ARTIST2 Summer School 2008 in Europe
Autrans (near Grenoble), France
September 8-12, 2008

Dataflow analysis for predictable multiprocessor design

Marco Bekooij
NXP semiconductors

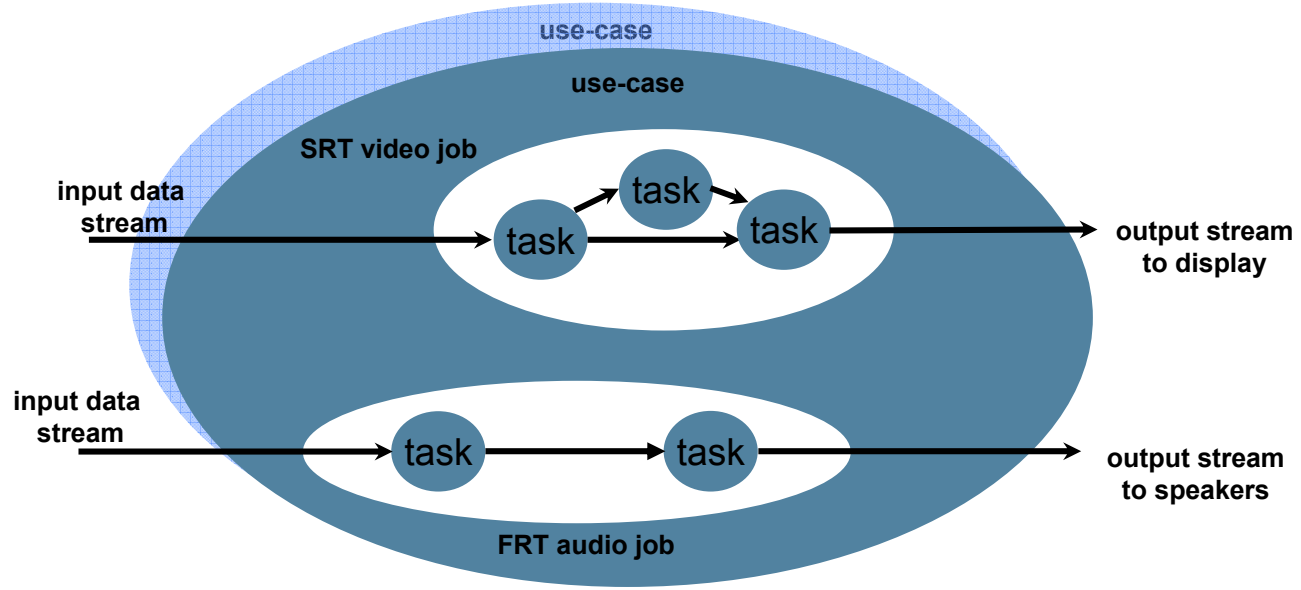
Outline

- Context:
  - Multi-stream car-entertainment systems
  - Application characteristics
  - Architecture characteristics
  - System requirements
- Approach
  - Apply budget schedulers
  - Compute budgets with dataflow analysis techniques
- Dataflow analysis techniques
  - Single rate dataflow analysis example
  - Latency-rate characterization of budget schedulers
  - Generalization: variable rate dataflow
- Summary

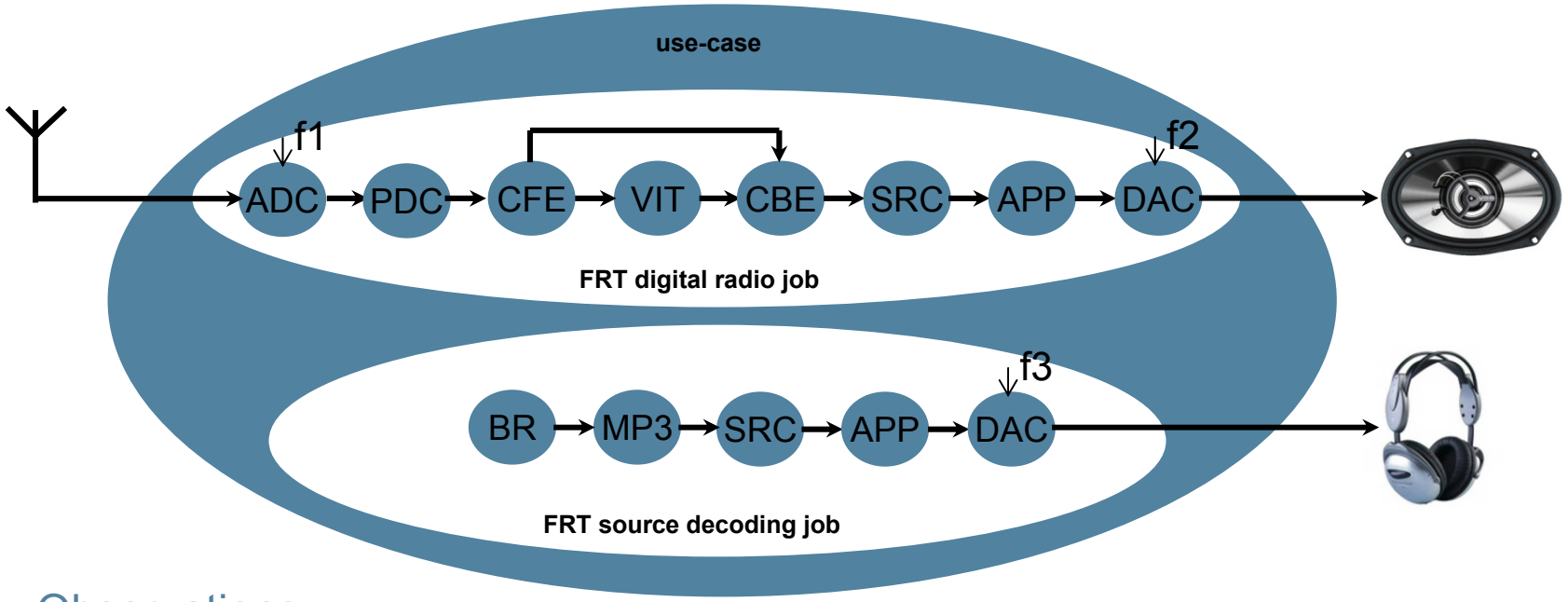
Context: Multi-stream car-entertainment system

Application model

- Jobs are *composed of tasks*
- Simultaneously running jobs *together form use-cases*
- Jobs often have *real-time requirements*
  - Firm (FRT) if deadline misses are *highly undesirable: steep quality degradation*
  - Soft (SRT) if occasional *deadline misses are tolerable*

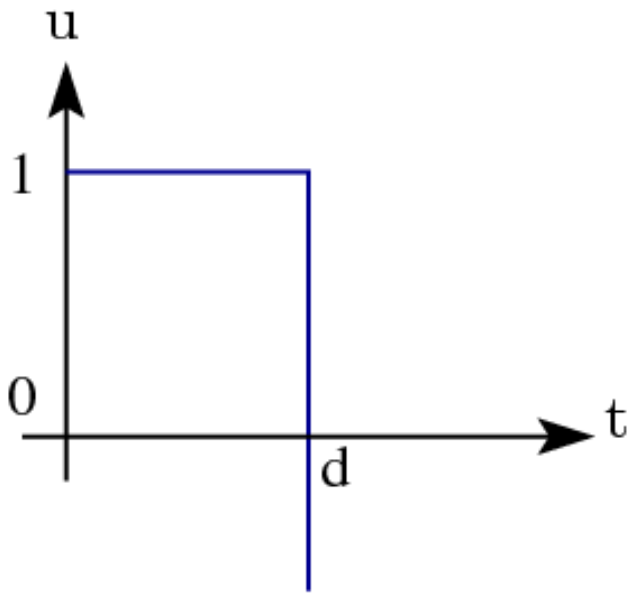


In-car entertainment use-case

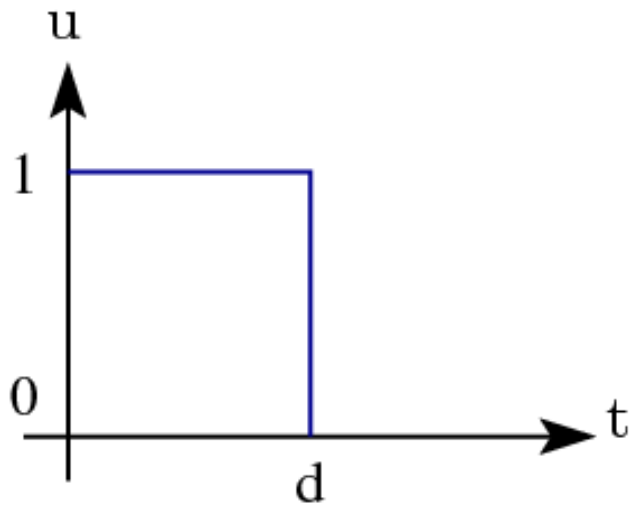


- Observations:
  - Reactive system because stream from transmitter cannot be slowed down
  - Firm real-time jobs because deadline misses are highly undesirable but not catastrophic
  - Both streams are equally important
  - Throughput constraints dominate latency constraints

