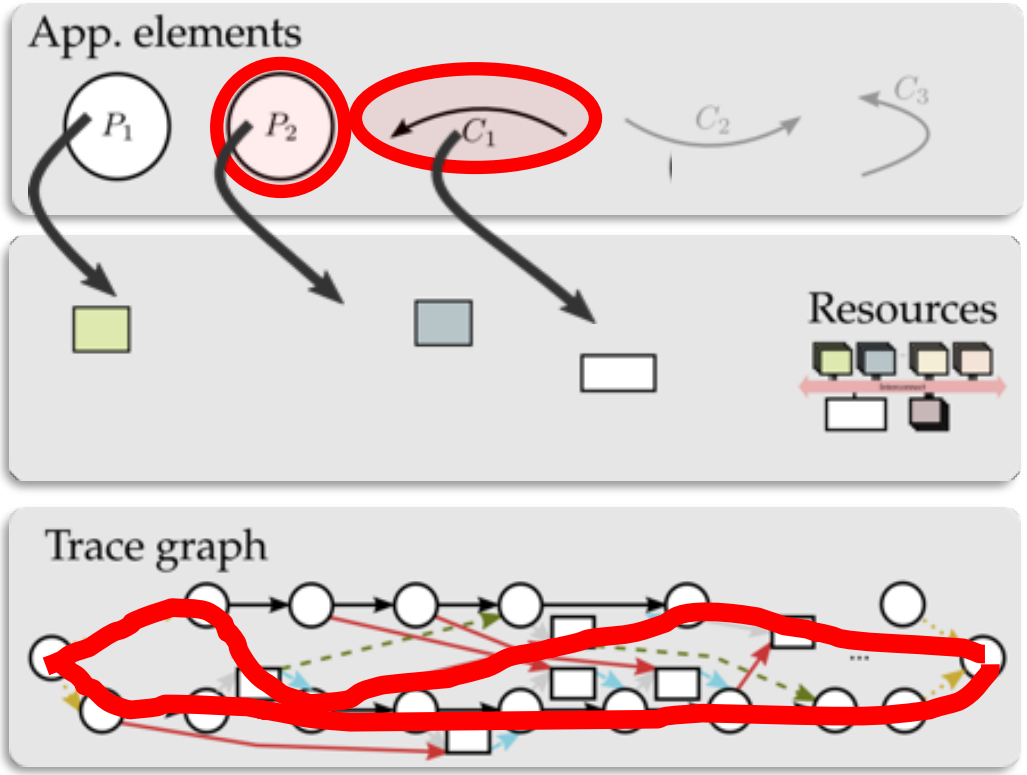
1) Initialize: All to all

2) Select element: Trace graph critical path

3) Reduce group

4) Assess & propagate

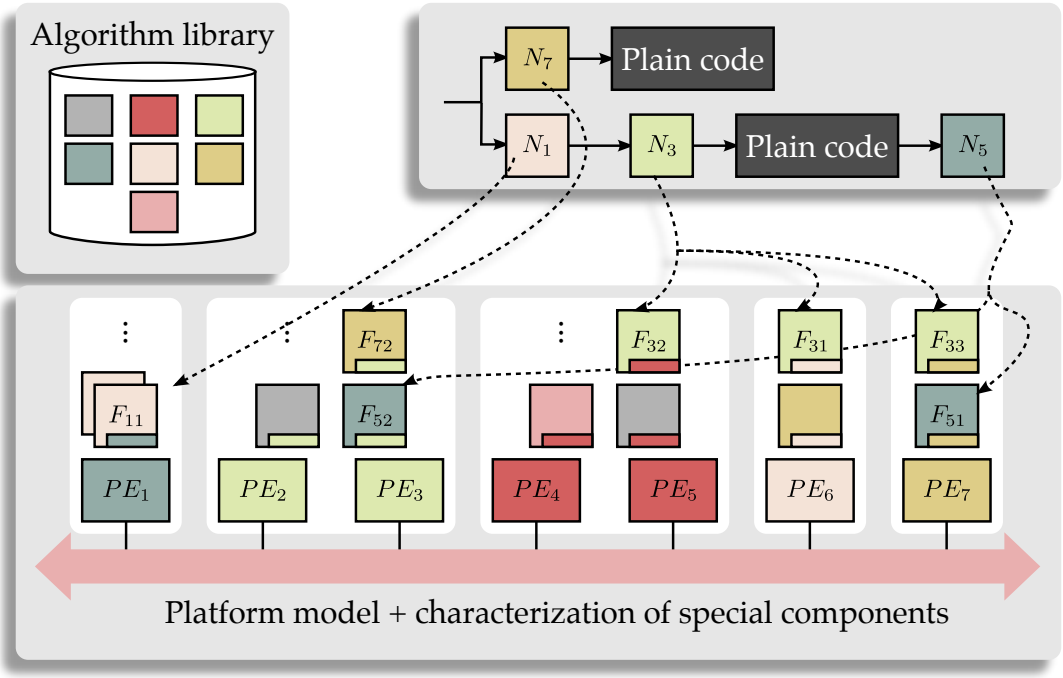
5) Quasi-homogeneous




[Castrill12]

# Mapping for HW accelerators


- Not only mapping but also configuration
  - Match algorithmic parameters with implementation parameters
  - Adjust synchronization and communication protocols



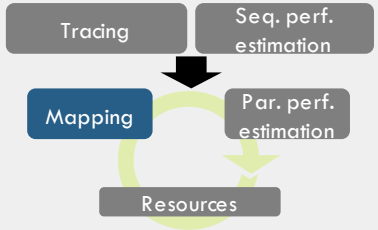

```
<parameter name="bitwidth"><value>32.0</value></parameter>
<parameter name="points"><values list="32 64 128 256" /></parameter>
<property name="latency"><function>16*points*100</function></property>
<input name="fft_in"><port>input</port>
<DataType representation="fixed_point" format="Q31" DataWidth="32" />
<Interface type="buffer_flag_1of2">
 <val name="size" val="8" /><val name="stride" /> <val name="cnt" val="64" />
 <val name="fsize" val="4" /></Interface>
<Interface type="buffer_flag_2of2">
 <addr name="addr" pool="in" min="0x04100000" max="0x041F0000" gran="8"
       size="8" stride="fft_in__stride" cnt="64"/>
 <addr name="faddr" pool="in" min="0x04100000" max="0x041F0000" gran="4"
       size="4" cnt="1"/></Interface></input>
```

