Technische Universität Braunschweig

TUHH
Technische Universität Hamburg

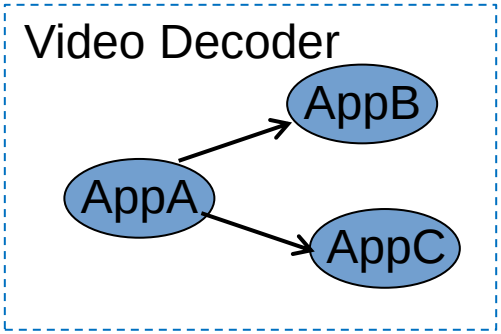
Modeling Accesses to Shared Memories in Multi-Processor Systems-on-Chip (MPSoCs)

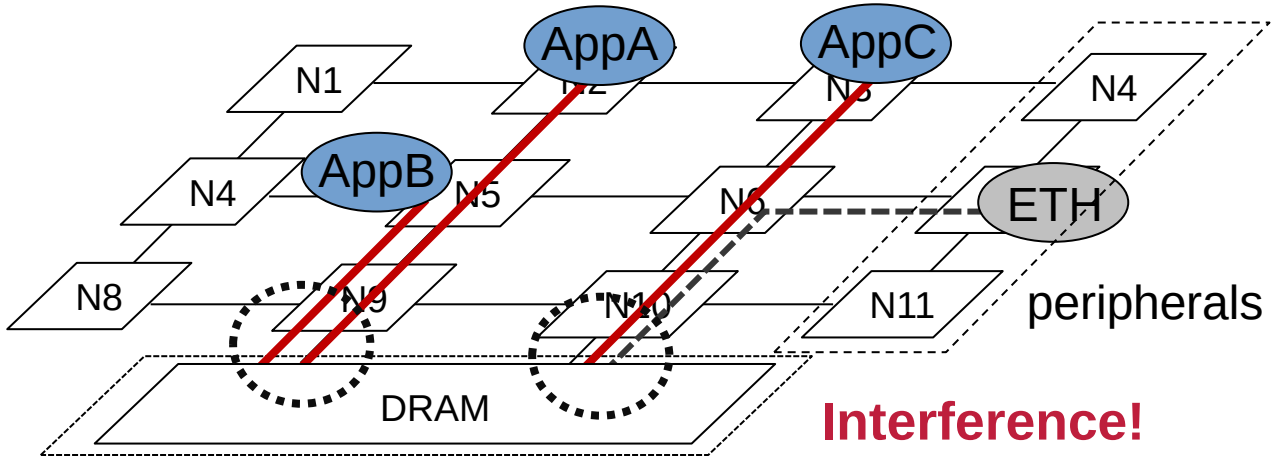
Adam Kostrzewa, Selma Saidi, Rolf Ernst,
{kostrzewa, ernst}@ida.ing.tu-bs.de, Selma.Saidi@tuhh.de

WATERS Workshop | Barcelona, Spain | 03 July 2018

Multiprocessor Systems-on-Chip

- allow **integration** of many components – **heterogeneous and dynamic**
- result → BEs and HRTs **share NoC resources e.g.** links and buffers
- **safety standards** in case of shared resources require
  - **functional independence -** still allow application communication
  - **timing independence** – still allow efficient scheduling





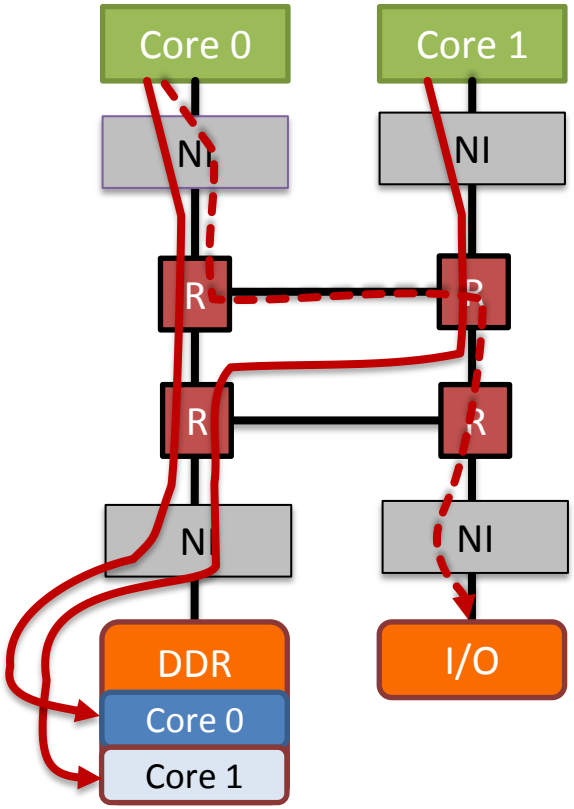
Goals for Time Predictability

- timing and/or safety relevant functions require predictable upper-bound on timing
- but this must be performance efficient
  - counter example: Time-Triggered Architectures
- performance efficient timing predictability is more complex
  - application requirements vary during execution
  - predictability requires powerful verification
- higher safety-levels require verification by formal methods
  – timing analysis
  - analysis requires predictable architectures and application behavior
  - but only for safety-critical application functions
    - easy to provide overly pessimistic settings for BE senders

Problems with MPSoCs

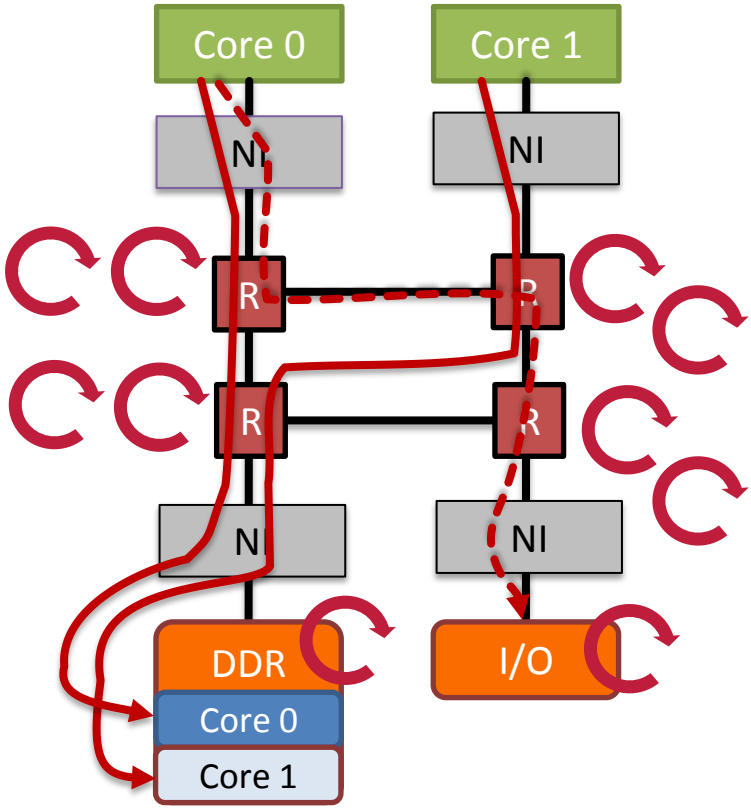
Problem 1: Heterogeneous Traffic

- memory → main shared resource
  - cache lines – short, sporadic
  - DMA transfers – long, regular
- heterogeneous communications
  - different applications – BEs and real-time
  - control, streaming
  - on-chip and off-chip traffic
- Consequences:
  - Dynamics leads to pessimistic guarantees
  - High complexity of analysis and hard verification



Problems with MPSoCs

- Problem 2: Heterogeneous Components
  - Memory Controllers and Peripherals
  - Routers – at least one or more
- Consequences:
  - Coupling of scheduling leads to pessimistic guarantees
  - High complexity of analysis and hard verification



Analysis must deal efficiently with these challenges!

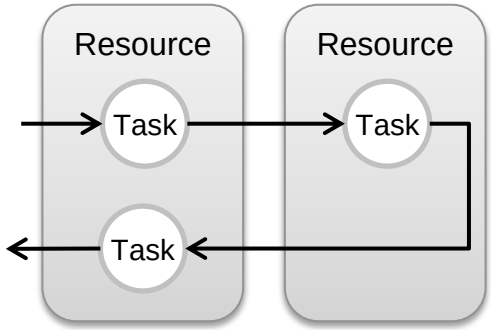
Presentation Outline

- Motivation MPSoC
- **MPSoCs with CPA**
- **Analysis**
  - **Arbitration at the NoC level**
  - **Interface to SDRAMs**
- **Results**
- **Summary**