Hard real-time versus firm real-time



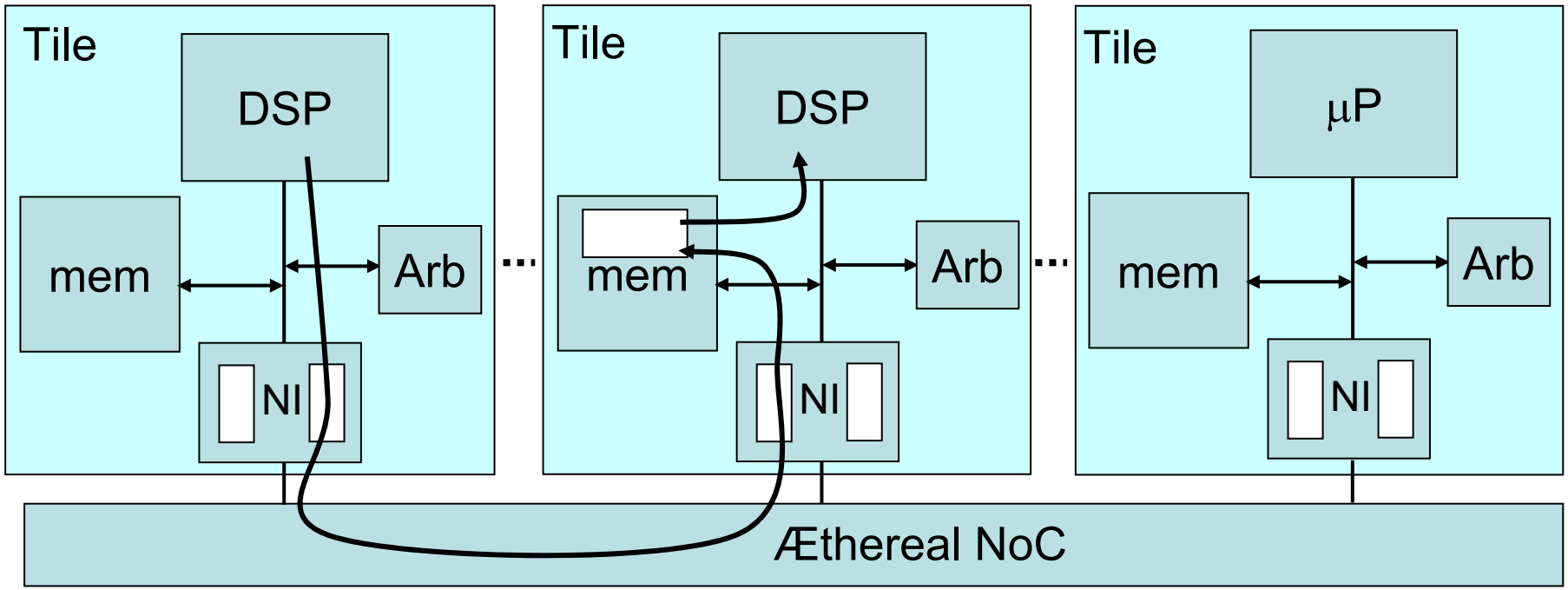
hard real–time



firm real–time

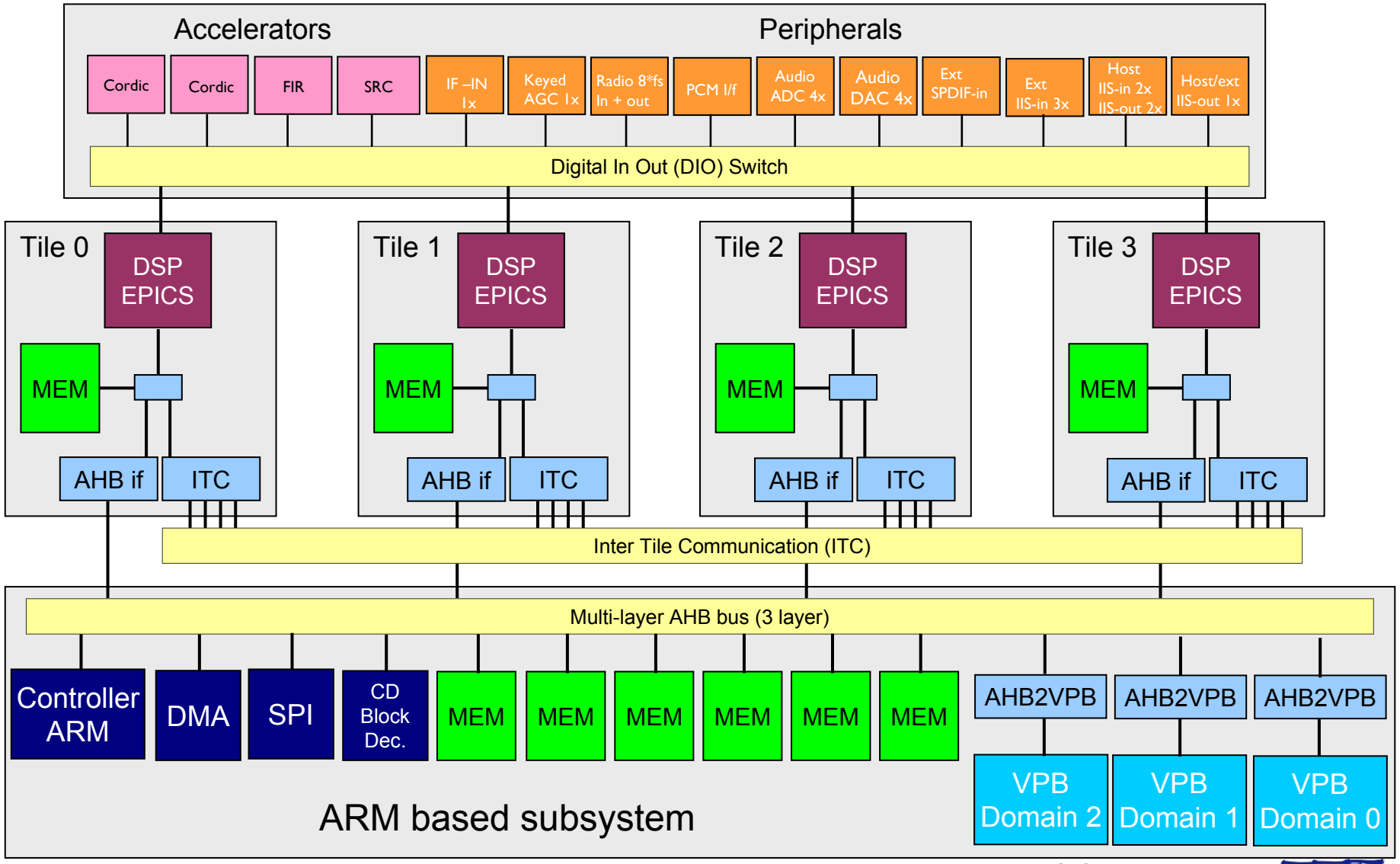
Hard real-time: apply redundancy
Firm real-time: use measured worst-case execution times

Multiprocessor architecture



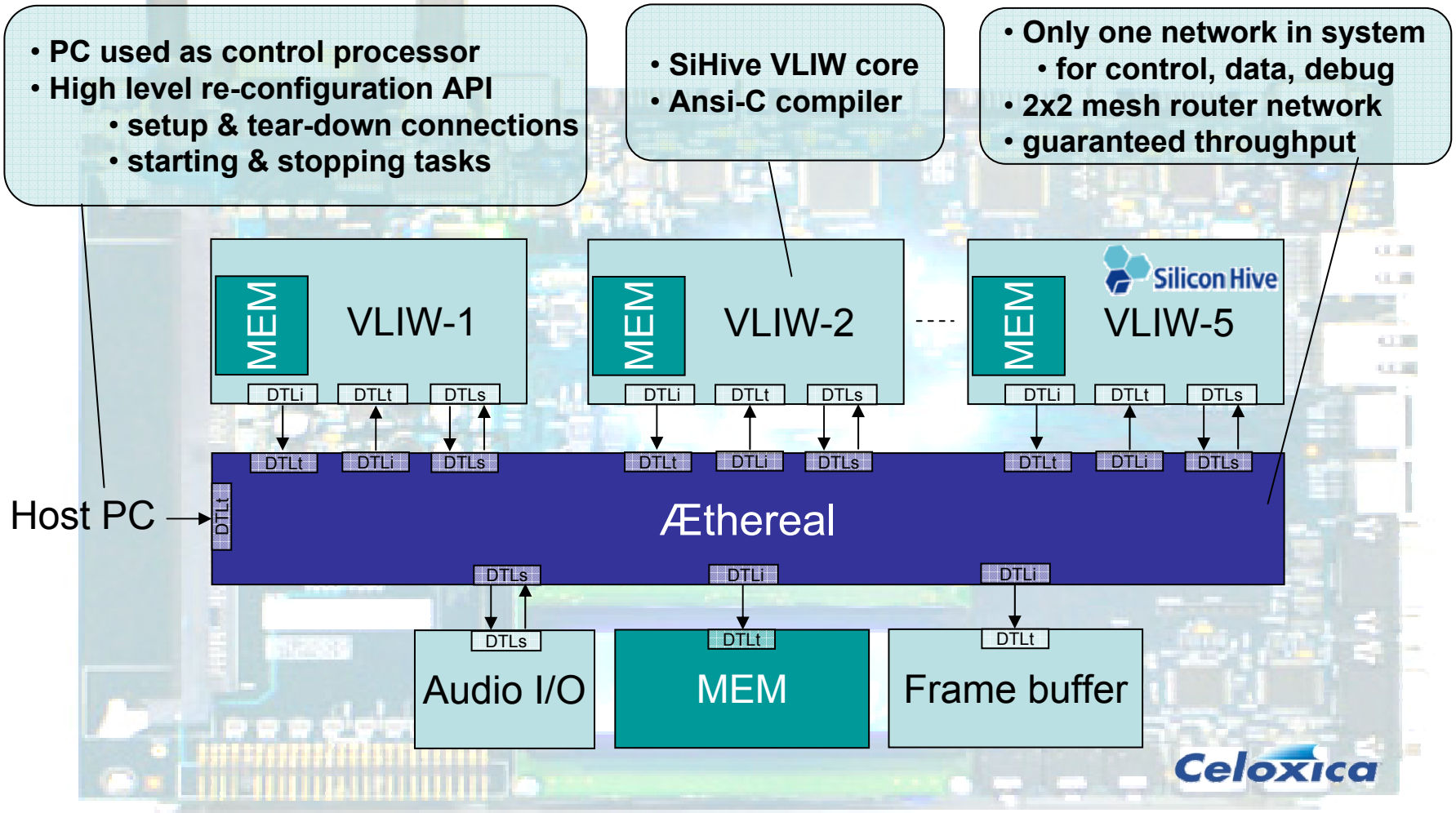
- Distributed multiprocessor architecture with non-uniform memory access latency
- Processors write in destination memory
- Each tile can have its own clock

Car-radio IC of NXP



[A. Moonen et. al., GSPx2005]

Network based FPGA demonstrator

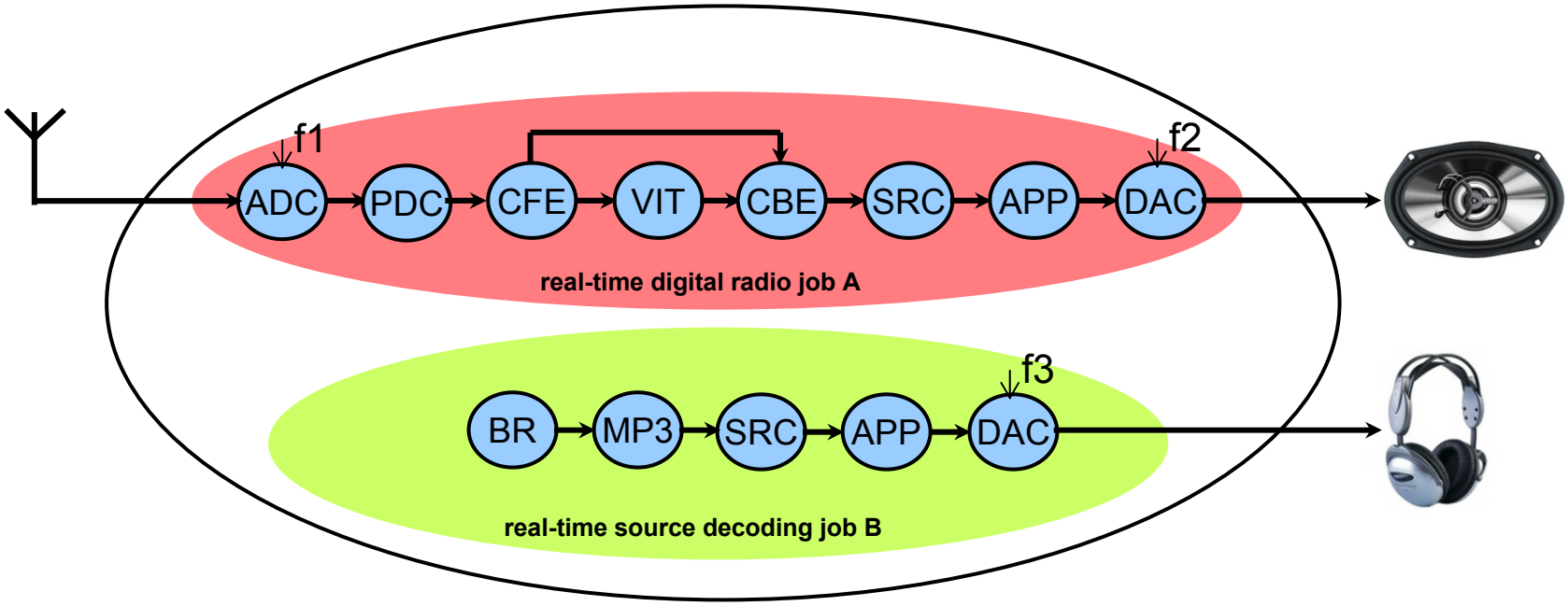


FPGA demo [A. Hansson, TODEAS 2008]