**[Castrill10, Castrill11]**

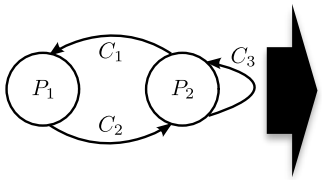
N: Algorithmic actors

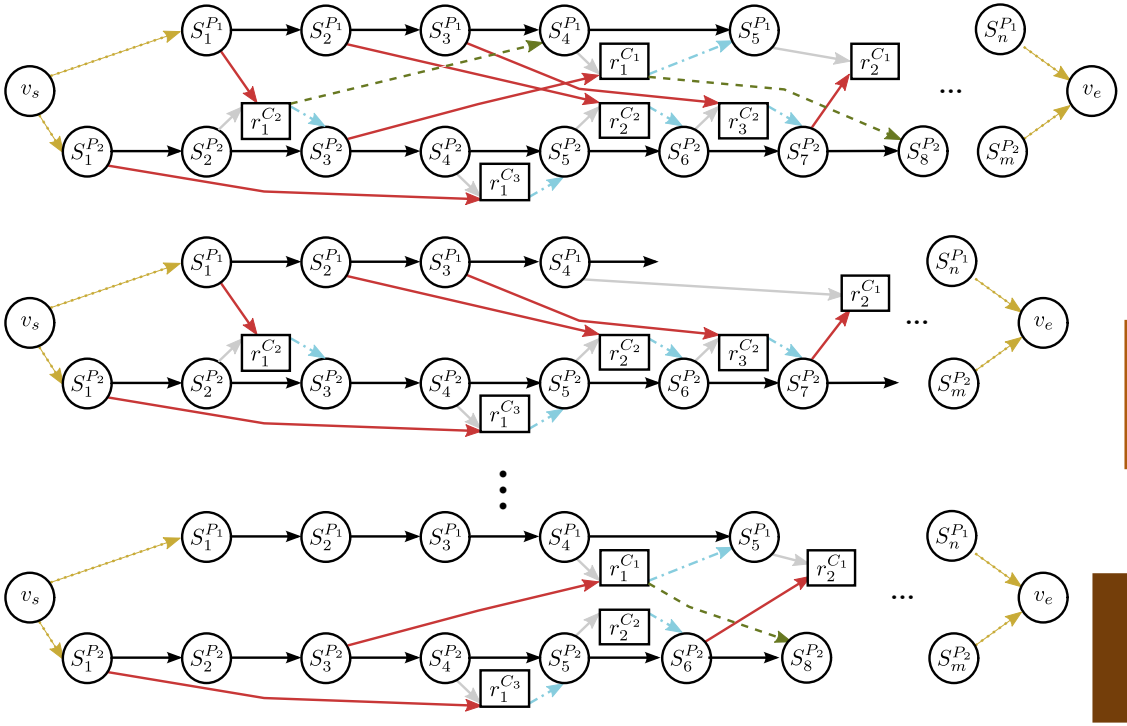
F: Existing implementation in target platform

# Multiple traces




- Different input → different behavior (traces)
  - Characterize behaviors and impact on mapping performance



Mapping configuration

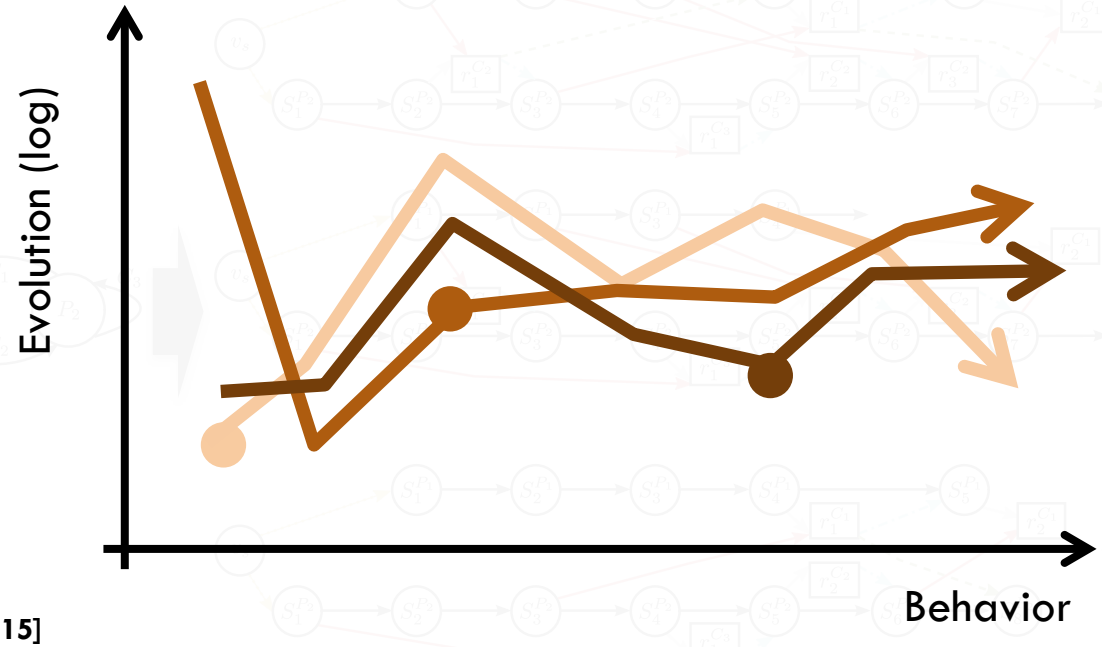
Mapping configuration

Mapping configuration

[Goens15]

# Multiple traces (2)


- Different input → different behavior (traces)
  - Characterize behaviors and impact on mapping performance



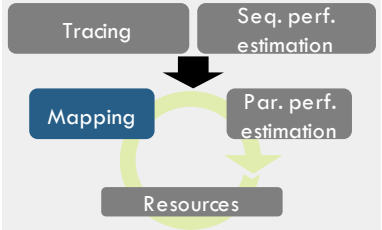

[Goens15]