Compositional Performance Analysis

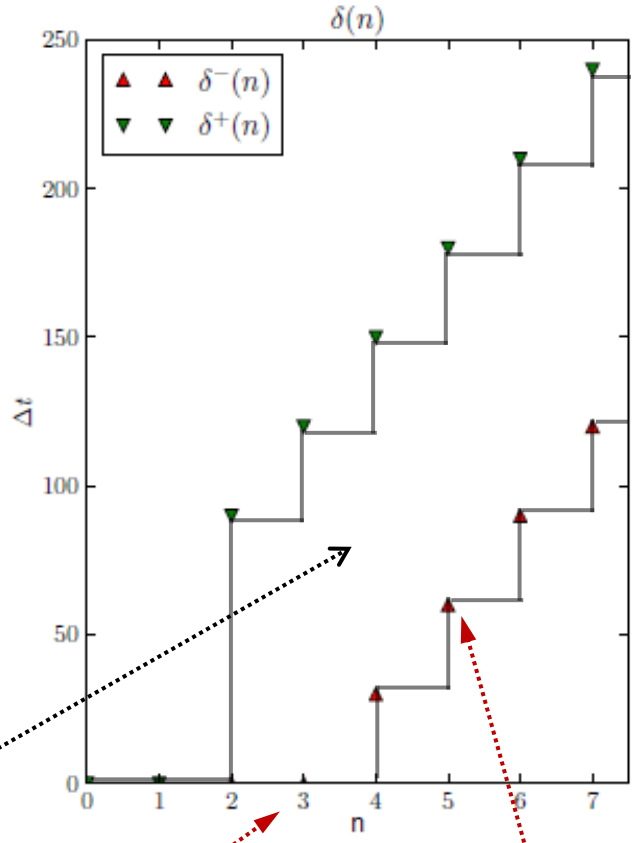
- originally used for scheduling analysis of tasks on processors
  - **resources** → provide service
    - scheduled according to policy (e.g. round-robin)
  - **tasks** → consume service
    - worst and best-case execution times
  - **event models** → activate tasks
    - $\eta^{+/-}(\Delta t)$: Minimum/Maximum number of activations within any time window Δt
    - $\delta^{+/-}(n)$: Maximum/minimum time interval between first and last activation of any sequence of n activations (pseudo-inverse to $\eta^{+/-}(\Delta t)$)

$\eta^{+/-}(\Delta t)$



Complex Activation Patterns

- covers typical PJD event model
  period (P), jitter (J), min. dist. (D)
- variety of activation patterns used in practice
  e.g. periodic + spontaneous, dual cyclic, TDMA
- timing verification can consider them through use of minimum distance functions
  - i.e. specification of the minimum distance between any n consecutive events ($\delta^-(n)$)



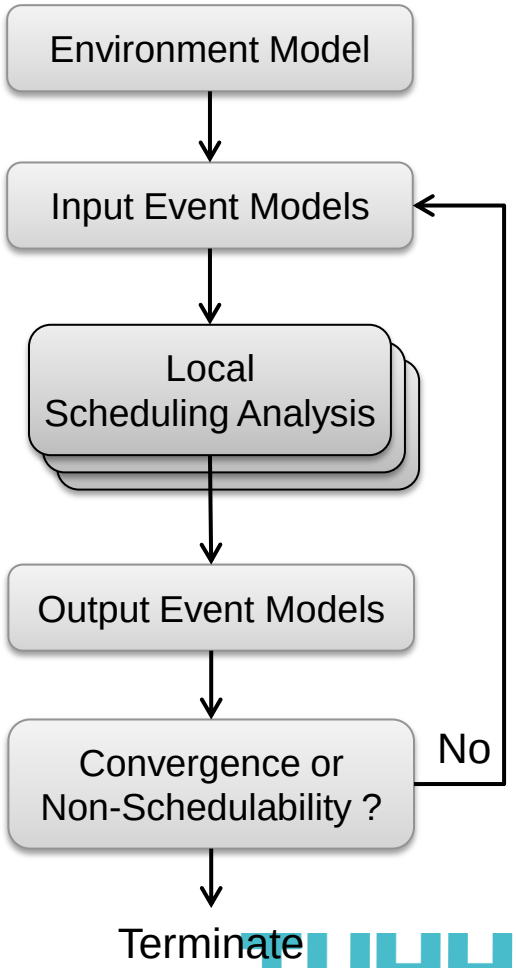
All traces which stay between $\delta^-(n)$ and $\delta^+(n)$ satisfy the event model

3 events may come at once

any 5 events are separated by at least 62.5 ms

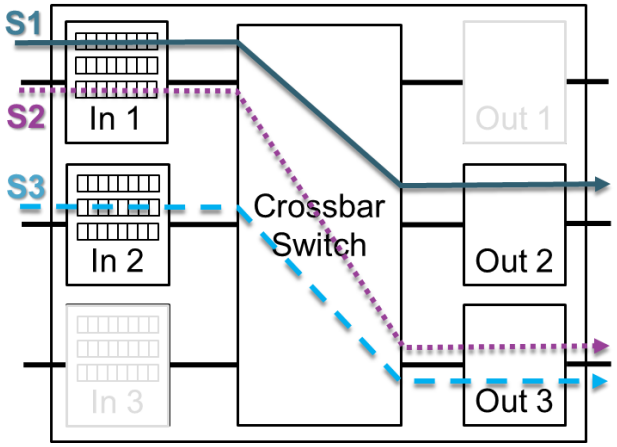
CPA Approach for MPSoC

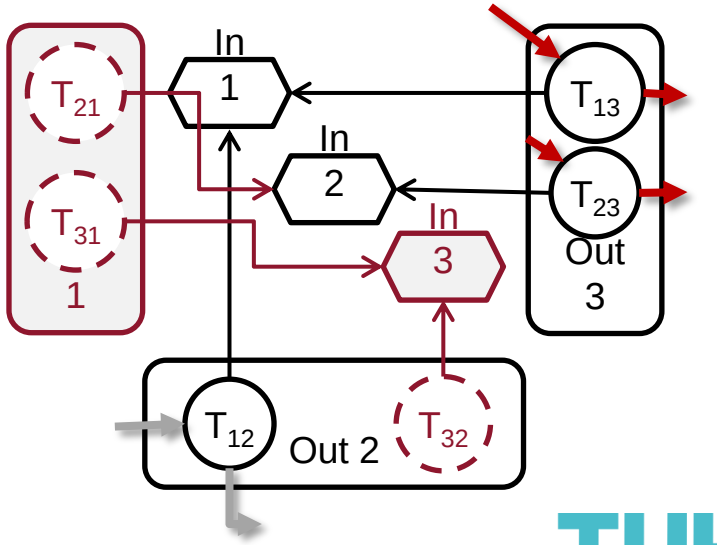
- analysis performed iteratively
- Step 1: define memory access patterns
- Step 2: analyze transmission time in the NoC
  - local analysis (at each router) or global (whole NoC) depending on the QoS mechanism
  - **compute worst-case response time (R^+) of flits based on critical instant (busy window)**
  - **propagate event models downstream**
  - derive output event models
- Step 3: analyze the memory – global
  - **use output model from the whole NoC**
  - **local analysis for the selected memory scheduler**
  - go to step 1 if non-schedulable
  - otherwise, terminate



Application of CPA:
Analysis of NoC Router with 2-Stage Scheduling

- output ports → processing resources
- input ports → shared resources with mutually exclusive access
- traffic stream → chain of tasks mapped to resources
- flit transmission → task execution
- flit arrival → task activation
- input and output event models



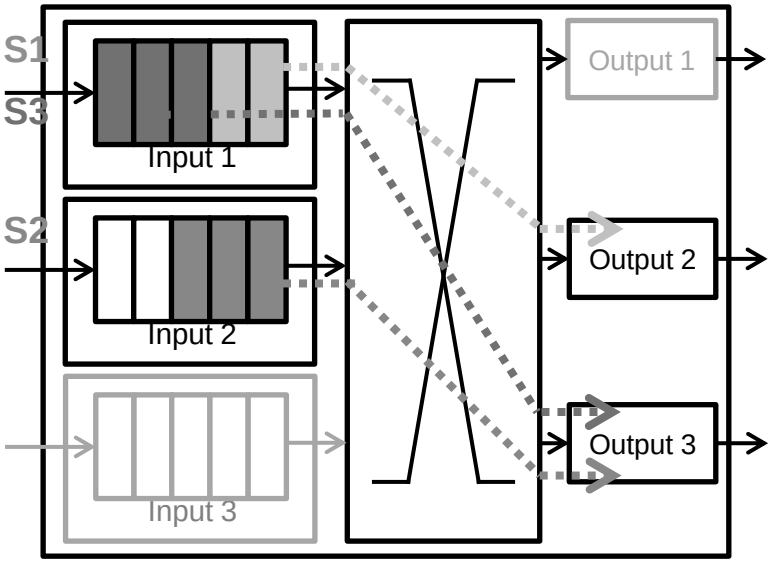


[1] Jonas Diemer, Jonas Rox, Mircea Negrean, Steffen Stein und Rolf Ernst, **"Real-Time Communication Analysis for Networks with Two-Stage Arbitration"** in *Proc. of EMSOFT*, Oktober 2011

Complex Multistage Scheduling

- flit transfer
- output blocking
- FIFO blocking
- backpressure blocking
  - avoid at all cost!

$$B_i^+(q, a_i^q) \leq \mathbf{q} * \mathbf{C} + B_i^{out}(B_i^+(q, a_i^q) - C, q) + B_i^{fifo}(B_i^+(q, a_i^q), q, a_i^q) + B_{P(i)}^{bp}(q)$$