Our objectives:

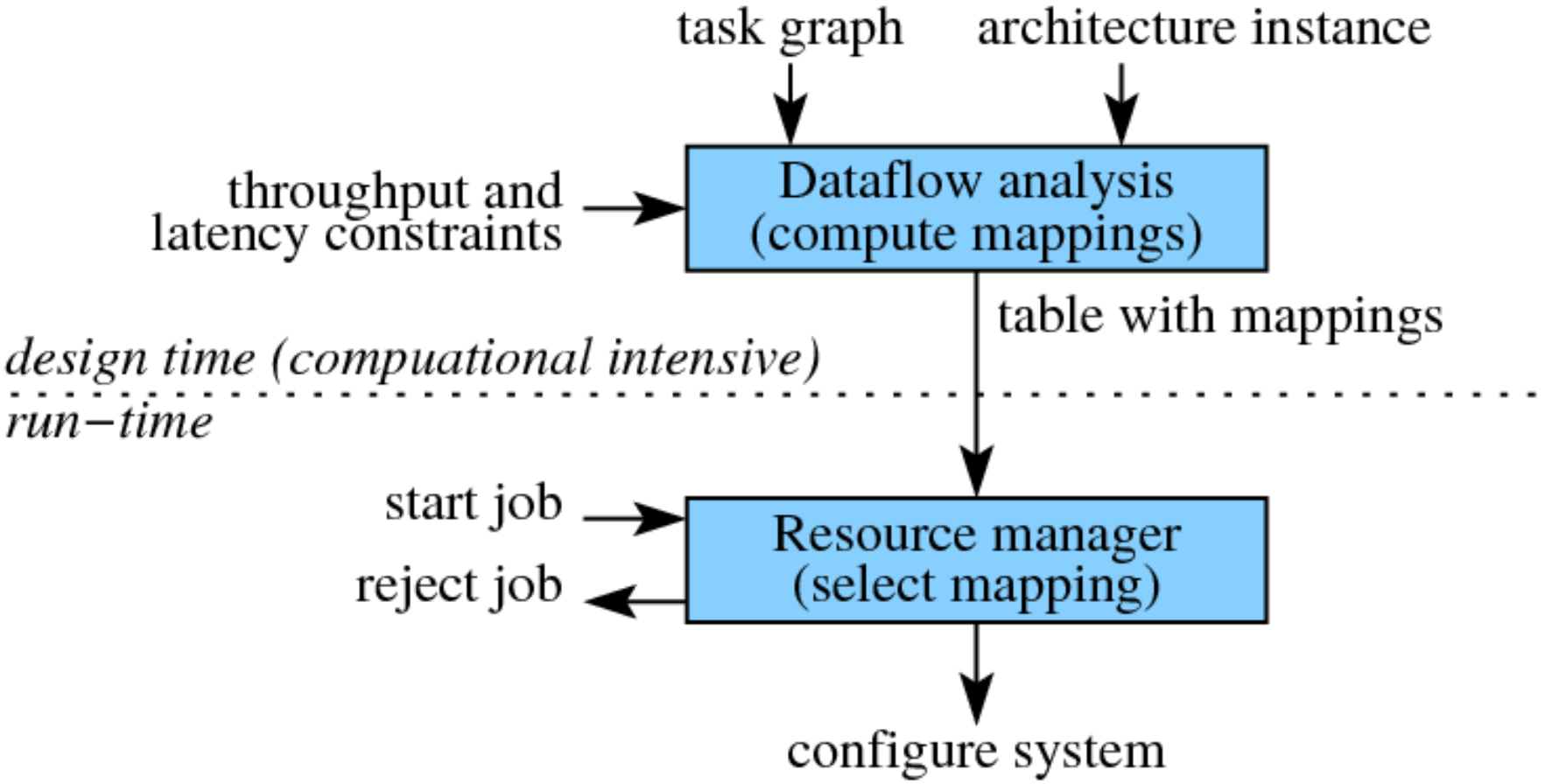
- Reduce design time
  - Enable independent development of jobs
- Improve robustness of the system
  - Guarantee real-time behavior of a job independently of other jobs

Approach: use budget schedulers



- Budget scheduler bound interference other jobs
  - Each job can be designed and characterized independently of other jobs
  - Erroneous behavior of a job has bounded effect on behavior other jobs

Approach: compute mapping alternatives at design-time



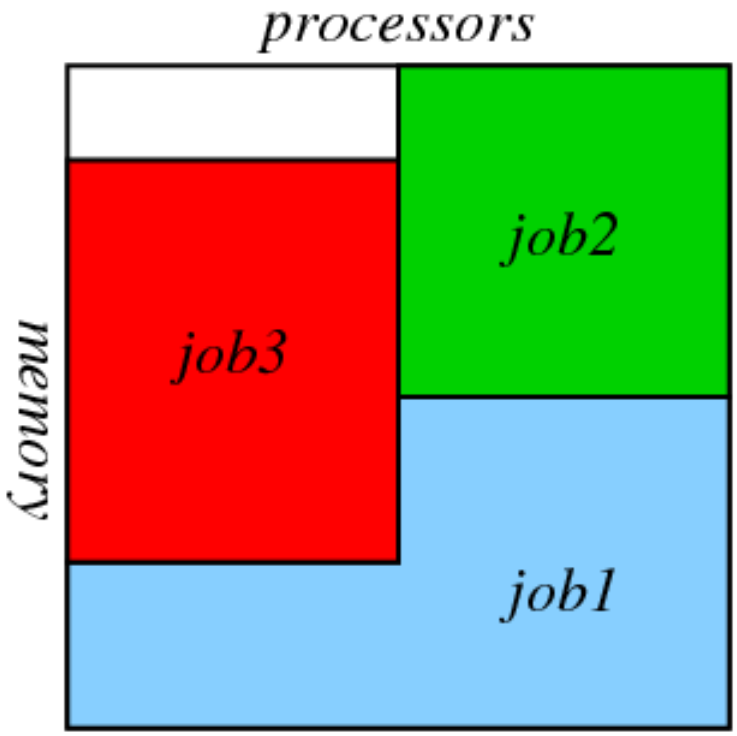
Mapping

- One mapping of a job specifies:
  - Tasks to processor binding
  - Memory map (buffer allocation, code allocation, stack, heap)
  - Scheduler settings (processors, memory controllers, network)
  - Routing in network

Mapping requirements

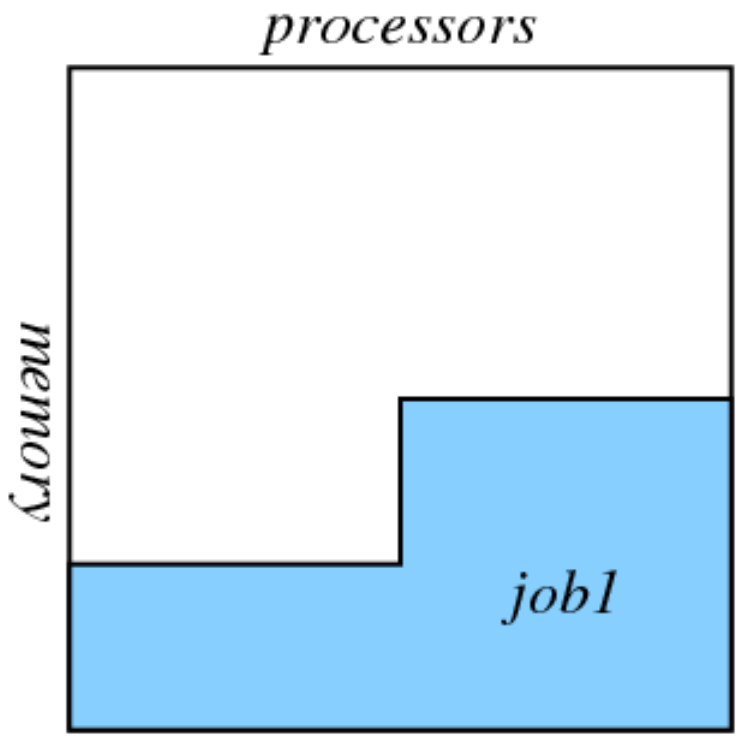
- System must support a predefined number of use-cases
  - Specification describes guaranteed functionality system
- Use-case switching must be non-disruptive
  - Starting a job may not result in a throughput constraint violation of other jobs
- Switching between use-cases may not take more than a predefined amount of time
  - e.g. maximum time between pressing a button and music