Mapping configuration

Mapping configuration

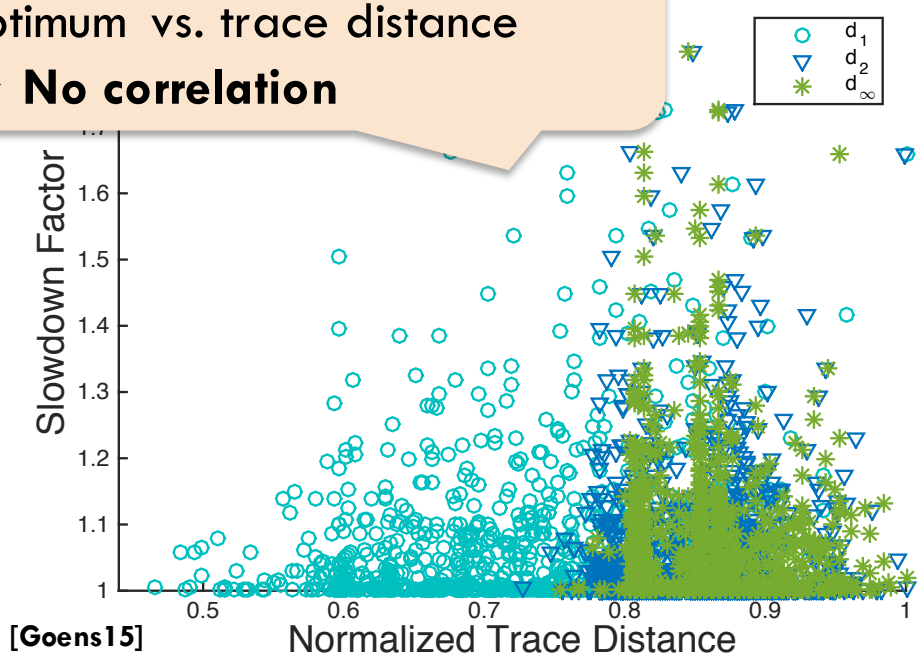
Mapping configuration

# Multiple traces (3)



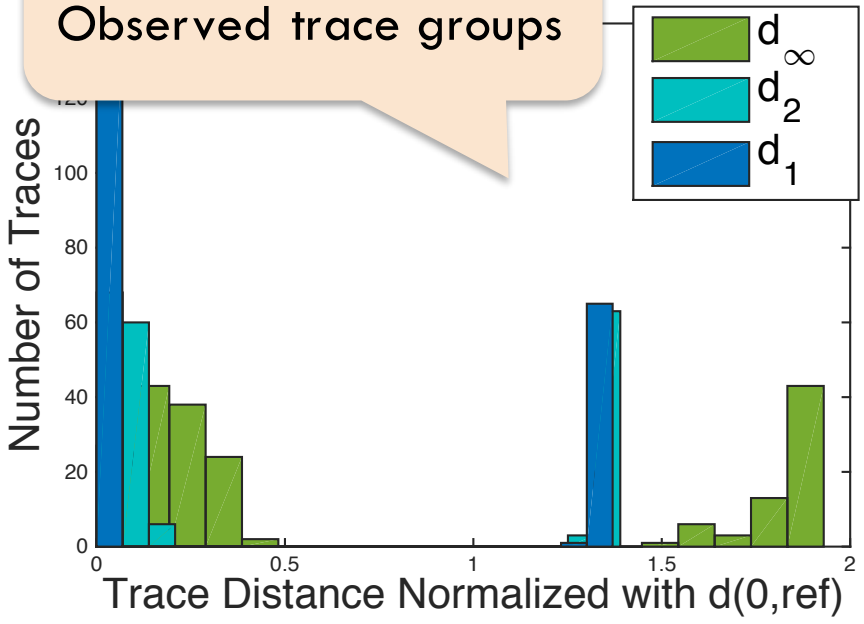

- Behavior difference as metric in trace/history monoid

Random KPNs: Slow down w.r.t. optimum vs. trace distance
→ **No correlation**

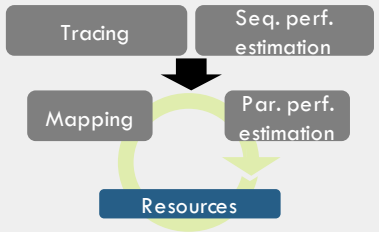



[Goens15]

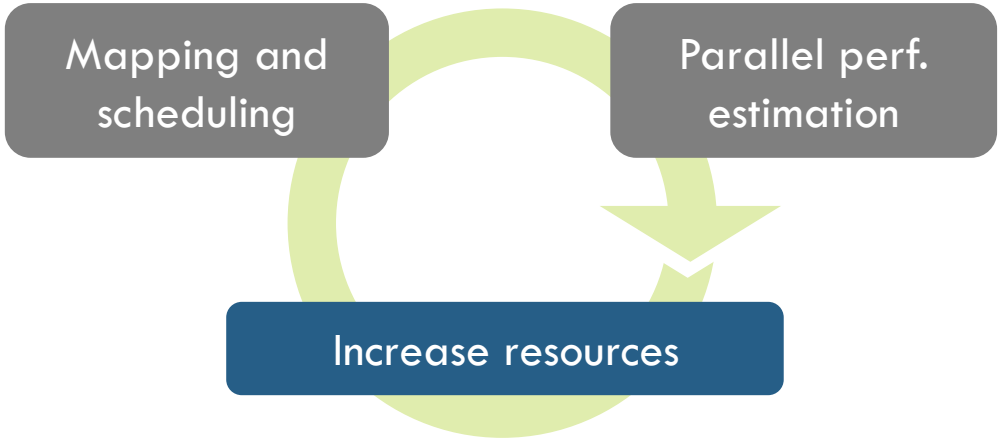
JPEG application: Observed trace groups



# Increasing resources

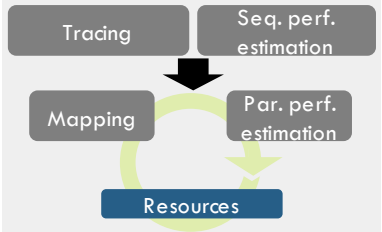



- Add resources to the synthesis until constraints are met

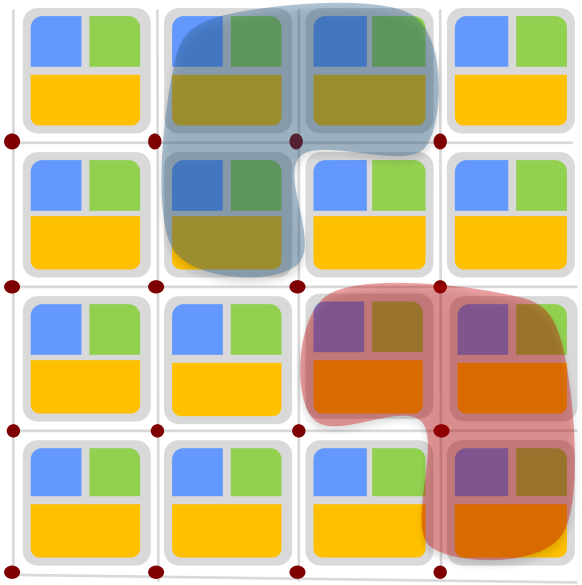



- Add processors and memories
  - Easy for homogeneous platforms
  - Non trivial for heterogeneous platforms

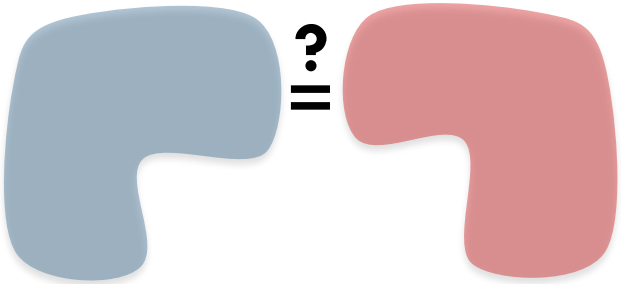
# Increasing resources: Exploit symmetries




- Identify mapping **equivalent classes** due to HW **symmetries**



[Goens15]