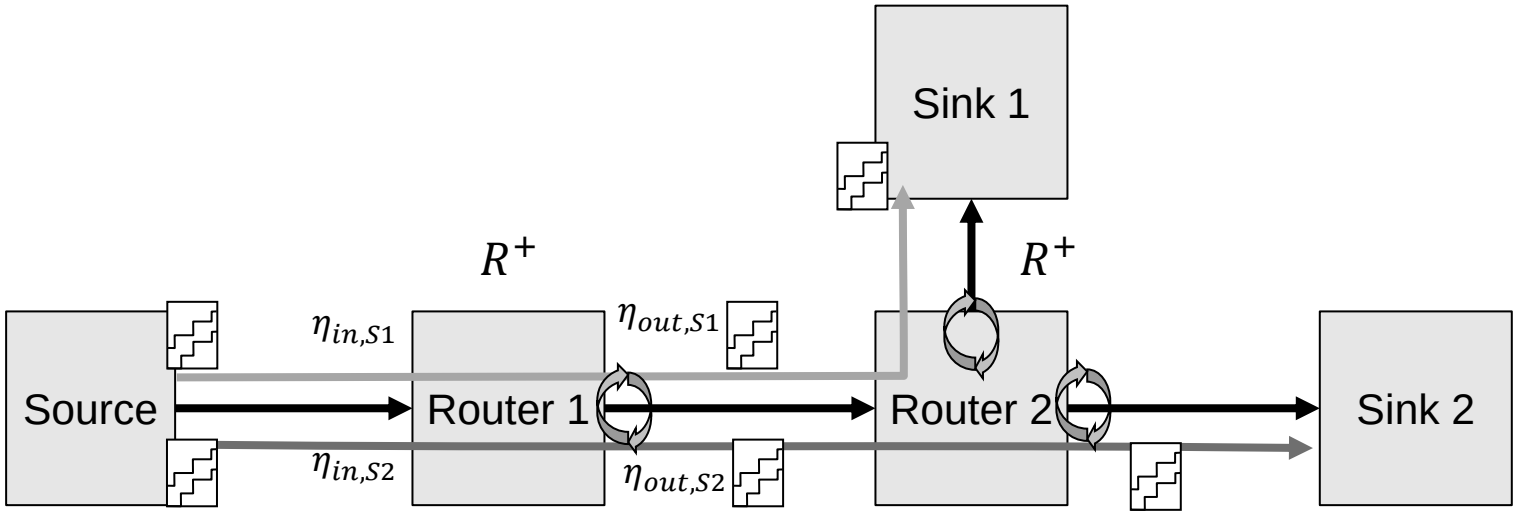


q : number of flits
a_i^q : arrival time of event q
C : single flit transmission time

[2] Sebastian Tobuschat und Rolf Ernst, "**Real-Time Communication Analysis for Networks-on-Chip with Backpressure**" in 2017 Design, Automation Test in Europe Conference Exhibition (DATE), 2017

NoC Analysis – Local QoS

- worst-case end-to-end latency
  - relies on response times $\boldsymbol{R}^+$ from local analyses
- for each stream
  - analyze routers along its path and propagate event models downstream
  - formally analyze routers iteratively



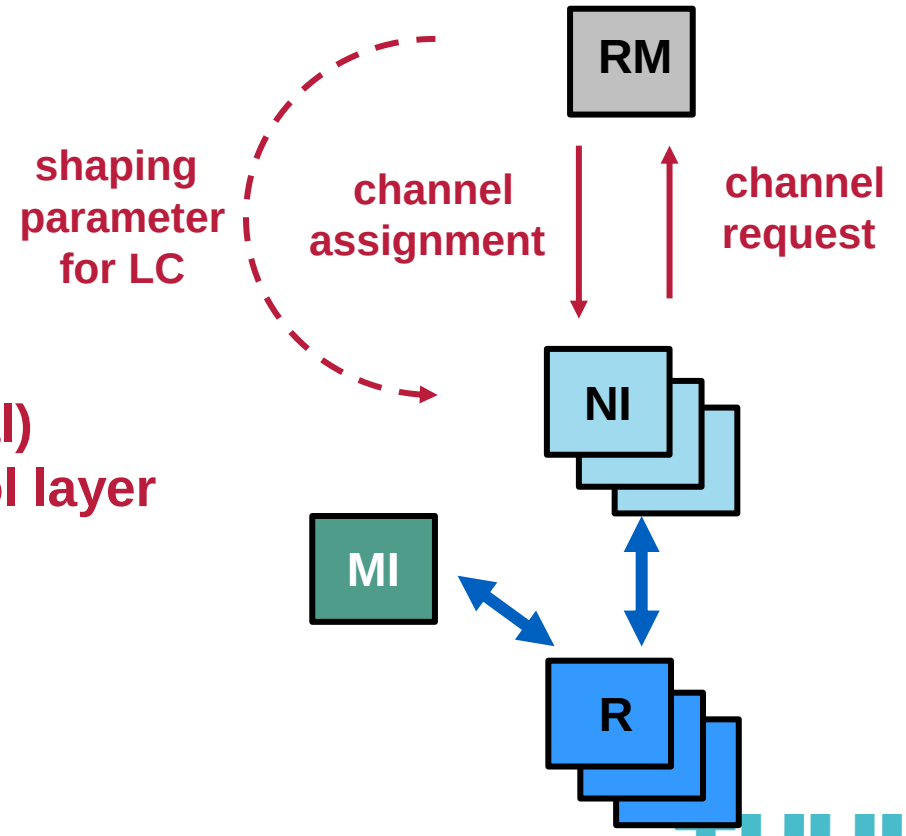
Problems with Local Arbitration in MPSoC

- **dynamics and heterogeneous components increase complexity of analysis**
  - difficult verification
  - frequently highly pessimistic

- **Solutions**:
  - further adjust the model's complexity to decrease pessimism
    - even higher complexity of analysis and hard verification
  - **introduce QoS mechanisms to simplify the model complexity**
    - **e.g. global arbitration for the interconnect**

Alternative Solution: Extension for dynamic channel reservation

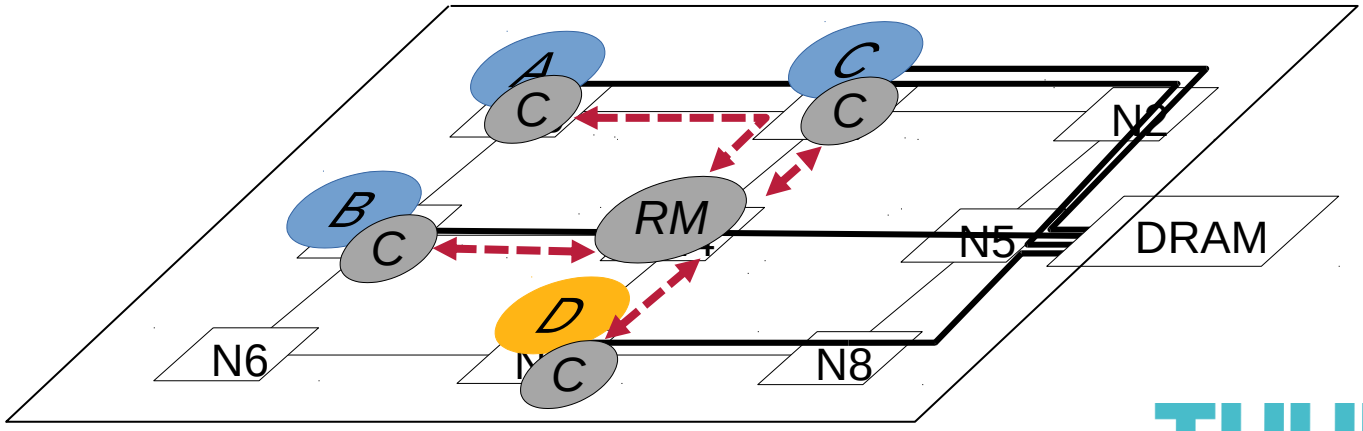
- include Resource Manager (RM)
  - controls resource assignment in NoC e.g. global and dynamic tile level traffic shaping
  - Basic concepts of Software Defined Network (SDNs) adjusted and extended for achieving functional safety
  - Simpler analysis especially for longer transmissions!



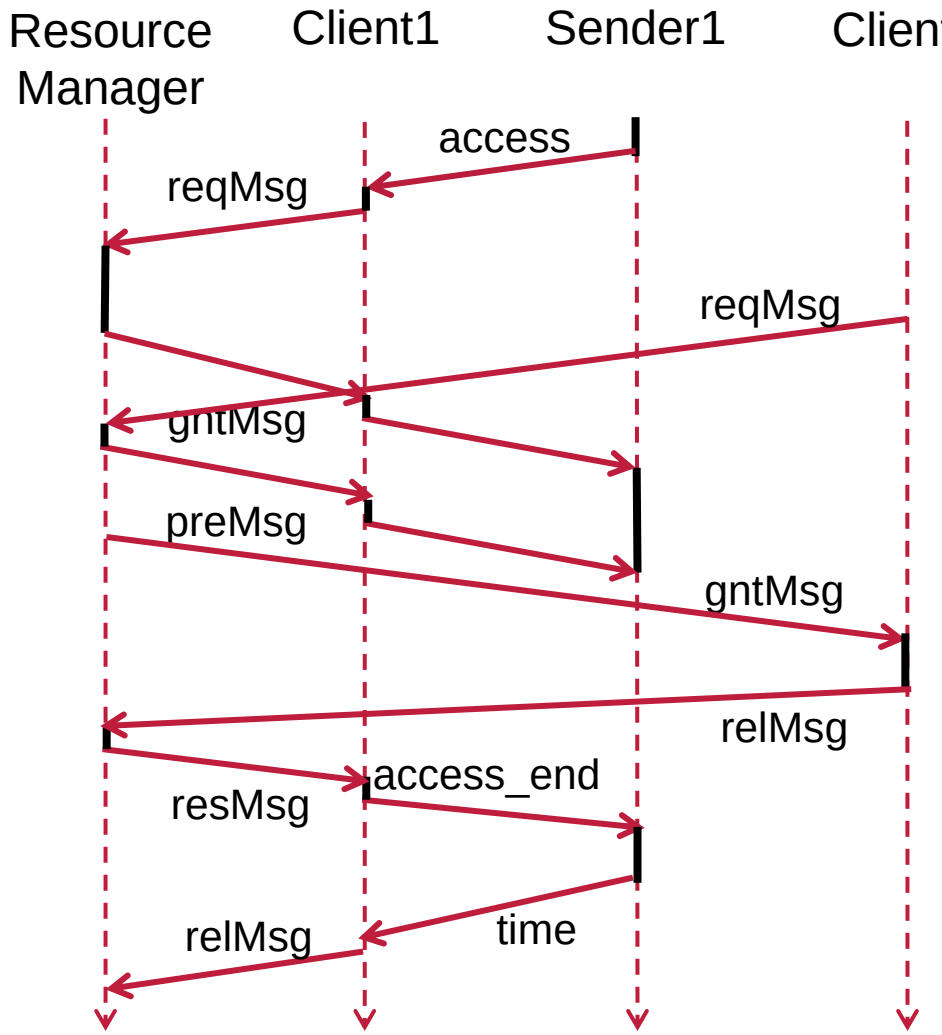
RM: resource manager
NI: network interface
R: router
MI: memory interface