Selection of a mapping at run-time

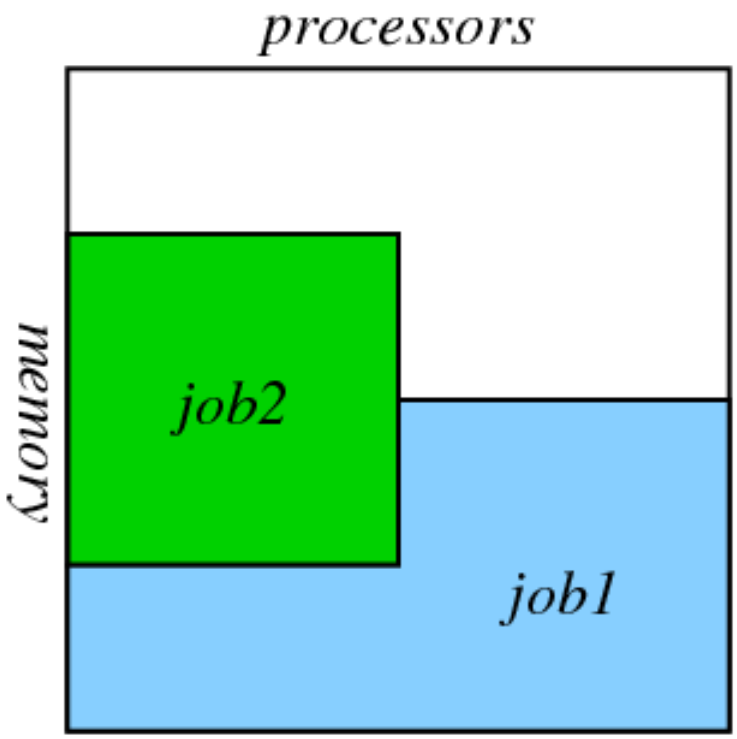


Vector bin-packing like

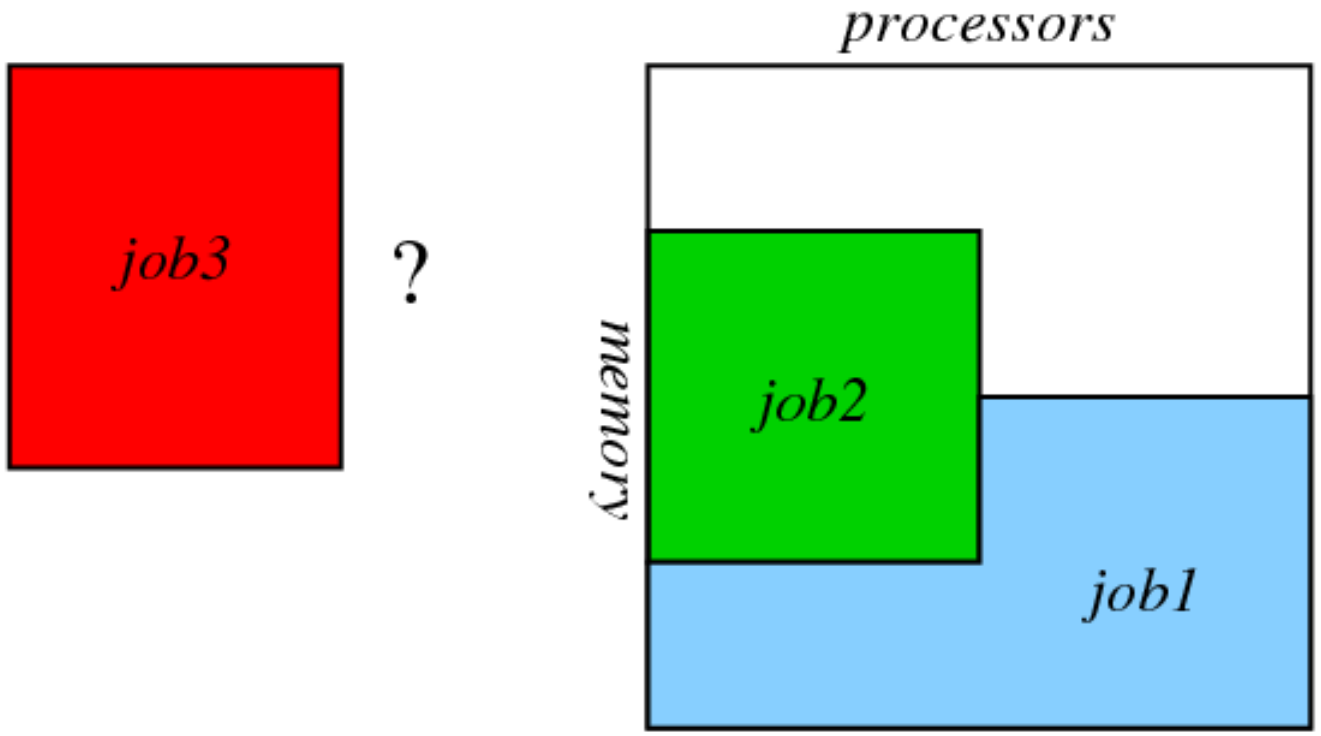
Select mapping of job 1



Select mapping of job 2

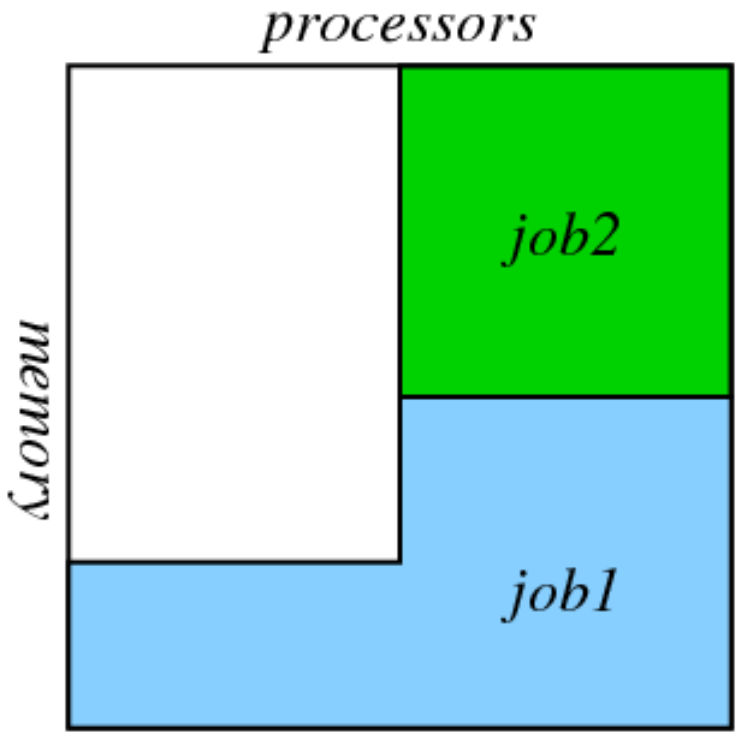


Select mapping of job 3

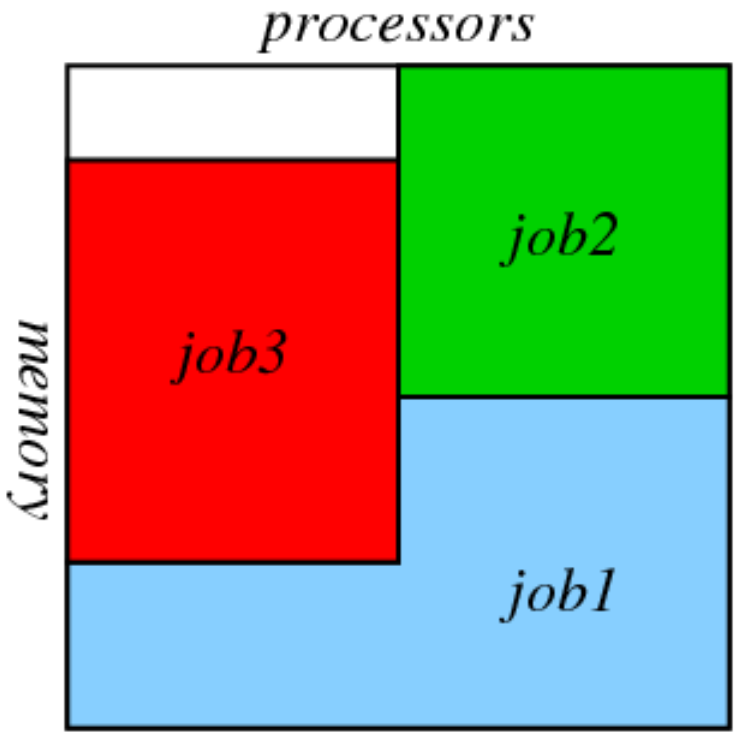


Admission control rejects job3

Select alternative mapping of job 2

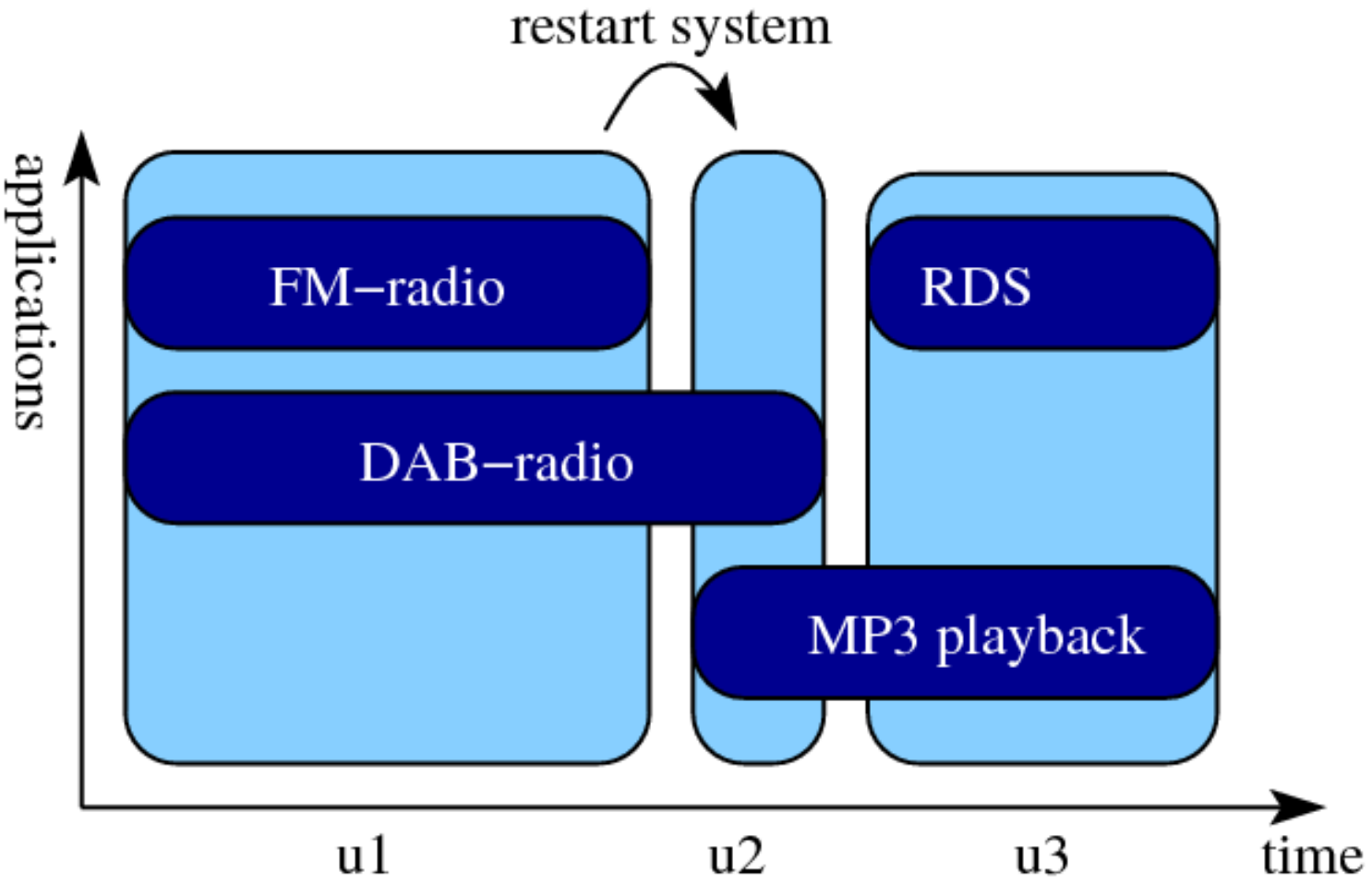


Mapping for job3 exists



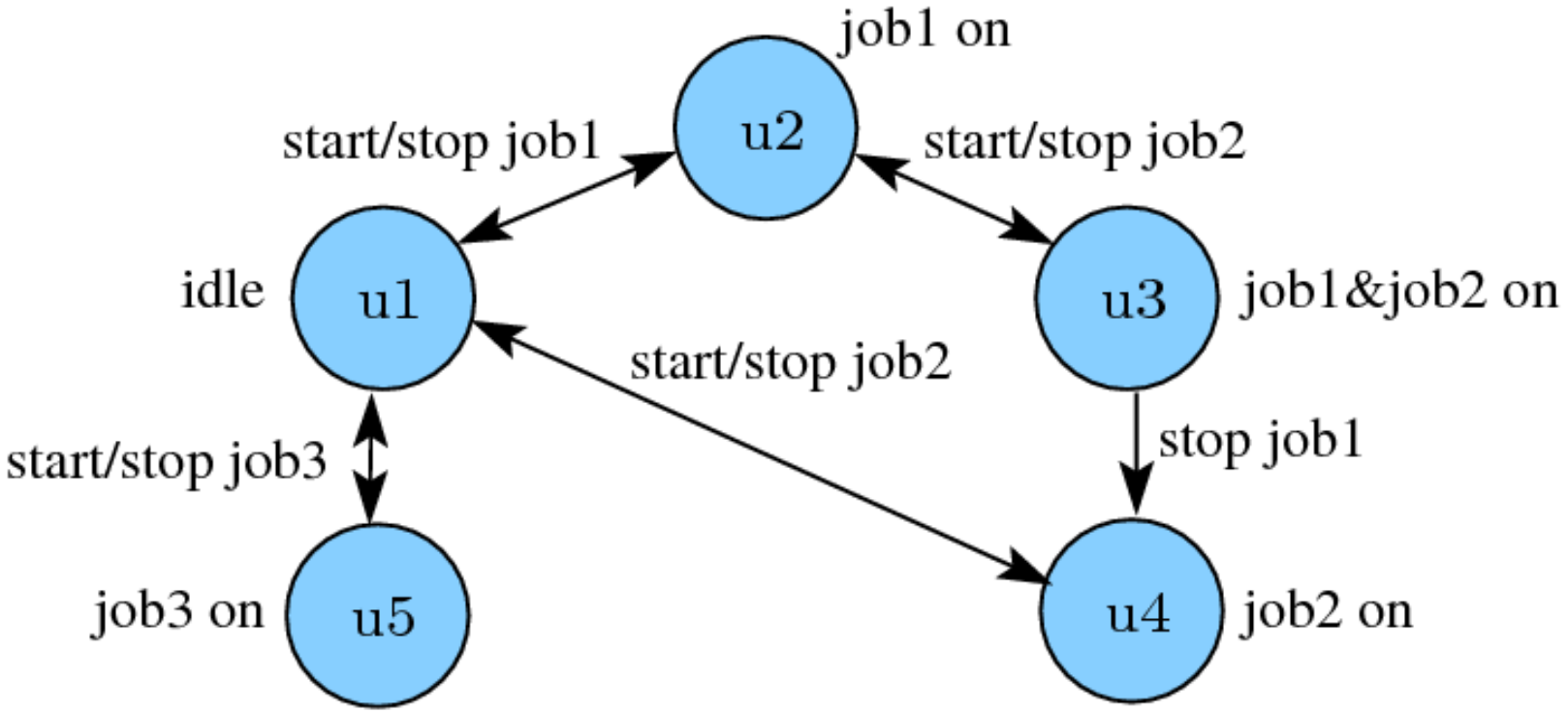
Is reconfiguration disruptive?
Must the resource manager be clairvoyant?

Disruptive use-case switching