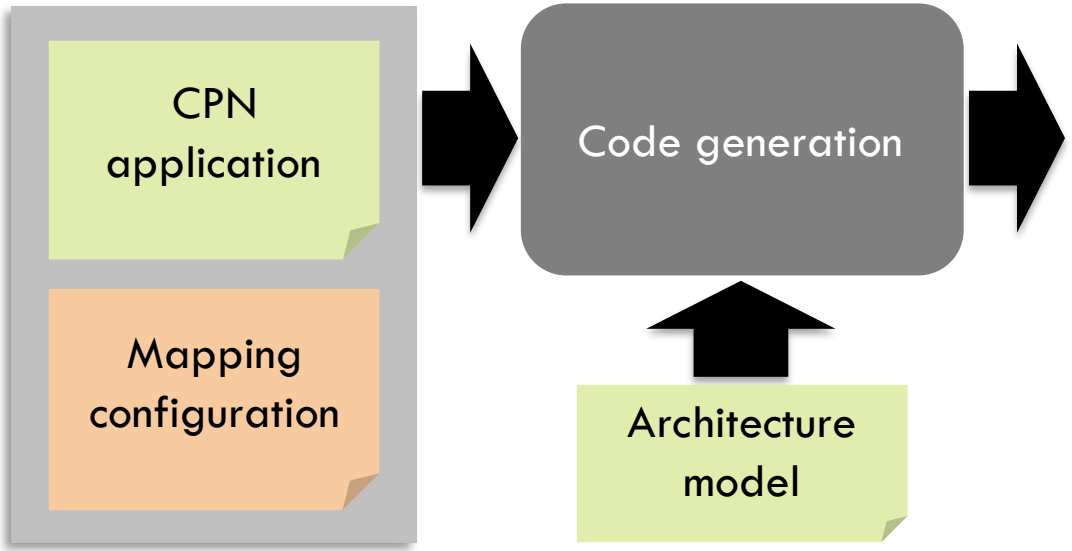



- Do not evaluate equivalent mappings
- Reduce search space when adding resources
- Multiple traces (revisit)
  - Random traces: **5 out of 83** classes account for **50%** of all optimal mappings
- Application to multi-application analysis

# Outline

- Motivation
- Input specs
- Analysis and synthesis
- **Code generation and evaluation**
- Summary

# Code generation



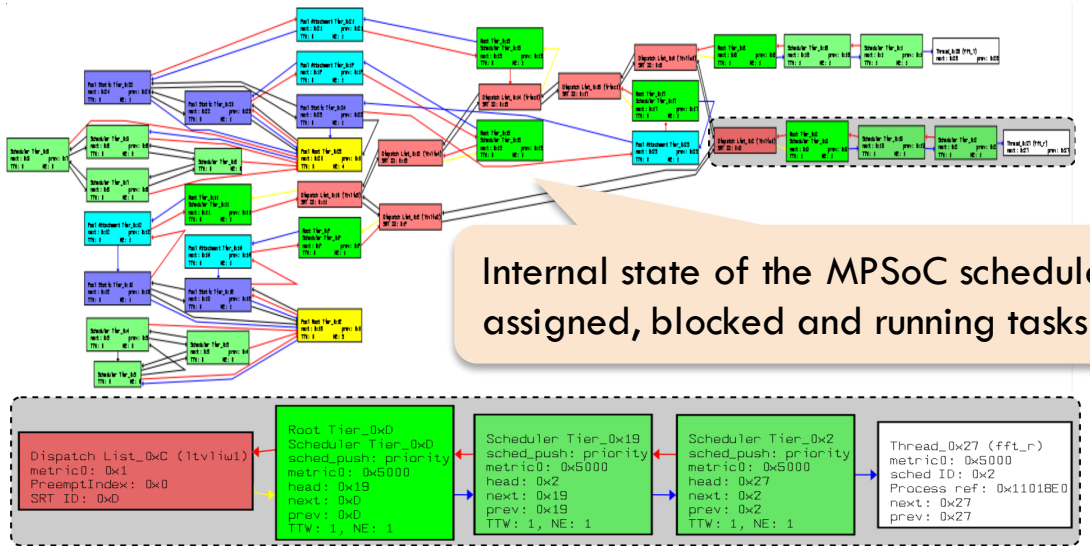

```
PNargs_ifft_r.ID = 6U;
   PNargs_ifft_r.PNchannel_freq_coef = filtered_coef_right;
   PNargs_ifft_r.PNnum_freq_coef = 0U;
   PNargs_ifft_r.PNchannel_time_coef = sink_right;
   PNargs_ifft_r.channel = 1;
   sink_left = IPCllmrf_open(3, 1, 1);
   sink_right = IPCllmrf_open(7, 1, 1);
   PNargs_sink.ID = 7U;
   PNargs_sink.PNchannel_in_left = sink_left;
   PNargs_sink.PNnum_in_left = 0U;
   PNargs_sink.PNchannel_in_right = sink_right;
   PNargs_sink.PNnum_in_right = 0U;
   taskParams.arg0 = (xdc_UArg)&PNargs_src;
   taskParams.priority = 1;
   ...
```

Debugging info

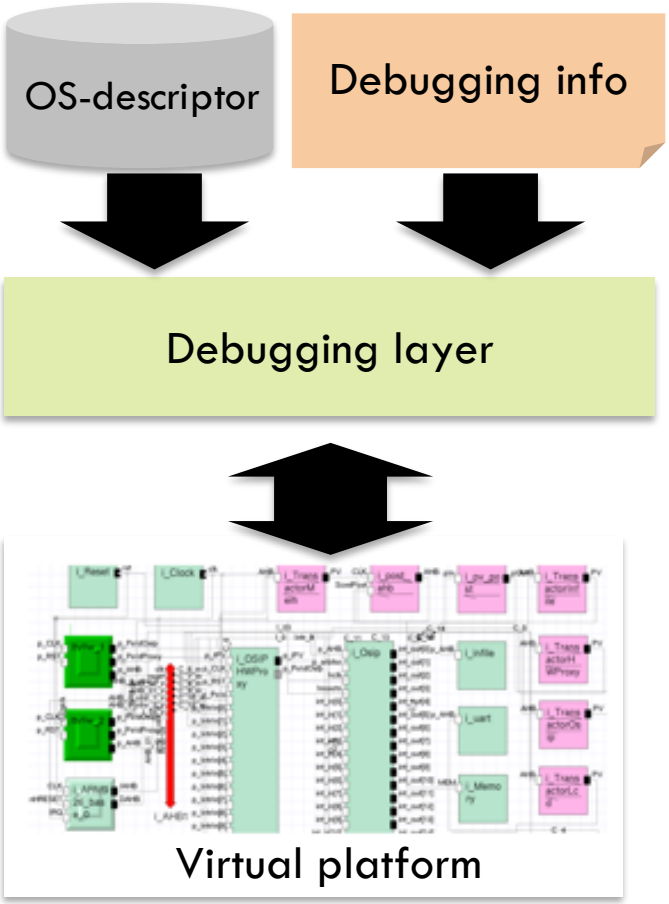
- Take mapping configuration and generate code accordingly
- From architecture model: APIs, configuration parameters, …
- Sample targets: experimental heterogenous systems, TI (Keysonte, TDA3x), Parallella/Ephiphany, ARM-based (Exynos, Snapdragon)