Overlay network

- overlay network to decouple data flow and control protocol
- data layer – data transport and data routing and arbitration
- control layer – global and dynamic arbitration
  - clients - admission control locally in nodes
  - RM – central scheduling unit
  - protocol based synchronization
  - similar to SDN but with real-time and safety for sufficient independence!



Example1: Resource reservation protocol for DMA



single flit control message size (128 bit)

[3] Adam Kostrzewa, Selma Saidi und Rolf Ernst, **"Dynamic Control for Mixed-Critical Networks-on-Chip"** in *IEEE Real-Time Systems Symposium (RTSS)*, (San Antonio, TX, USA), Dezember 2015.

Predictability

- the worst-case time necessary to conduct **q transmissions** synchronized with the RM using SPP-based scheduling is bounded by

$$w_i(q) = \underbrace{q * C^+_i}_{\text{duration of q transmissions}} + \underbrace{3q * C^+_{i,ctrl}}_{\text{for each transmission 3 ctrl messages (req, ack, rel)}} + \underbrace{B_{i,IPD}(w(q))}_{\text{blocking → preemptions of applications with lower priorities}} + \underbrace{B_{i,DPD}(w(q))}_{\text{blocking → preemptions of applications with higher priorities}}$$

Indirect Preemption Delay (IPD)

- additional latency which q requests will experience in the worst-case due to preemptions of ongoing transmissions with a lower priority:

$$B_{i,IPD}(\Delta t) = \min\left\{ q\,,\ \sum_{j \epsilon lp(i)} \eta^{+}_{j,RM}(\Delta t) \right\} * \left(\max_{\forall j \epsilon lp(i)} C^{+}_{i,ctrl} + \max_{\forall j \epsilon lp(i)} C^{+}_{i,pckt} \right)$$

# of trans.

# of trans. with lower priority

min function because we cant preempt

- more than activated trans. with lower priority
- and than our activations q.

duration of blocking

- sending a preemption message
- waiting until all packets are out of NoC

Direct Preemption Blocking (DPD)

- blocking which q requests experience in a time window Δt, due to transmissions with a higher priority :

$$B_{i,NPD}(\Delta t) = \sum_{j \in hp(i)} \eta^{+}_{j,RM}(\Delta t) * \left(C^{+}_{i,ctrl} + C^{+}_{j} + 2 * C^{+}_{j,ctrl} \right)$$

# of trans. with higher priority

duration of the preemption

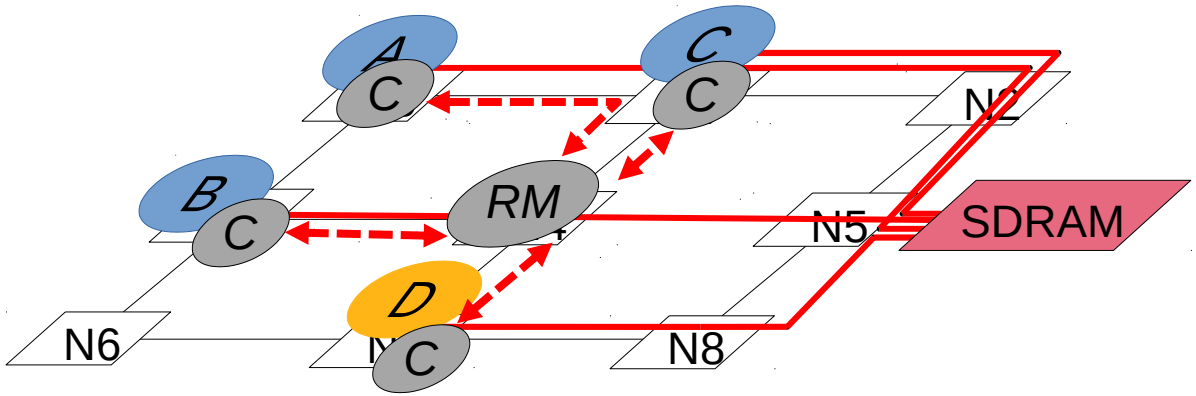
- ack and rel messages
- resume message
- duration of the high-prio transmission

Presentation Outline

- Motivation MPSoC
- MPSoCs with CPA
- Analysis
  - Arbitration at the NoC level
  - **Interface to SDRAMs**
- **Results**
- **Summary**

Resource Manager and SDRAM Scheduling

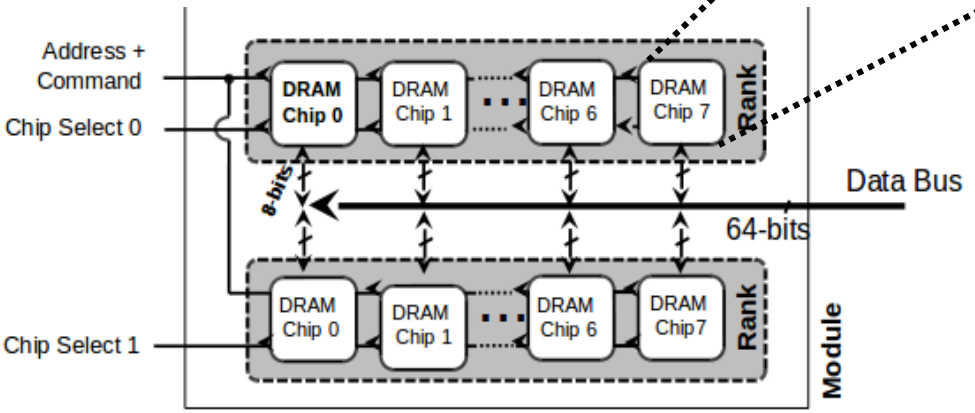
- due to the preservation of spatial locality, a simple SDRAM controller that serves requests in FCFS order suffices
  - SDRAM scheduling is implicitly delegated to RM
  - Hence, *safety* can be addressed at the RM level

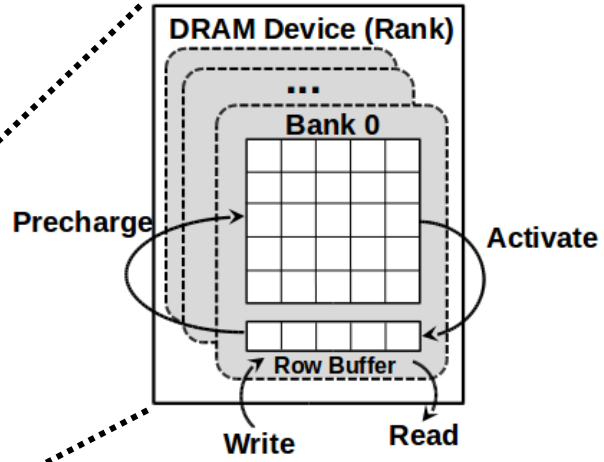


[4] Adam Kostrzewa, Selma Saidi, Leonardo Ecco und Rolf Ernst, **"Ensuring safety and efficiency in networks-on- chip“,** *Elsevier Integration, the VLSI Journal*, 2016.