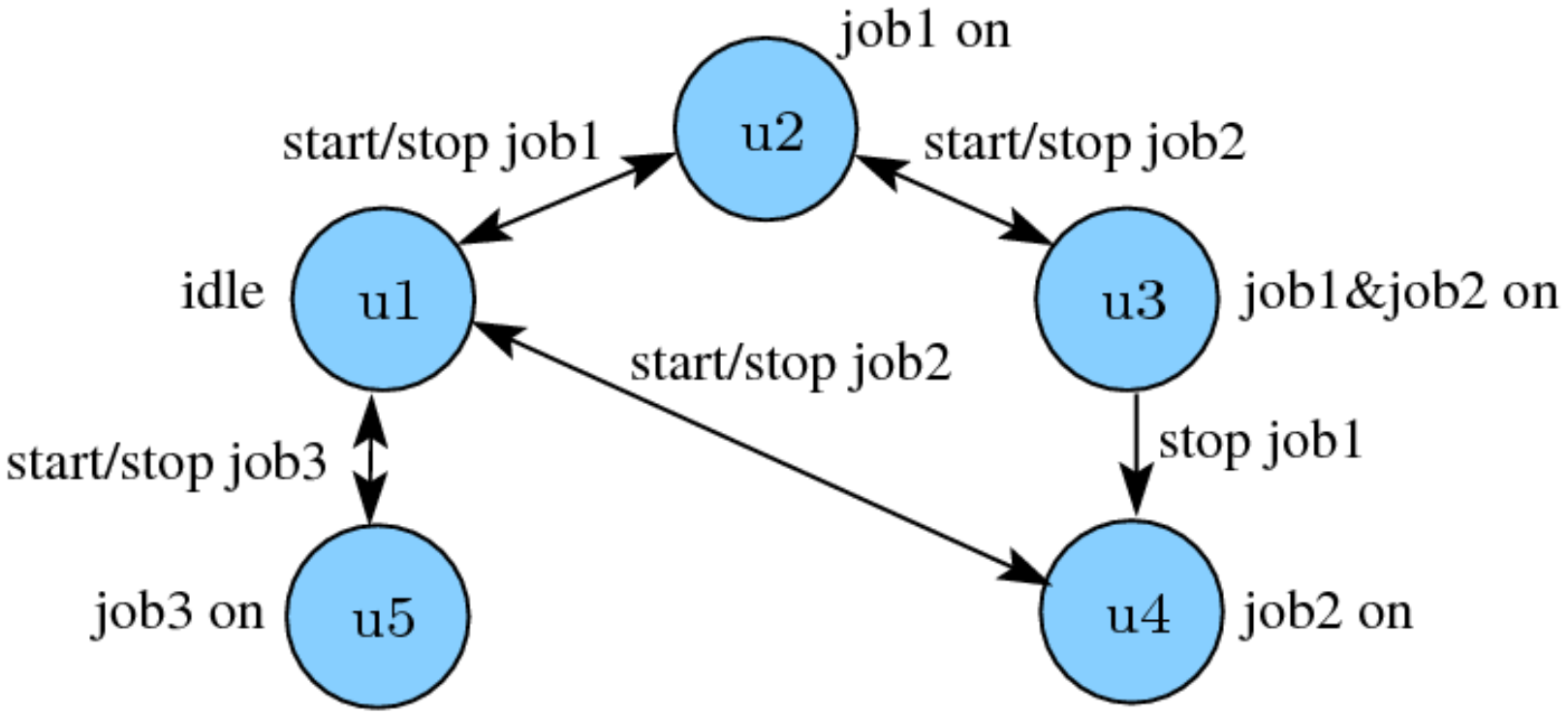
Clairvoyant resource manager

- Requirements resource manager:
  - System specification defines the set of supported use-cases
  - System specification defines maximum time for a use-case switch



Verification effort

- Observation: number of use-cases grows exponentially
- Budget reservation enables verification per job
- We compute budgets with dataflow analysis techniques



Budget allocation

For each job compute budget

- Try to find for each use-case a mapping given the computed budgets of the jobs

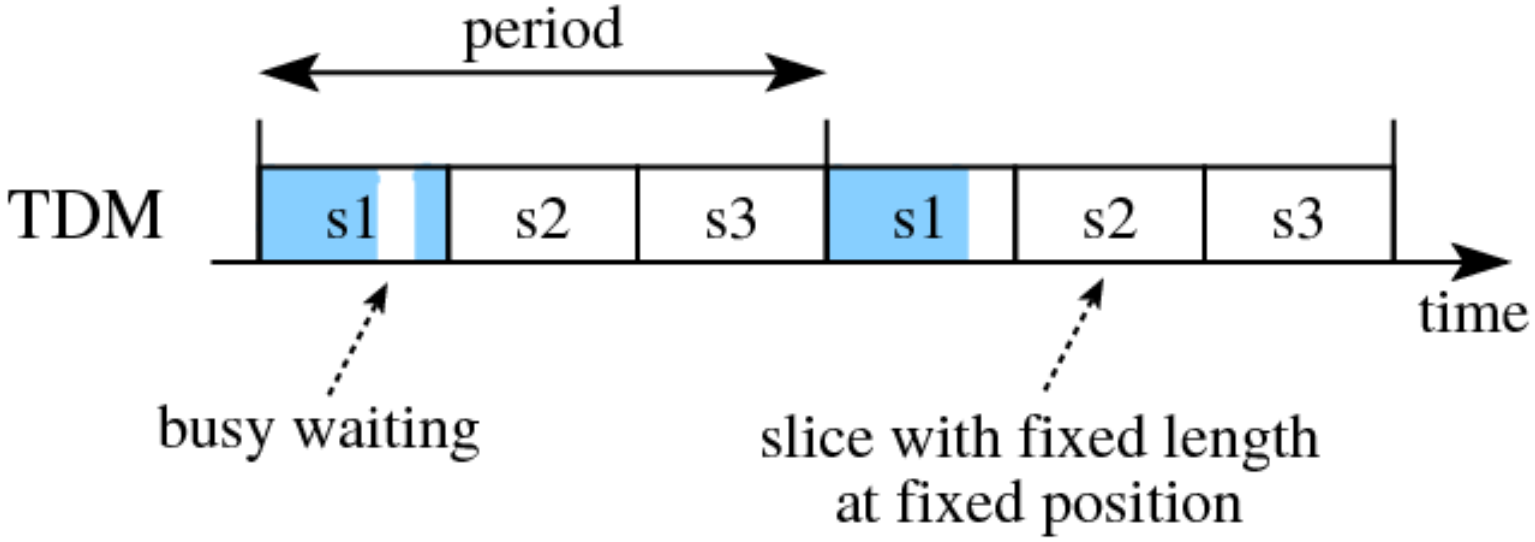
Redistribute budgets between jobs?