# Debugging with virtual platforms

- Interactive debugging
  - Get snapshots of the system state
  - Full system stop
  - Track progress irrespective of mapping

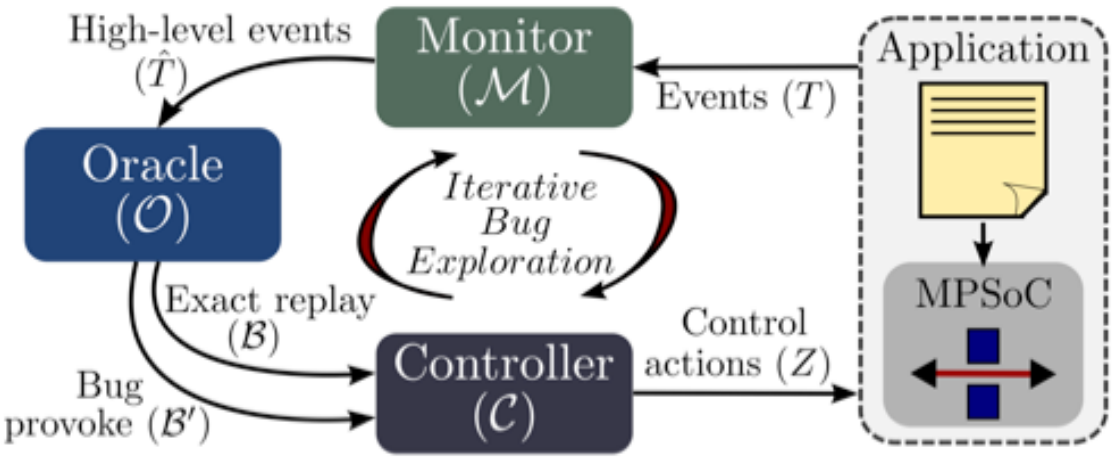


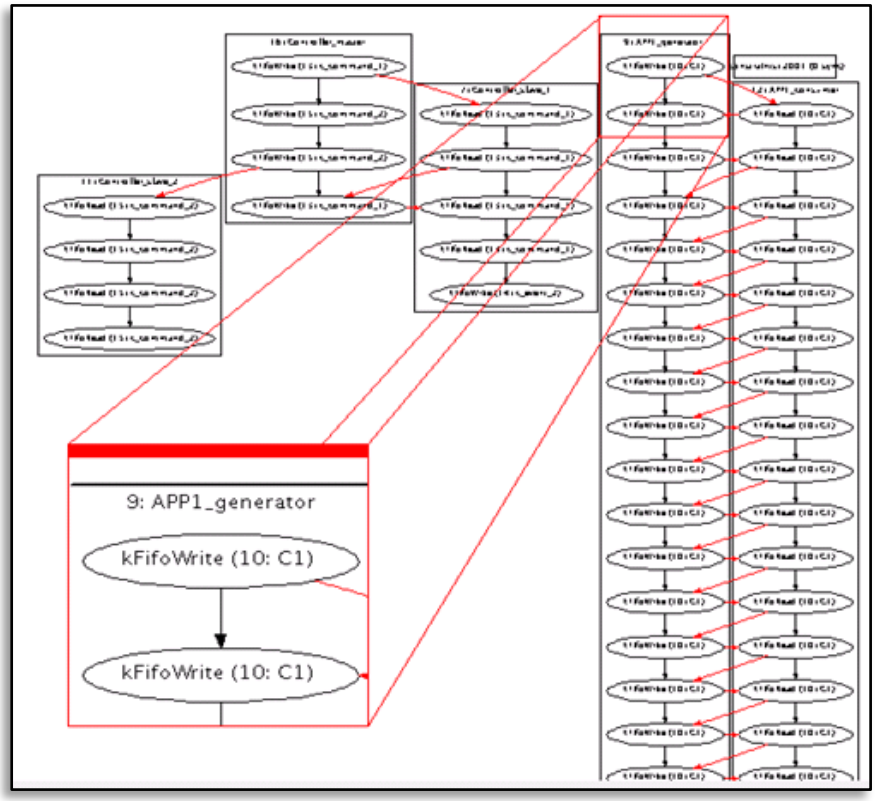

[Castrill11]



# Debugging with virtual platforms (2)

- Deterministic replay and automatic bug exploration

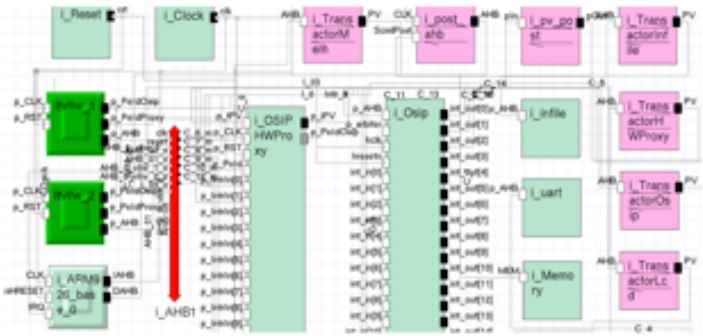



**[Murillo14]**

# Evaluation and results

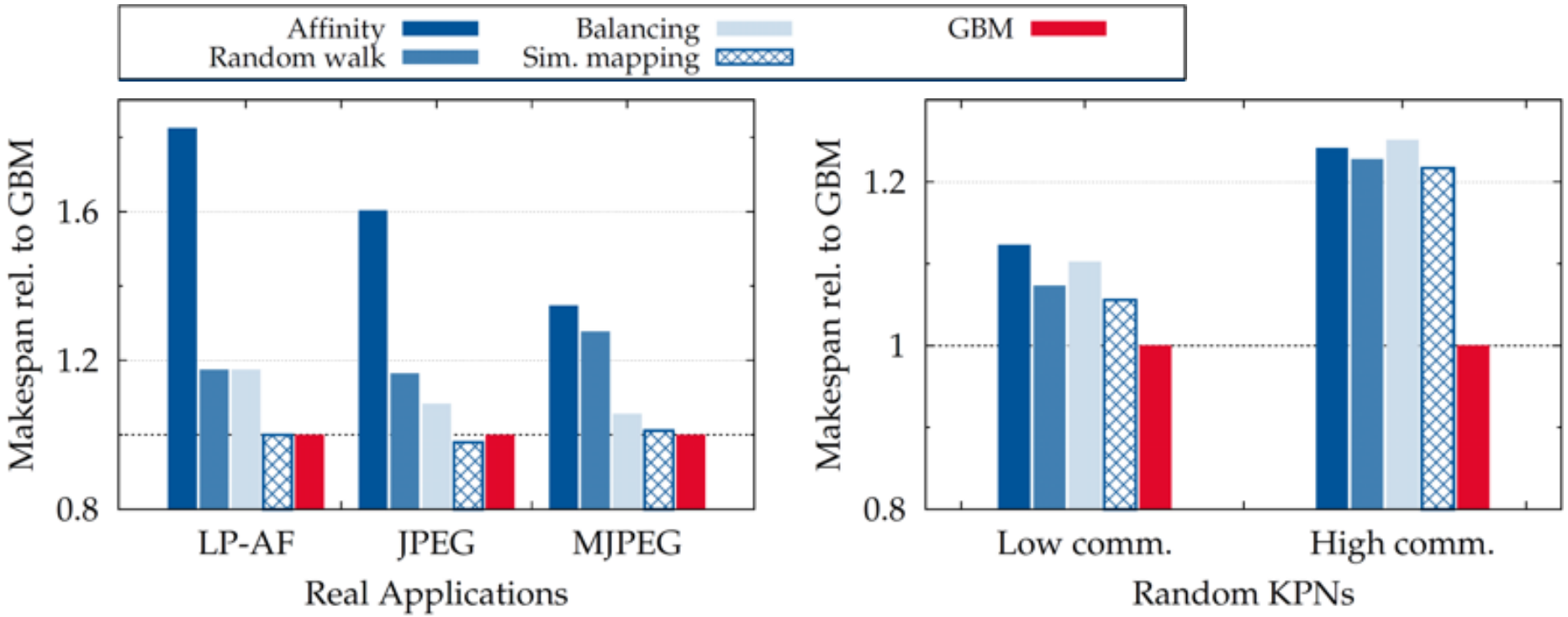
- Virtual platforms: SystemC models of full systems
  - Explore heterogeneous architectures
  - Easier to integrated state-of-the-art accelerators
  - Configurable accuracy