Interface with DRAMs Controllers

- **DRAM have a complex stateful structure**
- Timing depends on the sequence of memory accesses
- Response time depends on the history of accesses
  - always the most pessimistic latency
  - or account for the pattern (open research question)
  - DRAM latency is very sensitive to the **locality of accesses**





Activate [A] : Load a matrix row into a row buffer ***(Opens a row)***

Write [W] : Writes to the row buffer

Read [R] : Reads from the row buffer

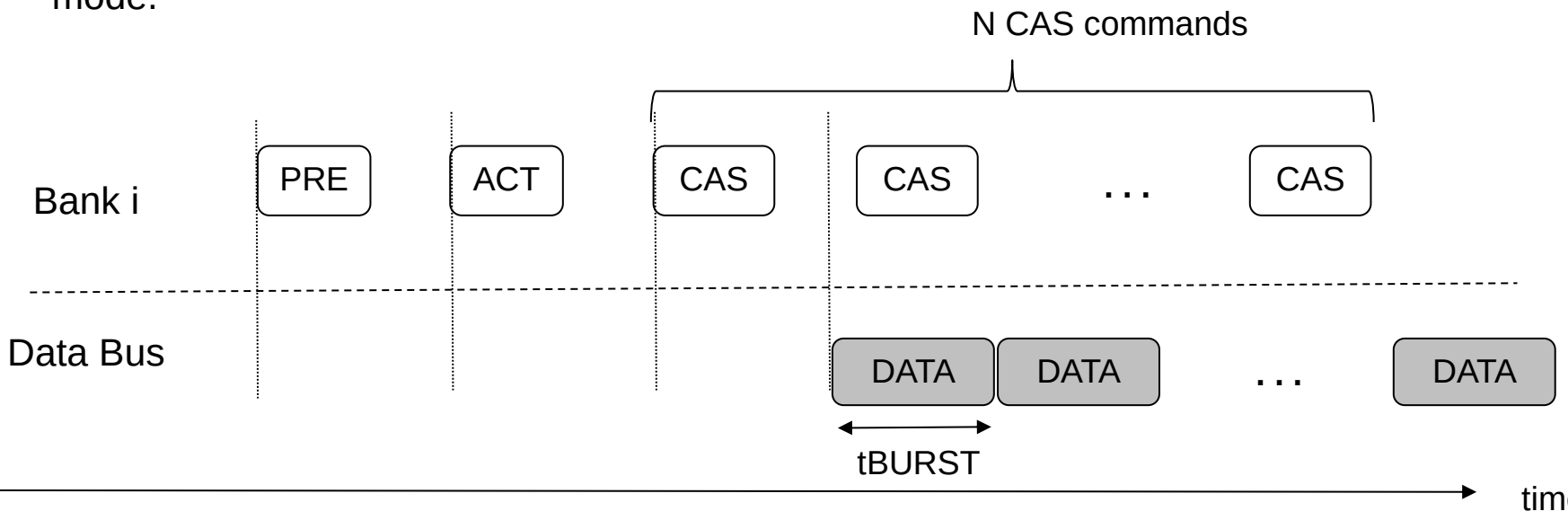
Precharge [P] : Send the contents of the row buffer back to the memory array ***(Closes a row)***

Refresh : executed regularly to prevent capacitors from discharging

Interface with DRAMs Controllers

DMA and DRAM Granularity

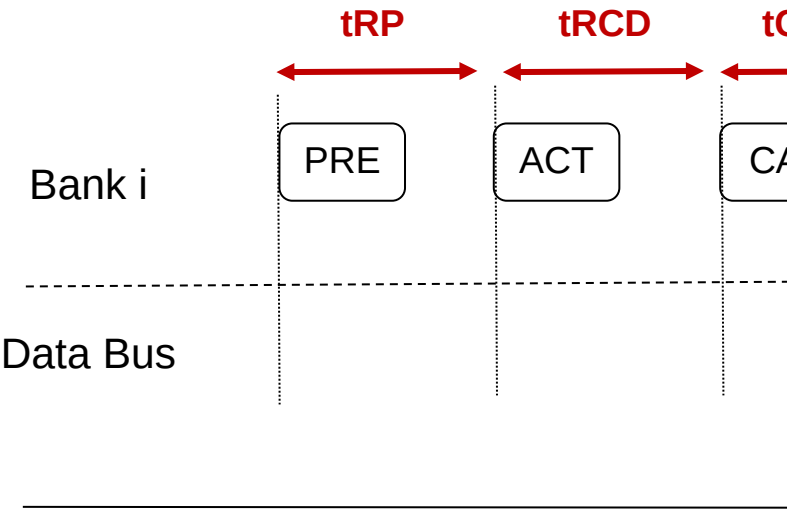
- A memory request to the DRAM is then decomposed by the memory controller into a sequence of internal DRAM commands: ACT, PRE and CAS executed every cycle.
- DRAM granularity = CAS granularity to perform one data transfer (e.g a memory request of 64 bytes requires 8 CAS commands.
- DMA command of granularity G is decomposed into N CAS commands served in burst mode.



Interface with DRAMs Controllers

DRAM internal timing constraints

- DRAMs impose internal timing constraints dictating minimum delays between consecutive commands
-



Constraint	Description	DDR2-800C	DD3-1868M
Exclusively intra-bank constraints (same bank)			
t_{RCD}	ACT to read or write delay	4	13
t_{RP}	PRE to ACT delay	4	13
t_{RC}	ACT to ACT delay	22	45
t_{RAS}	ACT to PRE delay	18	32
t_{WL}	write to data bus transfer delay	3	9
t_{RL}	read to data bus transfer delay	4	13
t_{RTP}	read to PRE delay	3	7
t_{WR}	end of a write operation to PRE delay	6	14
Exclusively inter-bank constraints (different banks)			
t_{RRD}	ACT to ACT delay	4	6
t_{FAW}	four ACTs window delay	18	33
Exclusively intra-bank constraints (any bank)			
t_{WtoR}	write to read delay	10	20
t_{RtoW}	read to write delay	6	10
t_{WTR}	end of write data transfer to read delay	3	7
t_{RTW}	read to write delay	6	10
t_{burst}	data bus transfer	4	4
t_{CCD}	read to read or write to write delay	4	4

RM vs. *predictable* SDRAM scheduling

- **predictable SDRAM schedulers**
  - where spatial locality of transfers is not enforced (e.g. TDM with small slots), *predictable* SDRAM controllers are required
  - to deal with lack of locality, such controllers employ *close-page* policy and/or bank interleaving - **D**edicated **C**lose **P**age-**C**ontrollers **(DCPC)**.
  - the operation DCPCs is controlled by two parameters: BI and BC
    - BI (Bank Interleaving) no of banks per access
    - BC (Burst Count) no of *read* or *write commands executed per bank*

- **RM + standard SDRAM controller**
  - keeps spatial locality
  - allows reduction of row buffer open and close operations
  - out-of-order optimization must be turned off (FCFS memory scheduling)
  - choose BI = 1 $\rightarrow$ leads to predictable access timing

Interface with DRAMs Controllers

Dedicated Predictable Memory Controllers: (DCPC).

- Use of a close page policy + bank interleaving
- The goal is to eliminate the correlation between DRAM latency and the locality of accesses
- Bank Interleaving (BI) = 1, Burst Count (BC) =1

Sequence of SDRAM commands:

Interface with DRAMs Controllers

Dedicated Predictable Memory Controllers: (DCPC).

- Use of a close page policy + bank interleaving
- The goal is to eliminate the correlation between DRAM latency and the locality of accesses
- Bank Interleaving (BI) = 1, Burst Count (BC) =1

Sequence of SDRAM commands:

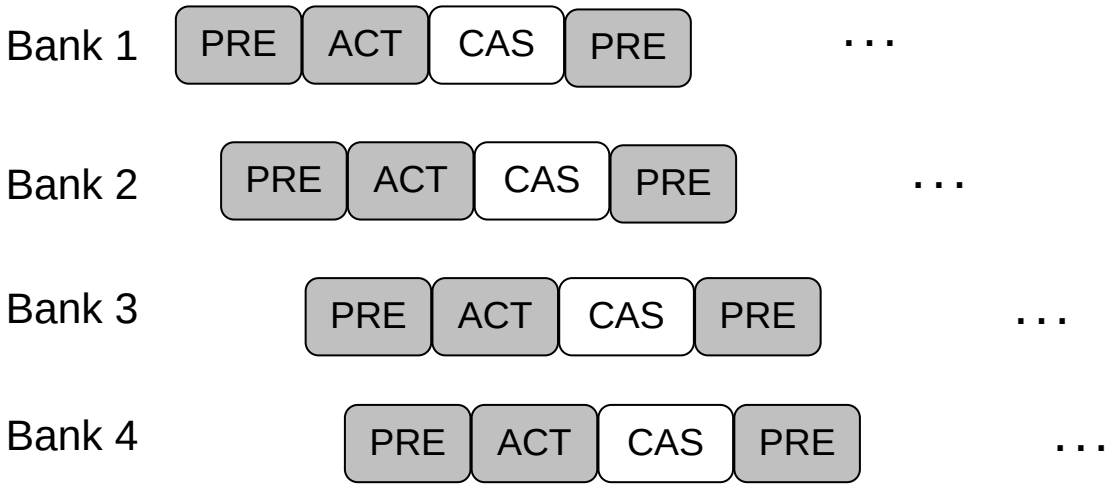
=> Reduced DRAM efficiency

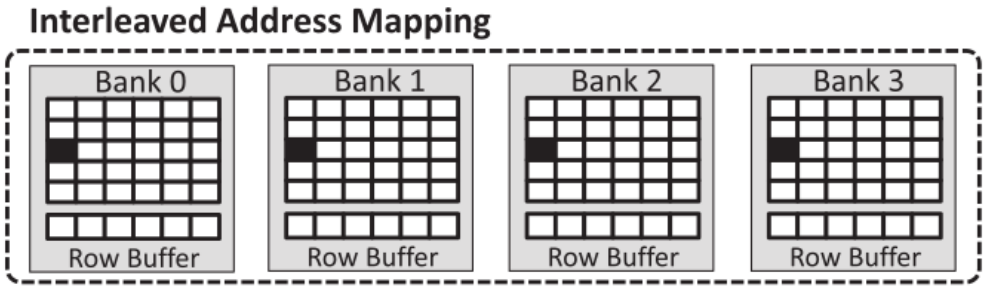
Interface with DRAMs Controllers

Dedicated Predictable Memory Controllers: (DCPC).