- Benefit: potentially a larger number of use-cases can be supported by same hardware
- Holistic approach, does not allow incremental design, can be very difficult

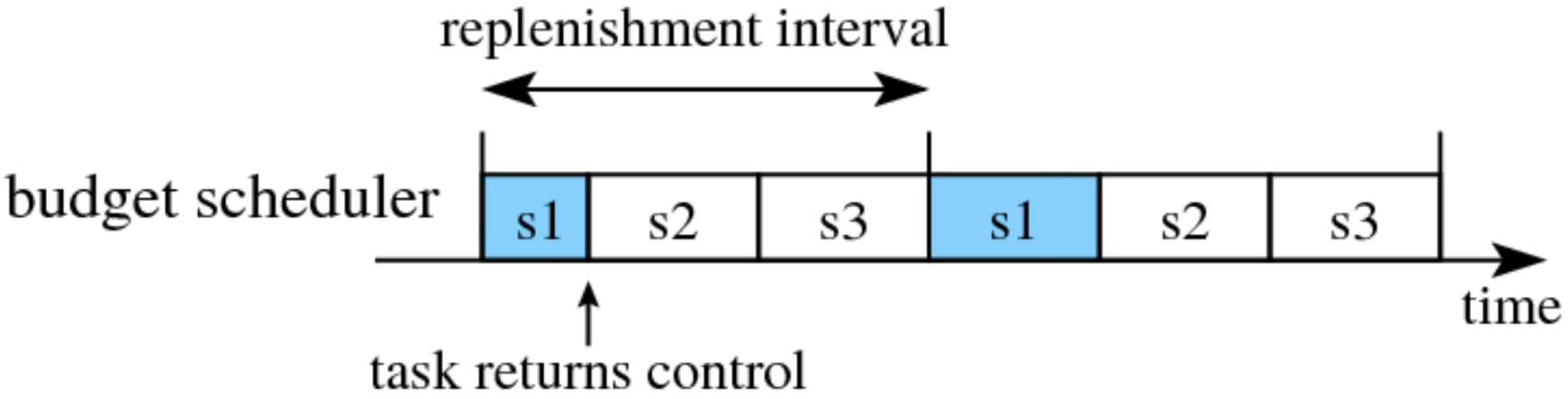
Budget schedulers

Resource reservation with budget schedulers

- Guarantee a minimum resource budget in an interval of time
  - Consequence *worst-case response time* of a task is independent of other tasks
- A time division multiplex scheduler is an example of a budget scheduler

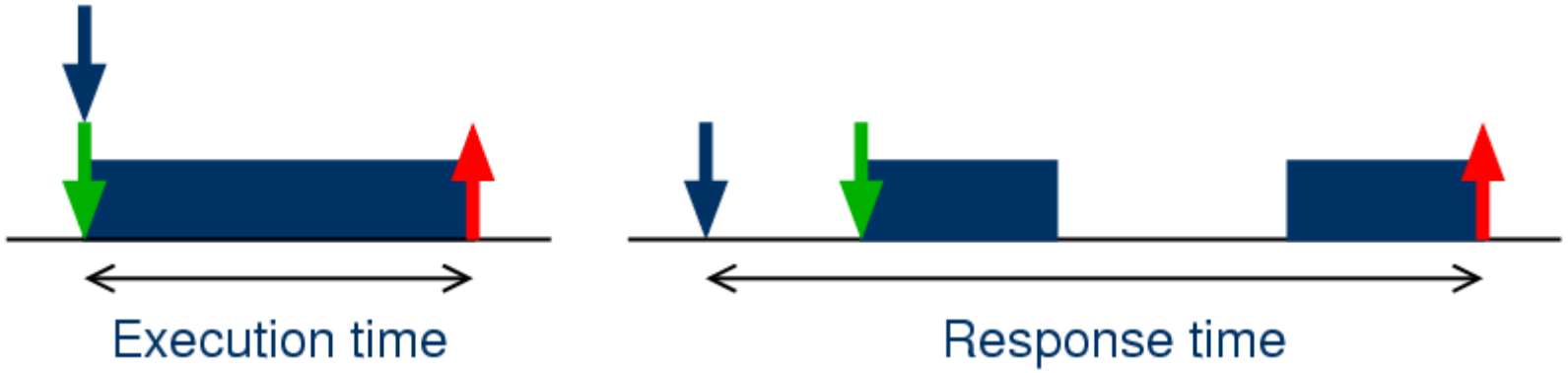


Work-conserving budget scheduler



Response times of tasks

- Enabling time : sufficient data and space is available in buffers
- **Execution time** of a task == time between enabling and finish
  - Execution in isolation
  - Enabling time == start time
- **Response time** of a task == time between enabling and finish
  - Resource is shared
  - Enabling time ≠ start time



Run-time scheduling

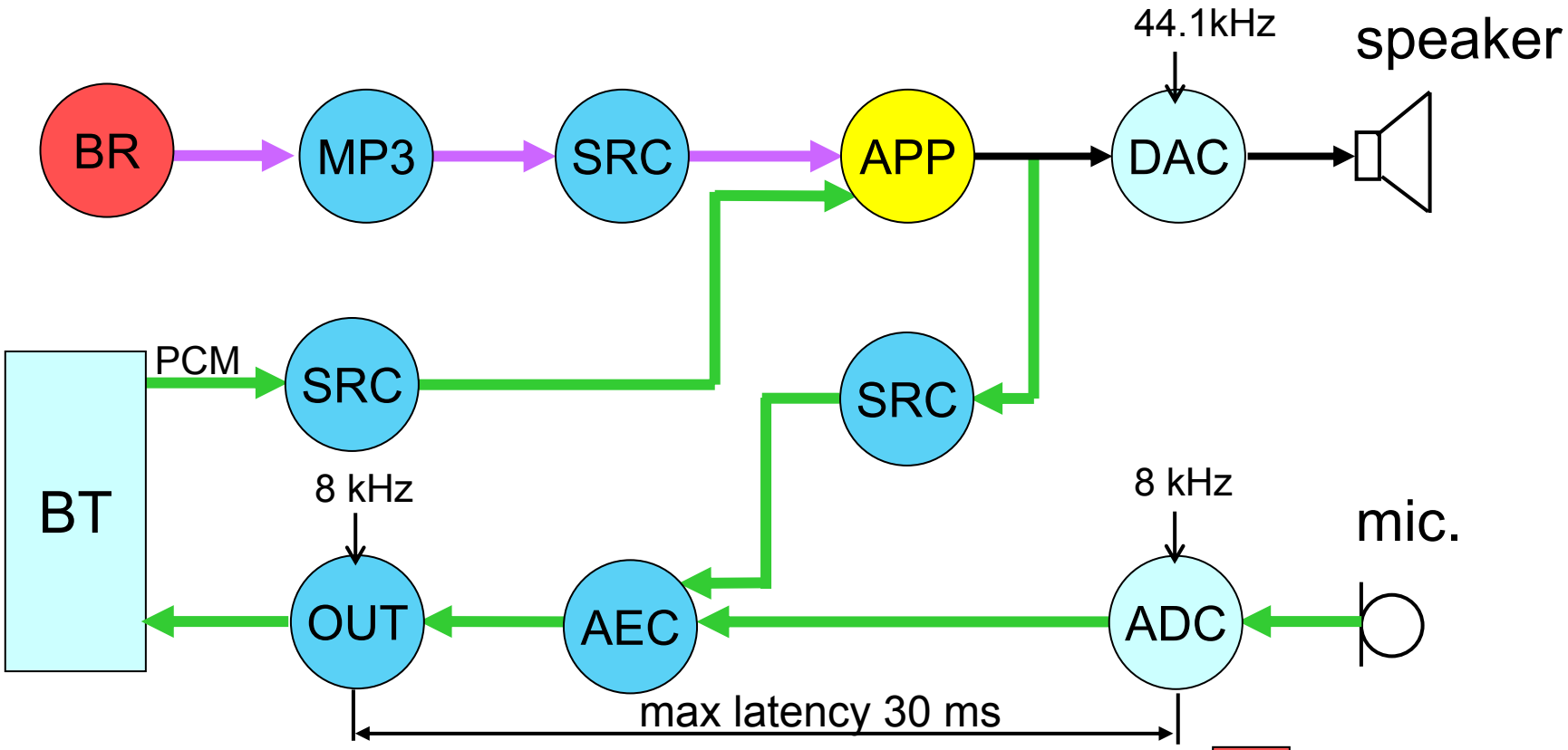
- Response time depends on
  - Execution time
  - Interference from other tasks
- Interference can depend on
  - Number of activations of other tasks
  - Execution times of other tasks
- Leads to three types of schedulers
  1. RT depends on activations & execution times
  2. RT depends on execution times
  3. RT independent

Run-time scheduling (cont.)

1. Dependence on: activations & execution times
   - Classic single-processor real-time schedulers
   - E.g. static priority pre-emptive
2. Dependence on: execution times
   - Latency-rate servers
   - E.g. round-robin
3. ***Independent: interference bounded by construction***
   - Budget schedulers
   - E.g. time-division multiplex

Throughput analysis
(given budgets what's the throughput?)

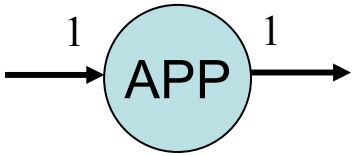
Audio echo cancellation example



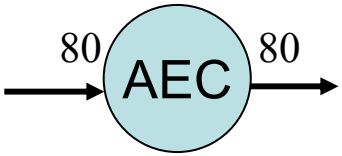
- multiple jobs and streams
- real-time constraints: throughput + latency
- PCM stream has external clock
- starting & stopping streams without clicks

Advanced task models are needed in the audio echo cancellation application

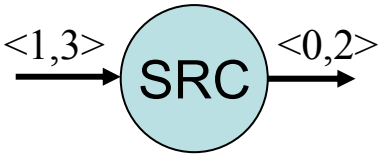
Single rate (SRDF)



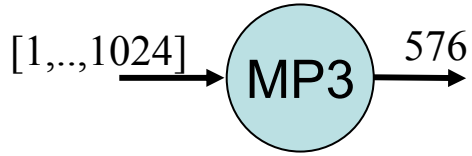
Multi rate (MRDF)



Cyclo static (CSDF)

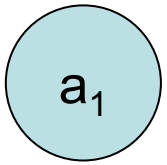


Variable rate (VRDF)



Single-rate dataflow analysis example

Elements of a single-rate dataflow graph



Nodes denote actors

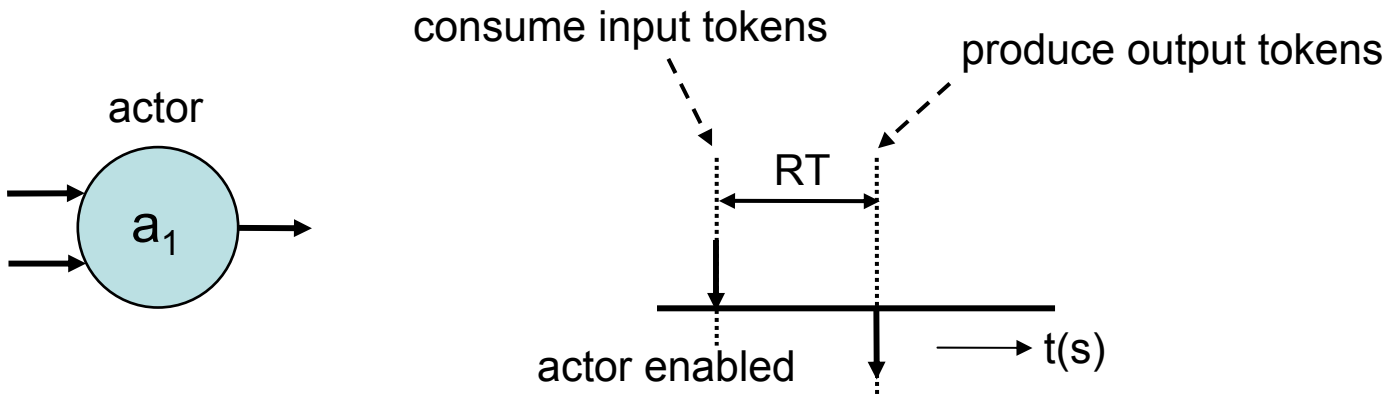
Edges denote unbounded queues

Dots denote tokens

Firing rule = enabling condition

Response time (RT) = interval between enabling and finish

Characteristics of a dataflow actor

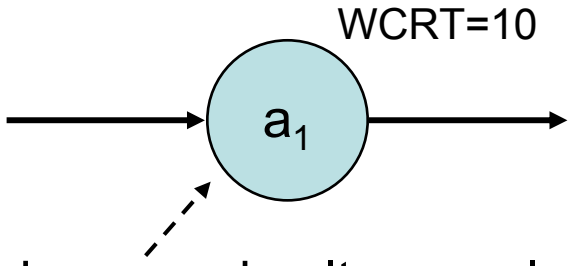


An actor:

- can represent a quantum of work
- has a firing rule (for single rate dataflow: a token on each input)
- is enabled if the firing condition is satisfied
- is stateless
- consumes input tokens when the actor starts
- produces output tokens in zero time when actor finishes its execution

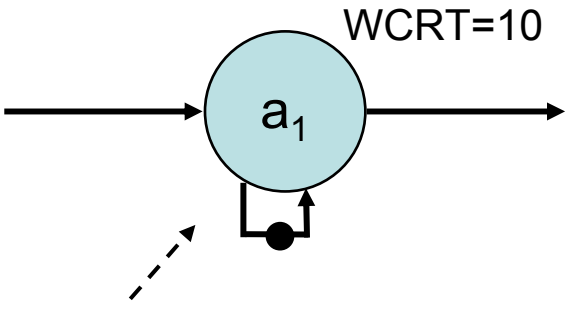
Auto-concurrency (overlapping execution)

Assume two tokens arrive at t=0



two tokens produced at t=10

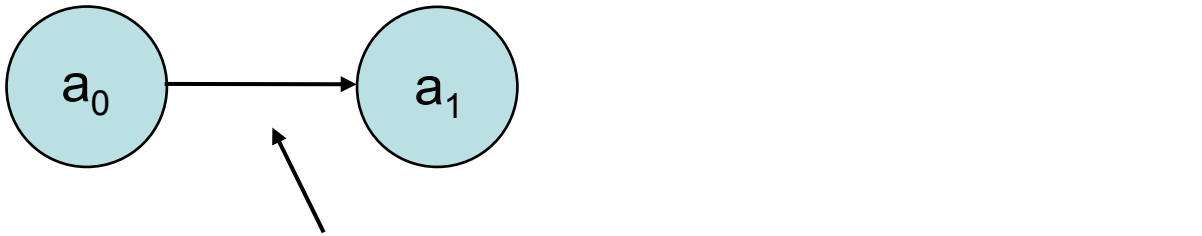
multiple executions happen simultaneously



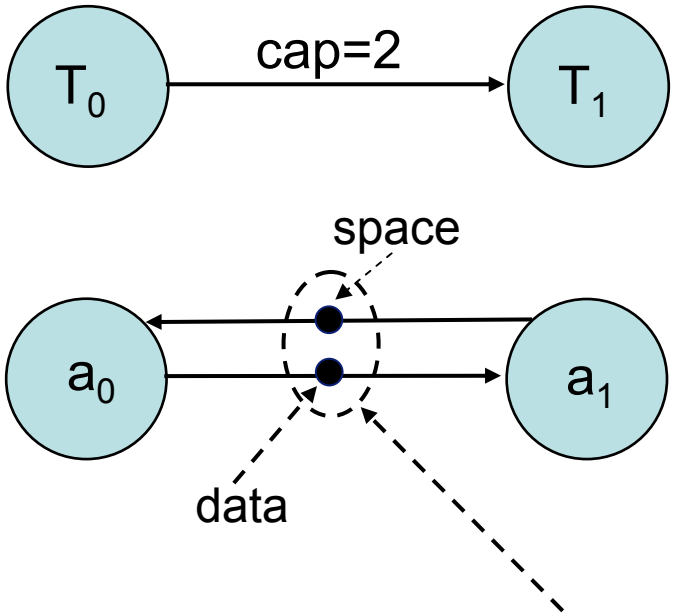
tokens produced at t=10, t=20

next execution cannot start before previous has finished

Unbounded queue



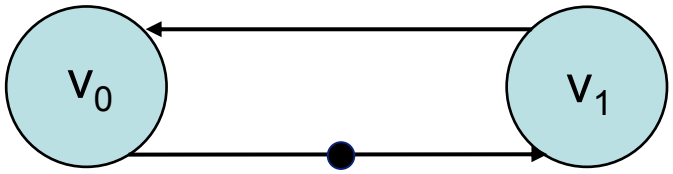
Bounded FIFO model



Number of tokens on the cycle equals the FIFO capacity

Monotonicity

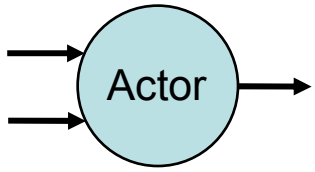
- Monotonic temporal behavior:
  - An earlier production of a token cannot result in a later start of an actor during self-timed execution
- Consequence:
  - Sufficient to show that a schedule exist that satisfies the throughput and latency constraints given worst-case response times
  - Smaller response time result in earlier arrival tokens
    - Scheduling anomalies do not occur during self-timed execution of a dataflow model
- Requires sequential firing rules [Lee & Parks 1995]

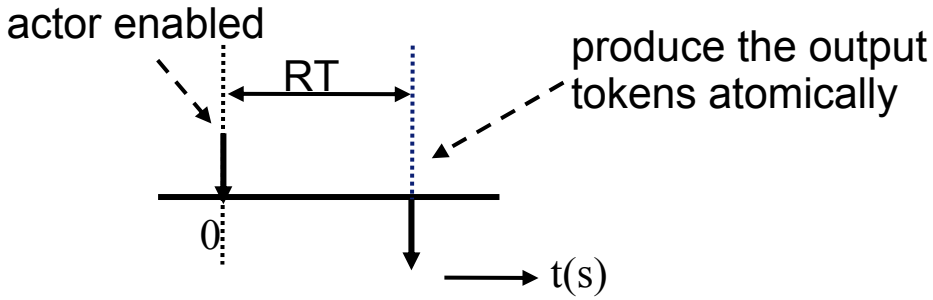


Earlier arrival token results in earlier start v_1 and v_0

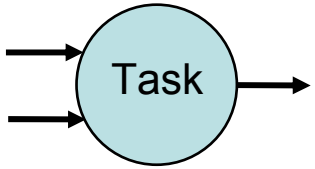
Tasks versus dataflow actors

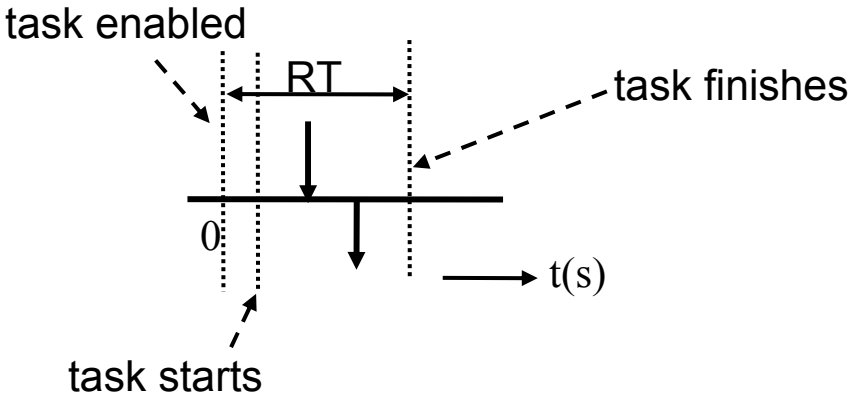
dataflow model





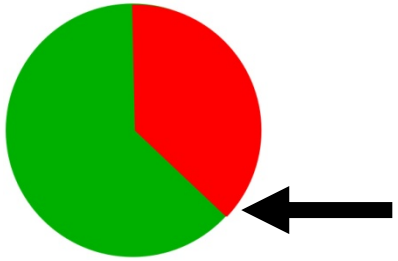
task graph





Response time calculation

- Time-division multiplex (TDM) is a budget scheduler
- Classical response time computation
  - Independent of arrival times
  - Assumes worst-case enabling time

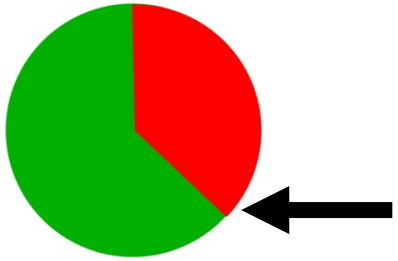


Worst-case enabling time (TDM)

Response time calculation

- Time-division multiplex (TDM) is a budget scheduler
- Classical response time computation
  - Independent of arrival times
  - Assumes worst-case enabling time

$$wcrt = wcet + (P - B)\left\lceil \frac{wcet}{B} \right\rceil$$

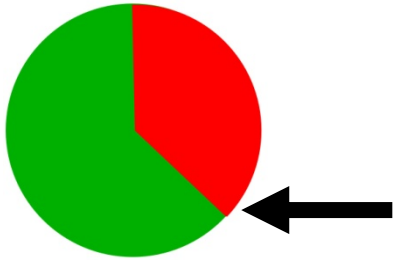


Worst-case enabling time (TDM)

Response time calculation

- Time-division multiplex (TDM) is a budget scheduler
- Classical response time computation
  - Independent of arrival times
  - Assumes worst-case enabling time

$$wcrt = wcet + (P - B)\left\lceil \frac{wcet}{B} \right\rceil$$



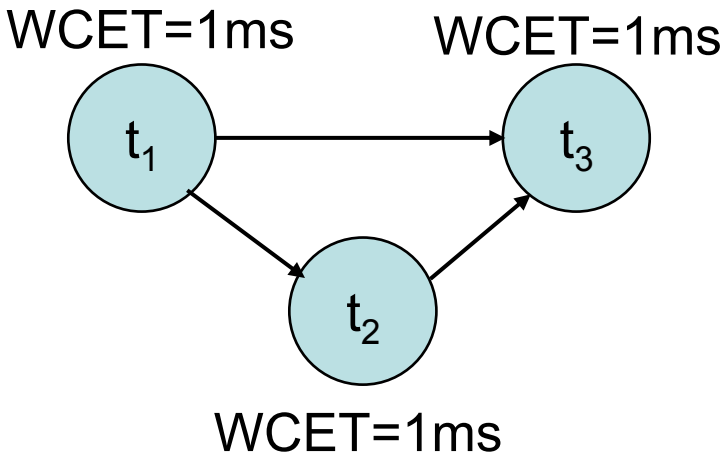
Worst-case enabling time (TDM)

- Upper bound on finish time
  - Independent of previous finish time

$$f(i) = e(i) + wcet + (P - B)\left\lceil \frac{wcet}{B} \right\rceil$$

Simple throughput analysis example

Task-graph

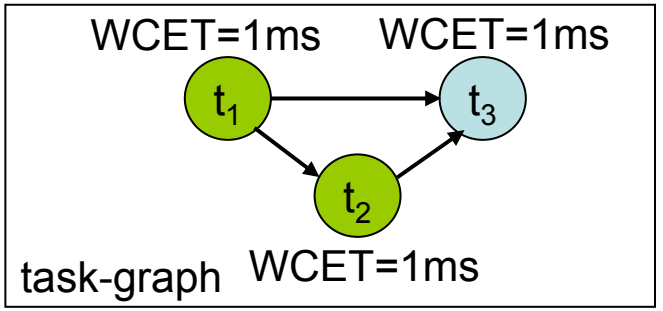


Assume:

- t_1 and t_2 share one processor, each task get a TDM-slice of 1 ms every 2 ms
- Each task produce and consume one token per execution
- Capacity of each buffer is 2 tokens

What is the minimum throughput?