- Real platforms for validation
  - Speedup on commercial platforms
  - Code generation against vendor stacks

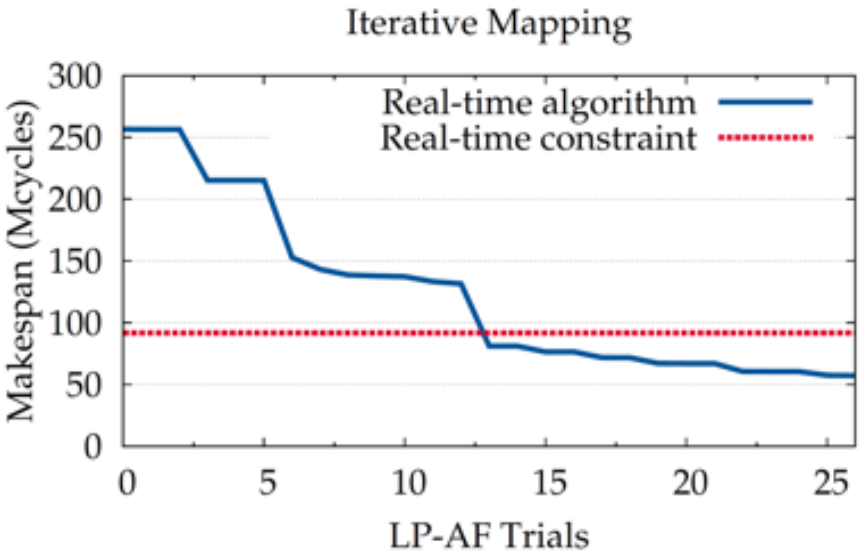
# Example: multi-media applications

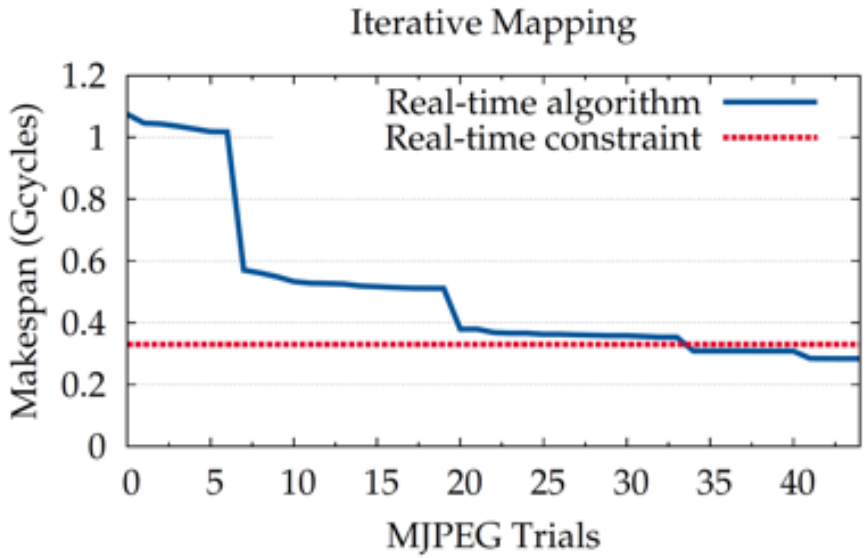
- Platform: 2 RISCs, 4 VLIW, 7 Memories



# Example: multi-media applications (2)

- Dealing with real-time constraints



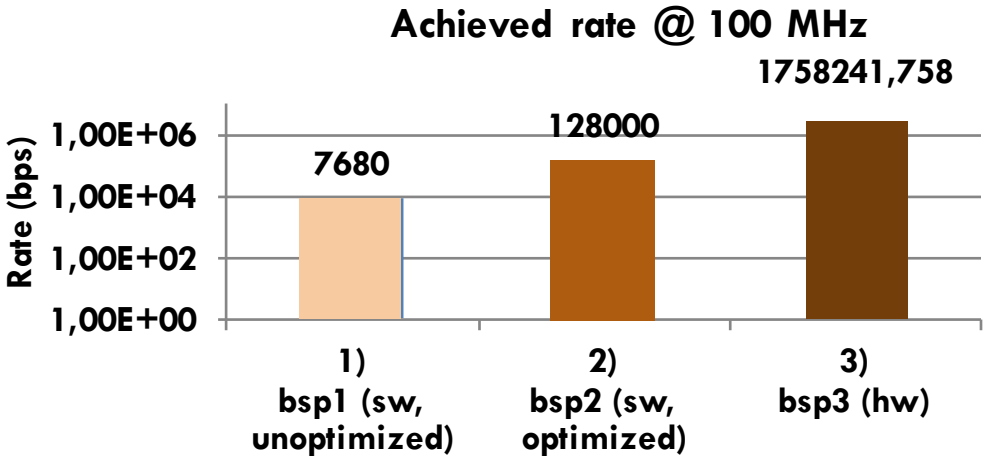



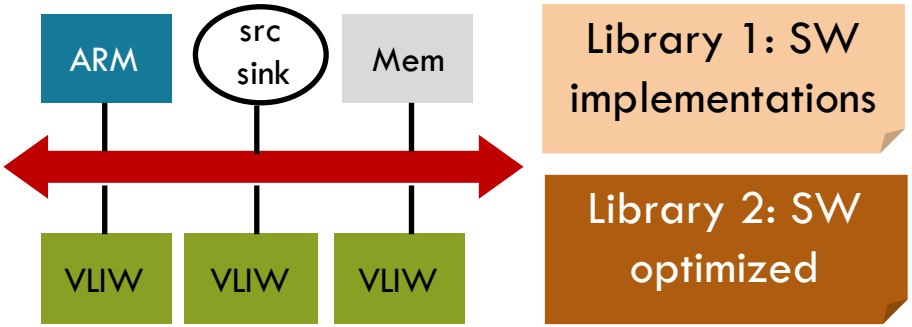

Tool: ~1 min. for LP-AF, ~7 min. for MJPEG — ~10 min.