- Bank Interleaving (BI) = 4, Burst Count (BC) =1

Sequence of SDRAM commands:





Interface with DRAMs Controllers

Timing Analysis of FCFS SDRAM Controller

- We define a simple timing analysis for SDRAM controller considering FCFS scheduling strategy,
- This analysis can be used in combination with a NoC mechanism which guarantees freedom from interference for an entire DMA transfer composed of multiple packets.
- If freedom of interference is guaranteed then locality of accesses of packets from the same DMA request is also guaranteed,
- The model also assumes that all data is aligned within the boundaries of a single DRAM row
- The analysis determines the worst-case execution time of an SDRAM DMA request

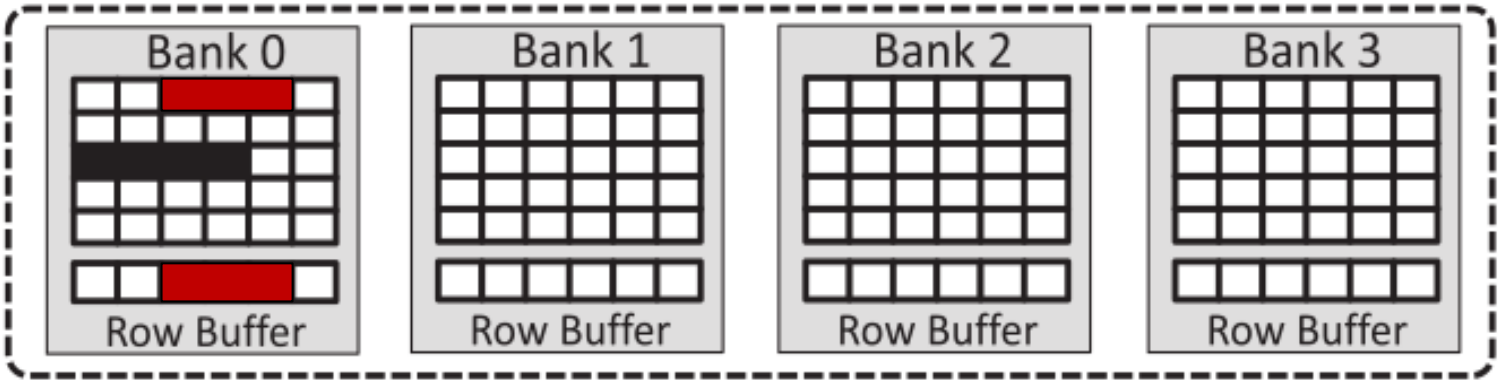
Interface with DRAMs Controllers

Timing Analysis of FCFS SDRAM Controller

- The worst execution time of an SDRAM request is computed as follows under the following worst-case assumptions,
- Worst-Case Assumptions:
  1. The target row is **NOT** already activated,
  2. The previous request is accessing the **same** bank,

Previous request

Current request

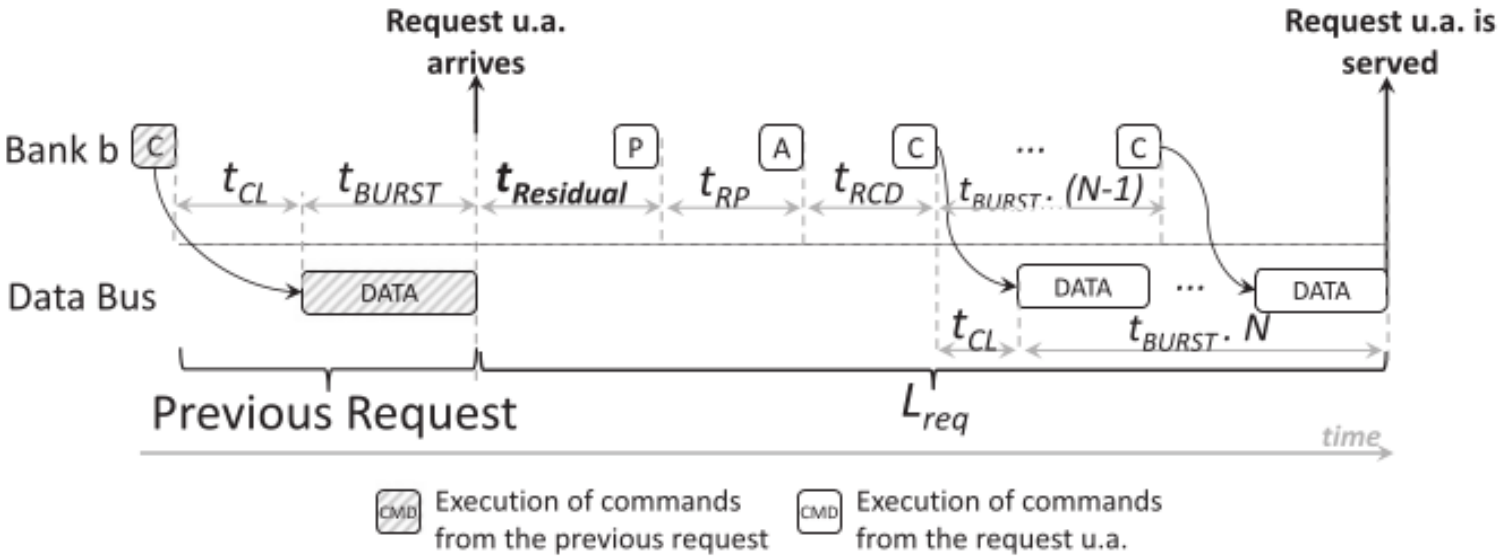


Interface with DRAMs Controllers

Timing Analysis of FCFS SDRAM Controller

- The worst execution time of an SDRAM request is computed as follows,

$$L_{req}^{sdram} = t_{Residual} + t_{RP} + t_{RCD} + t_{CL} + t_{BURST} \cdot N$$