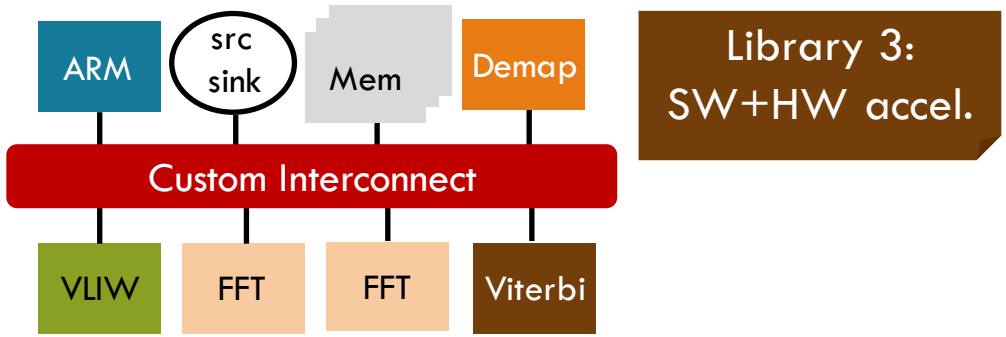
Sim.: ~6 days for LP-AF, ~24 for MJPEG — ~10 days (~x$10^3$)

# Example: With HW acceleration

- Application: MIMO OFDM receiver
- Hardware
  - Platform 1: Baseline software
  - Platform 2: Optimized software
  - Platform 3: Optimized SW + HW



# Manual vs. Automatic: TI Keystone

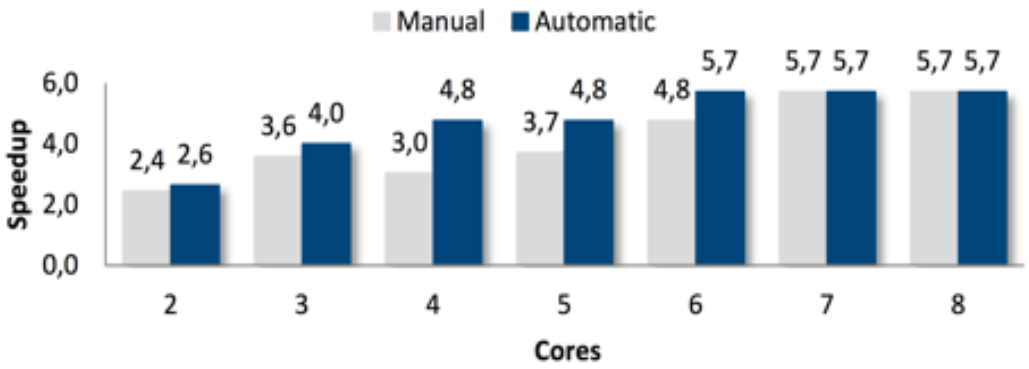
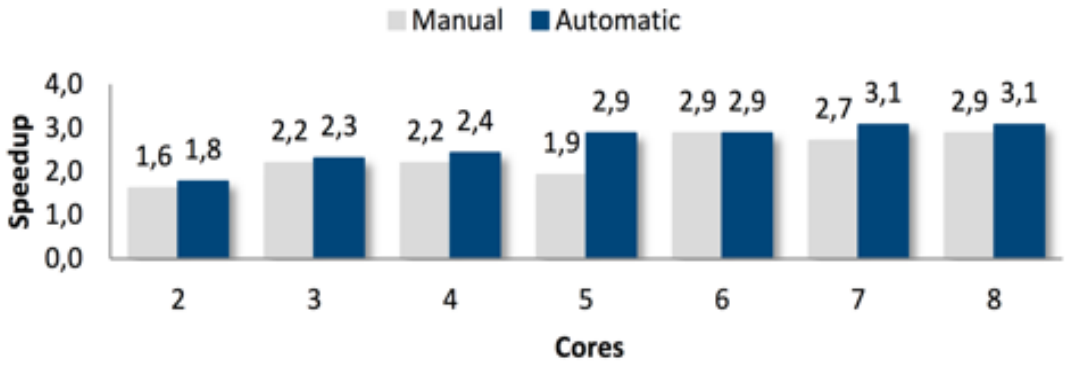
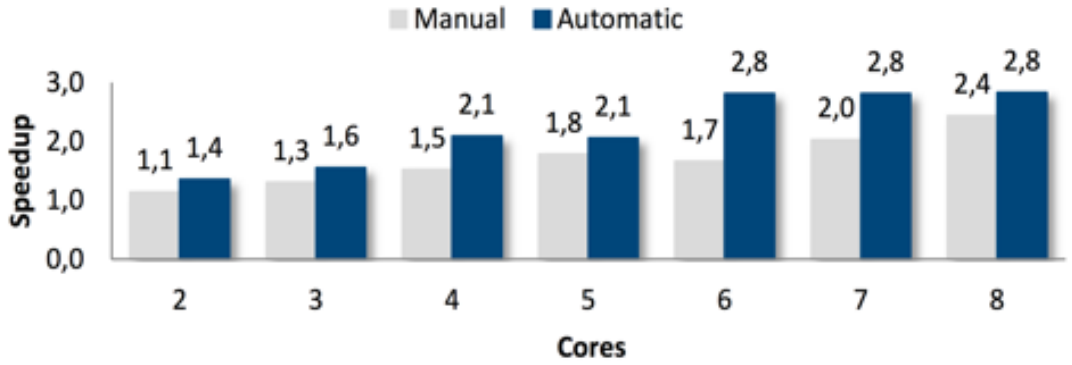