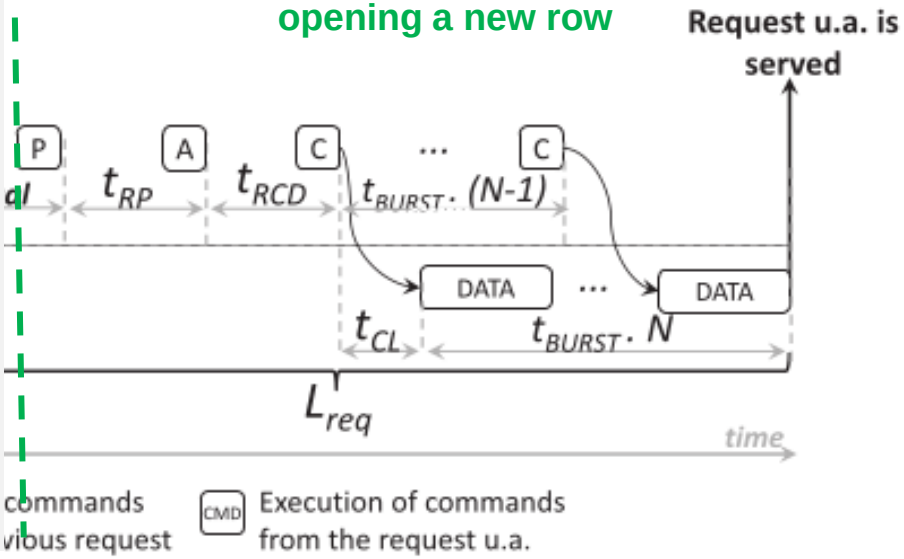


Interface with DRAMs Controllers

Timing Analysis of FCFS SDRAM Controller

- The worst execution time of an SDRAM request is computed as follows,
-

$$+ t_{RP} + t_{RCD} + t_{CL} + t_{BURST} \cdot N$$



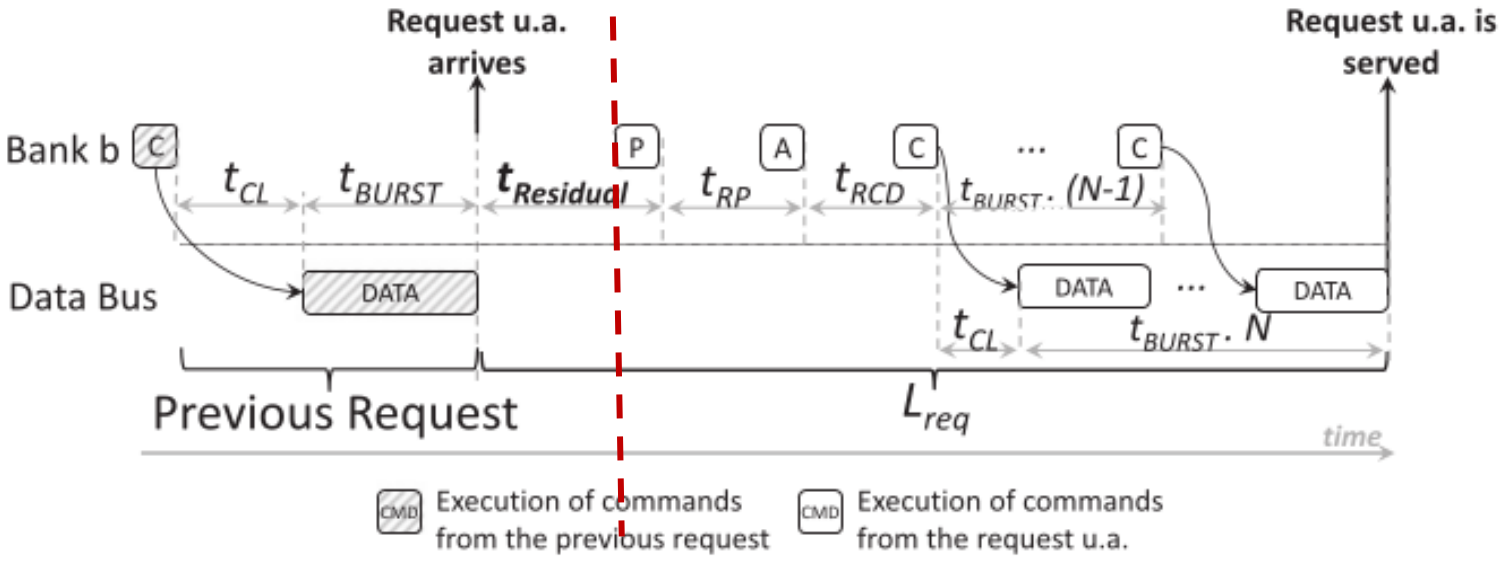
Interface with DRAMs Controllers

Timing Analysis of FCFS SDRAM Controller

- The worst execution time of an SDRAM request is computed as follows,

$$L_{req}^{sdram} = t_{Residual} + t_{RP} + t_{RCD} + t_{CL} + t_{BURST} \cdot N$$

Residual time due to previous request



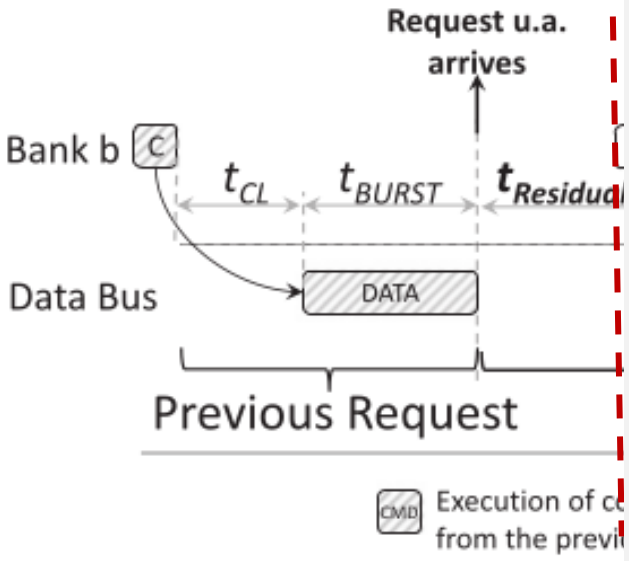
Interface with DRAMs Controllers

Timing Analysis of FCFS SDRAM Controller

- The worst execution time of an SDRAM request is computed as follows,

$$L_{req}^{sdram} = t_{Residual} +$$

Residual time due to previous request



Residual time depends if the previous request is a read or a write,

$$t_{Residual} = \max(case_r, case_w)$$

In case of read,
Differece between the time to perform a read and the ACT to PRE command to load the previous row

$$case_r = \max(t_{RAS} - (t_{RCD} + t_{RL} + t_{BURST}), 0)$$

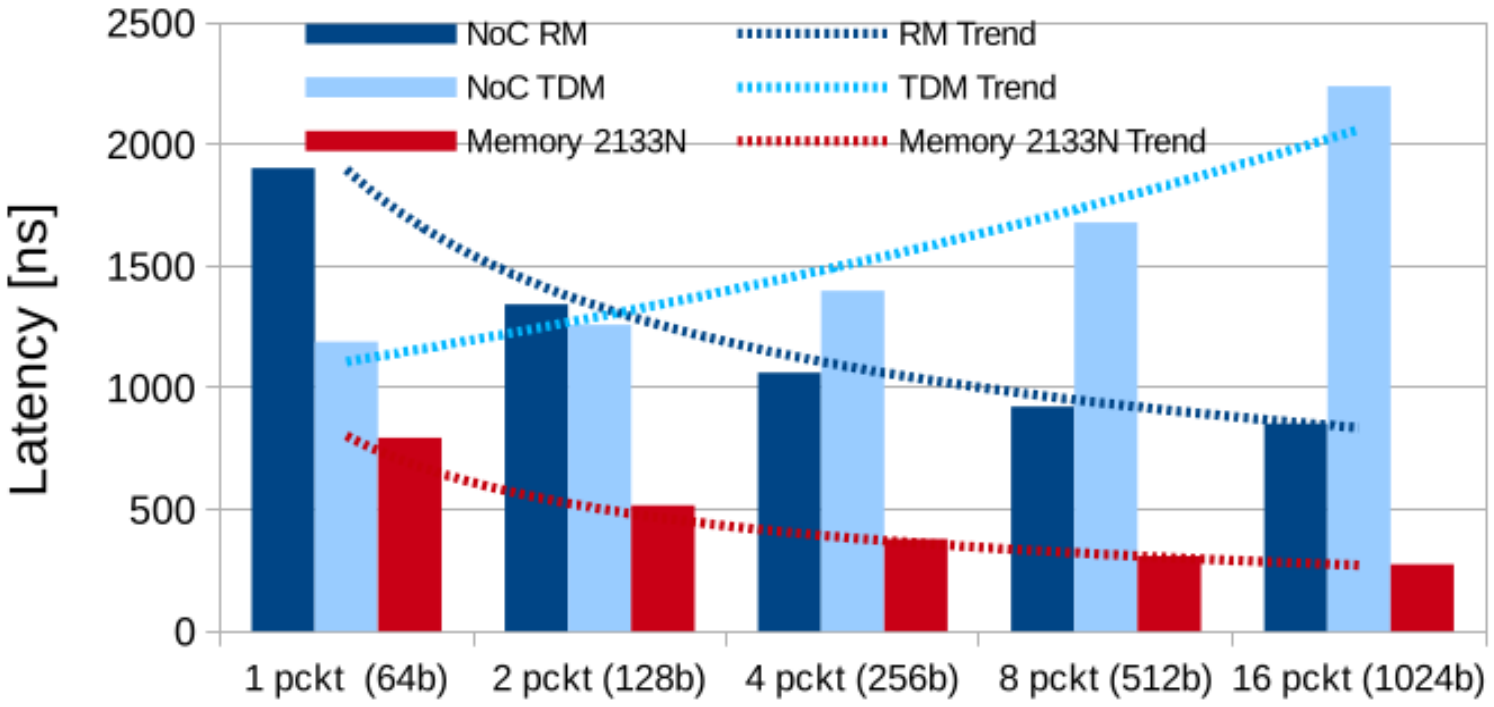
In case of write,
End of WR operation to PRE delay

$$case_w = t_{WR}$$

Presentation Outline

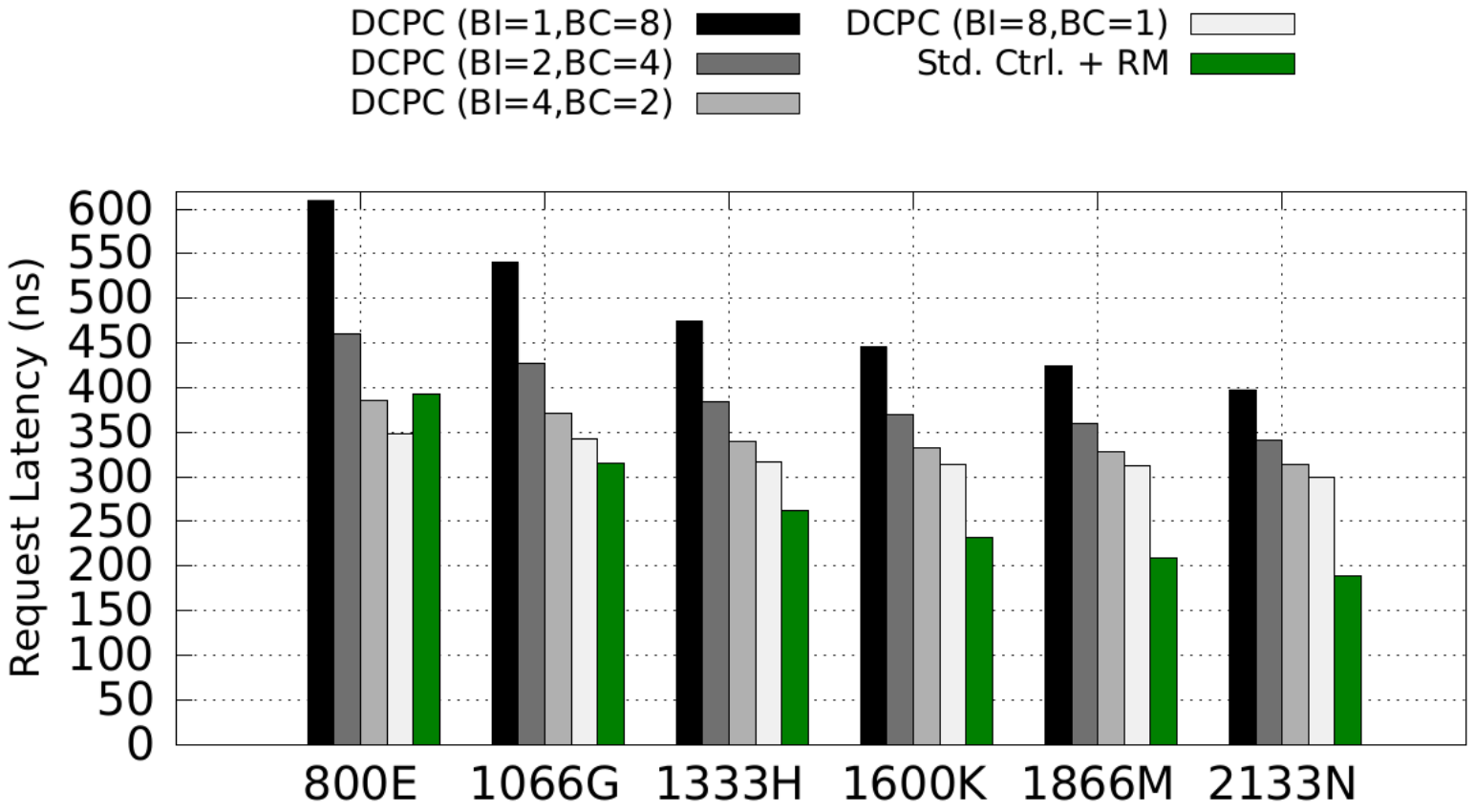
- Motivation MPSoC
- MPSoCs with CPA
- Analysis
  - Arbitration at the NoC level
  - Interface to SDRAMs
- **Results**
- **Summary**