**Image processing**



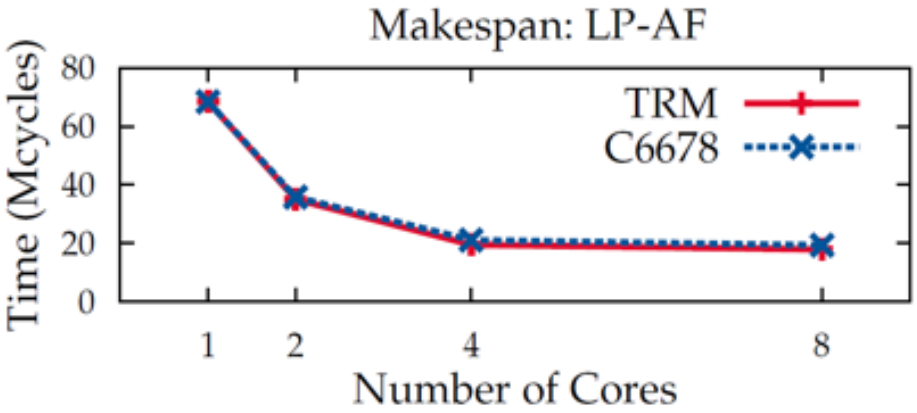

**Audio filtering application**

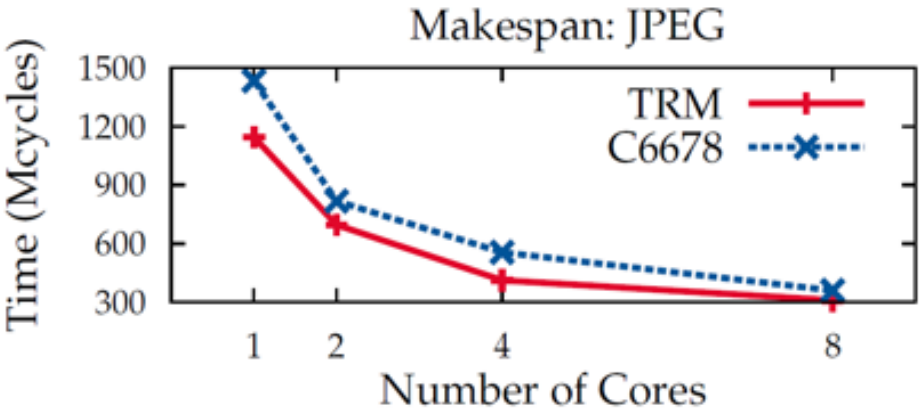


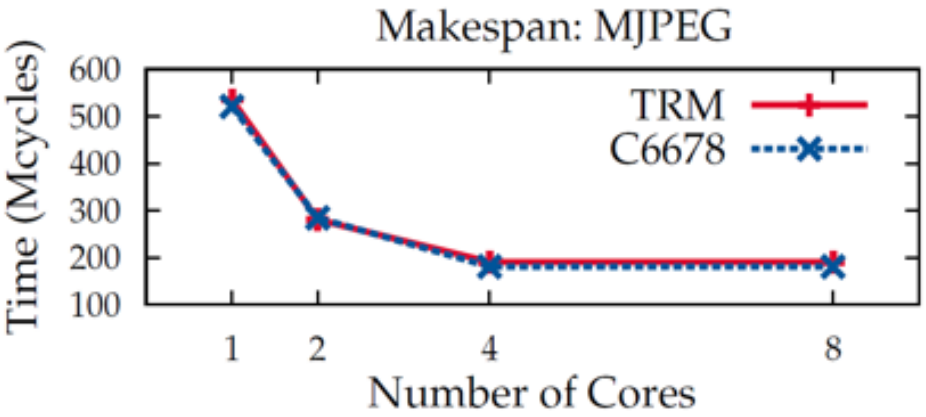

**LTE digital receiver**

[Aguilar14]

# TRM vs. Actual execution: TI Keystone




Makespan error < 20% (Avg. 7%)

Speedup error < 8% (Avg. 5%)

# Outline

- Motivation
- Input specs
- Analysis and synthesis
- Code generation and evaluation
- **Summary**

# Summary

- Tool flow for mapping KPN applications
  - CPN language: a language for KPNs close to C
  - Analysis and synthesis based on traces
  - Extensions for: multiple traces and algorithmic descriptions
  - Backends for multiple platforms (bus and NoC-based)
- Current and future work
  - More on static code analysis
  - Continue on multiple-traces and symmetries
  - Applying to other domains (server applications)

# References


**[Trommer15]** Trommer, et al., "Functionality-Enhanced Logic Gate Design Enabled by Symmetrical Reconfigurable Silicon Nanowire Transistors", IEEE Trans on in Nanotechnology, pp.689-698, July 2015

**[Voigt14]** Andreas Voigt, et al., "Towards Computation with Microchemomechanical Systems", In Journal of Foundations of Computer Science, vol 25, 2014

**[Castrill14]** J. Castrillon , et al., "Programming Heterogeneous MPSoCs: Tool Flows to Close the Software Productivity Gap", Springer, 2014

**[EETimes11]** http://www.eetimes.com/document.asp?doc_id=1259446

**[Sheng14]** W. Sheng, et al., "A compiler infrastructure for embedded heterogeneous MPSoCs", Parallel Comput. 40, 2, 51-68, 2014

**[Oden13]** M. Odendahl, et al., "Split-cost communication model for improved MPSoC application mapping", In International Symposium on System on Chip pp. 1-8, 2013

**[Arnold13]** O. Arnold, et al. "Tomahawk - Parallelism and Heterogeneity in Communications Signal Processing MPSoCs". *TECS, 2013*

**[Castrill10]** J. Castrillon, , et al., "Component-based waveform development: The nucleus tool flow for efficient and portable SDR," Wireless Innovation Conference and Product Exposition (SDR), 2010

**[Castrill11]** J. Castrillon, , et al., "Component-based waveform development: The nucleus tool flow for efficient and portable software defined radio", Analog Integrated Circuits and Signal Processing, vol. 69, no. 2–3, pp. 173–190, 2011

**[Eusse14]** J.F. Eusse, , et al., "Pre-architectural performance estimation for ASIP design based on abstract processor models," In SAMOS 2014 pp.133-140, 2014

**[Castrill10b]** J. Castrillon, et al., "Trace-based KPN composability analysis for mapping simultaneous applications to MPSoC platforms", In DATE 2010, pp. 753-758

**[Castrill13]** J. Castrillon, et al., "MAPS: Mapping concurrent dataflow applications to heterogeneous MPSoCs," IEEE Transactions on Industrial Informatics, vol. 9, 2013

**[Castrill12]** J. Castrillon, et al. , "Communication-aware mapping of KPN applications onto heterogeneous MPSoCs," in DAC 2012

**[Goens15]** A. Goens, et al., "Analysis of Process Traces for Mapping Dynamic KPN Applications to MPSoCs", In IFIP International Embedded Systems Symposium (IESS), 2015, Foz do Iguaçu, Brazil (Accepted for publication), 2015

**[Castrill11]** J. Castrillon, et al., "Backend for virtual platforms with hardware scheduler in the MAPS framework", In LASCAS 2011 pp. 1-4, 2011

**[Murillo14]** L. G. Murillo, et al., "Automatic detection of concurrency bugs through event ordering constraints", In DATE 2014, pp. 1-6, 2014

**[Aguilar14]** M. Aguilar, et al., "Improving performance and productivity for software development on TI Multicore DSP platforms," in Education and Research Conference (EDERC), 2014 6th European Embedded Design in , vol., no., pp.31-35, 11-12 Sept. 2014

# Thanks!
# Questions?