Resource Managers and SDRAM scheduling



Transmission Size (Memory Granularity)

Resource Managers and SDRAM scheduling



Worst-case latency for a 256-bytes request on DDR3 devices (with 8-bit wide interfaces)

Presentation Outline

- Motivation MPSoC
- MPSoCs with CPA
- Analysis
  - Arbitration at the NoC level
  - Interface to SDRAMs
- Results
- **Summary**

Conclusion

- **NoC based many-cores are entering safety critical system design**
- **analysis is not trivial**
  → must cover simultanously safety dynamics and performance!
- **dynamic resource management using a research manager is a highly efficient NoC control mechanism for such NoCs providing worst case guarantees**
- **mechanism supports simpler memory control and transient error handling**

Thank you!

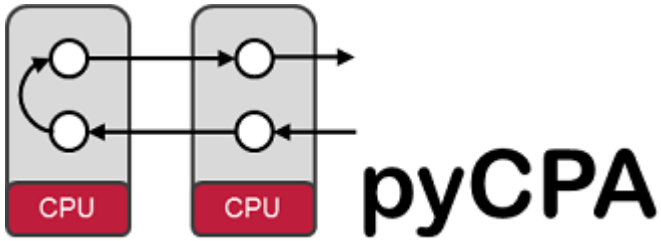
Acknowledgement: Some of the slide contents have been provided by Leonardo Ecco, Sebastian Tobuschat, and Eberle Rambo

Backup Slides

Step 1: Deriving Event Models

- enforced by the system (HW or SW)
  - due to the nature of controller process e.g. periodic activations in control
  - predictable memory models
  - later enforced e.g. by rate-limiter in network interface (cf. RM)

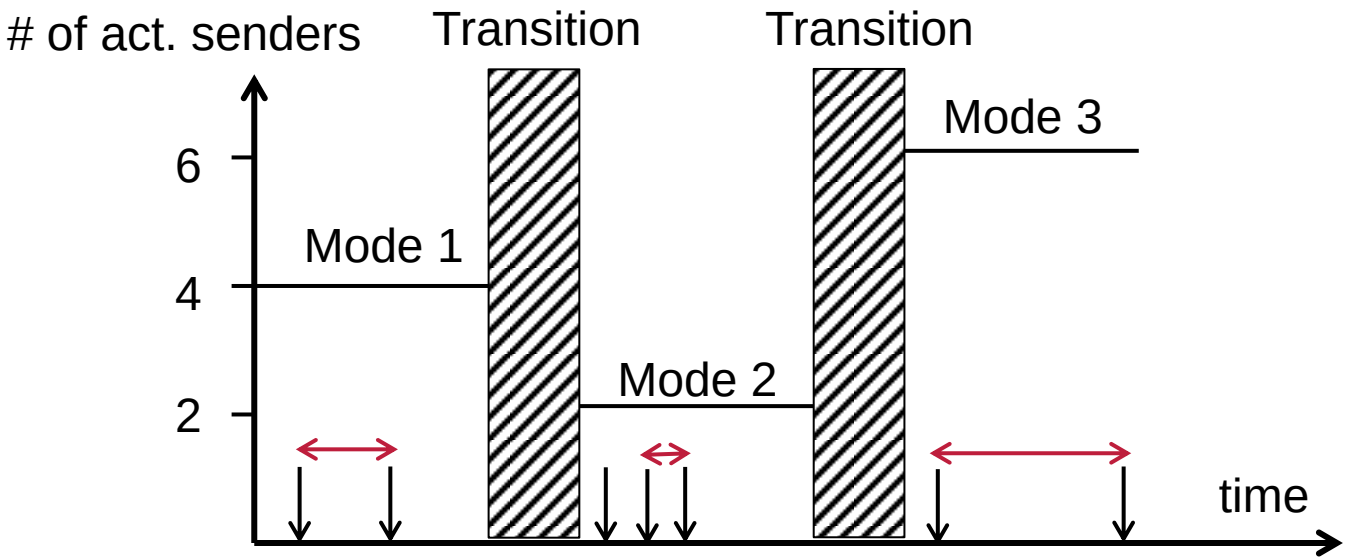
- from simulation / trace
  - obtain models from trace (as shown before)
- **from design / analysis**
  - **formal worst-case models**



- **compatible with current, standard automotive design process**
  - **SymTA/S tool provided by Symtavision (now: Luxoft)**
  - **open source PyCPA tool**

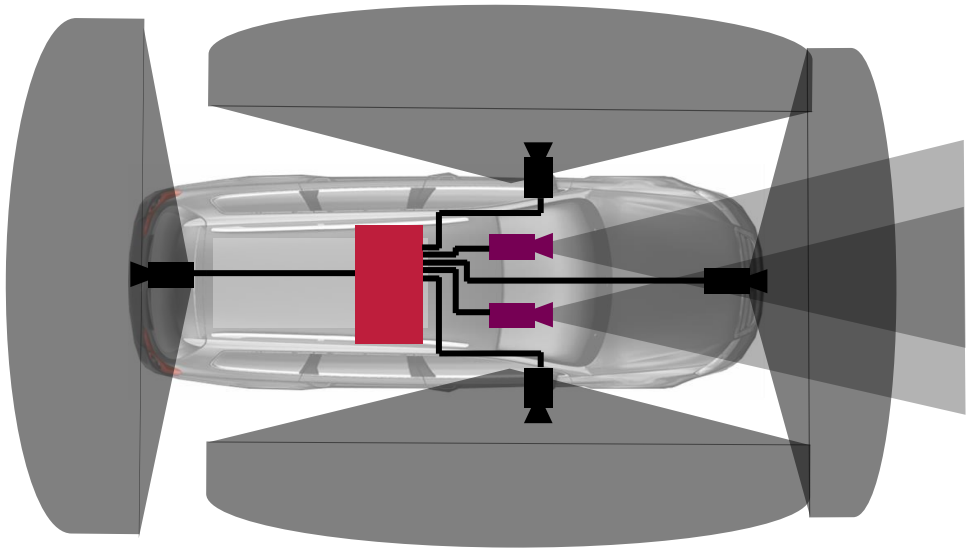
Example2: Cache Protocol

- dynamically adjust arbitration
- correlate rates with the load of the system
  - i.e. the number of simultaneously active senders at runtime
- observe and enforce **behavioral models** supported by the analysis
  - **adaptive QoS -** based on the load of the system **at runtime**



Motivation

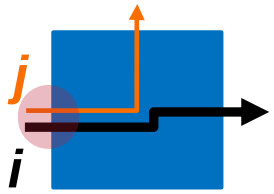
- many-core systems are reaching critical embedded systems
  - sensor fusion and recognition in highly automated driving
- high performance + service guarantees
  - safety + availability
- dynamic system loads
  - function modes
- limited power and cost budget
  - higher systems integration
  - mixed criticality



Calculating the Interference

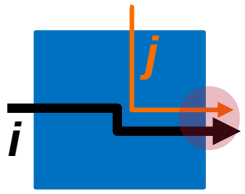
There are 4 possible interference scenarios

Direct Input Blocking



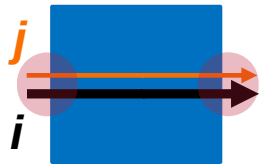
Same Input Port, different Output Ports

Direct Output Blocking



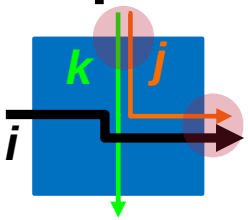
Same Output Port, different Input Ports

Overlapping Streams



Same Input Port and same Output Ports

Indirect Output Blocking



Same Output Port, different Input Ports*

***i**: stream under analysis* ***j**, **k**: interfering stream*

Overview of Formal Analysis Methods/Tools

- compositional approaches
  - real-time calculus (RTC)
    - implemented e.g. in MPA (free, open-source)
  - network calculus
    - implemented e.g. in RTaW-PEGAS (commercial)
  - compositional performance analysis (CPA)
    - implemented in SymTA/S (commercial)
    - now also available in pyCPA (free, open-source; from IDA, TUBS)

Jitter Propagation