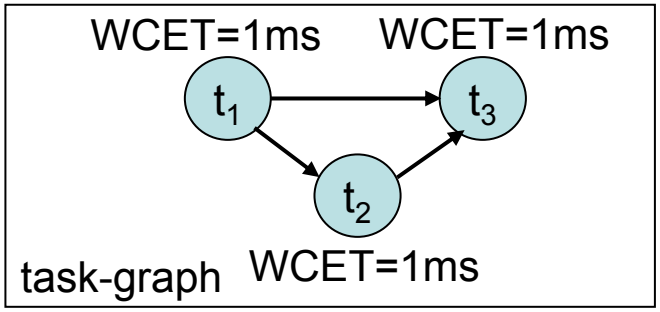
Worst-case response time



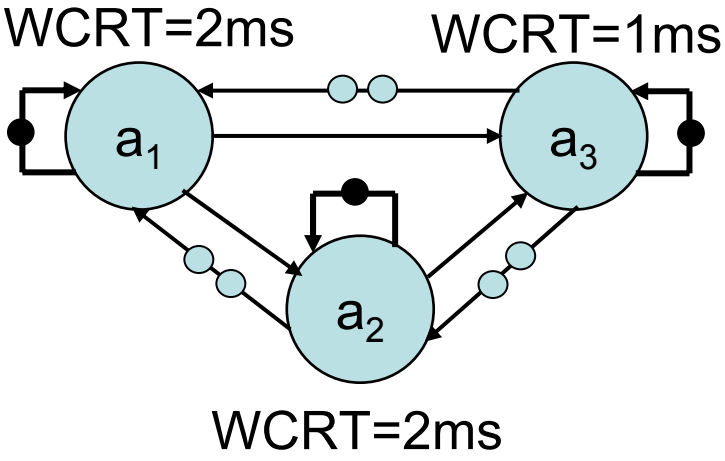
$$wcrt_1 = wcet_1 + (P - B)\lceil \frac{wcet_1}{B} \rceil$$

$$wcrt_1 = 1 + (2 - 1)\lceil \frac{1}{1} \rceil = 2ms$$

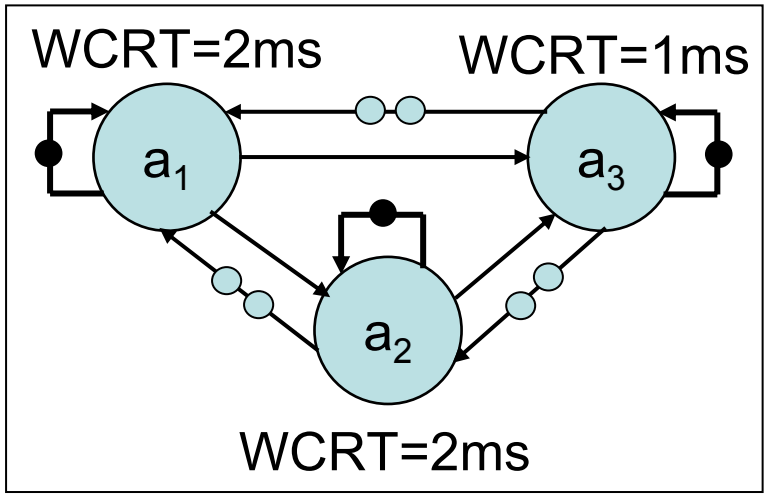
Throughput analysis

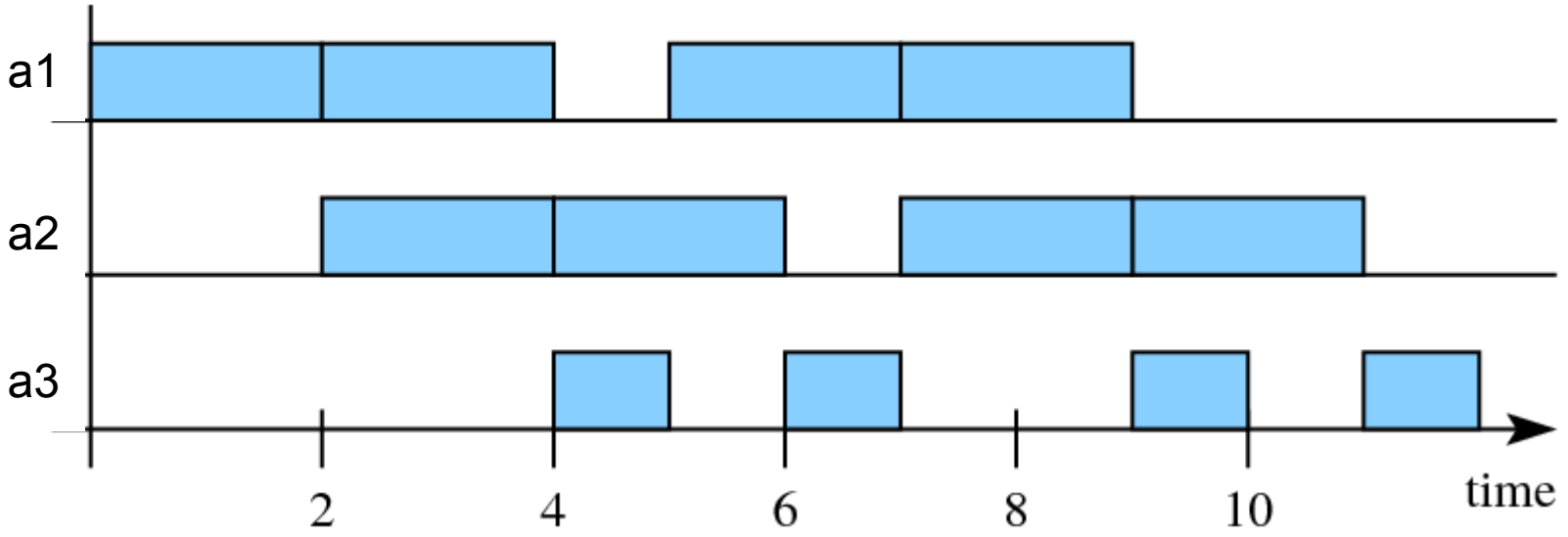


Dataflow graph



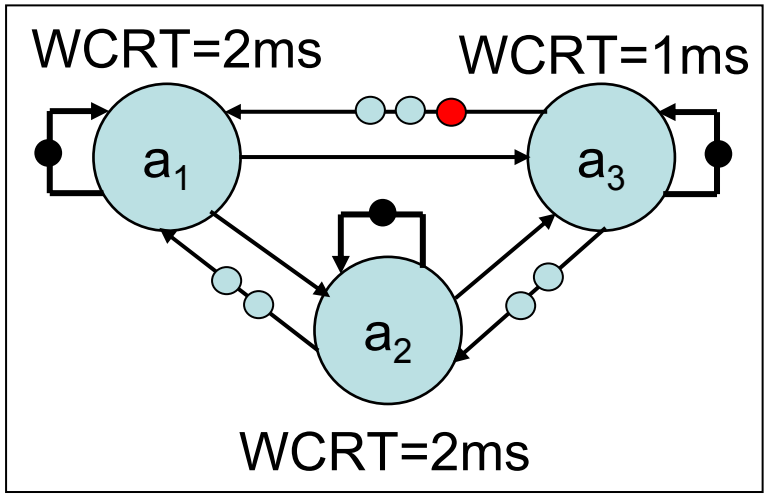
Valid schedule (1)

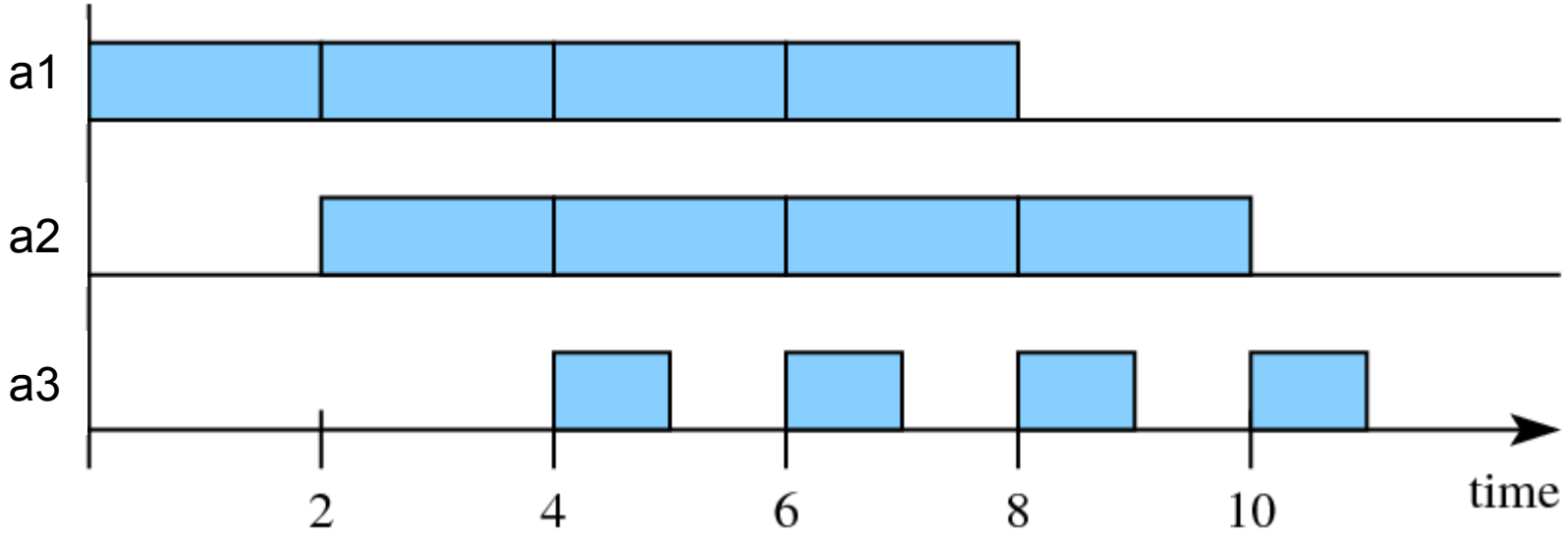




1/Throughput = 2.5 ms/token

Valid schedule (2)

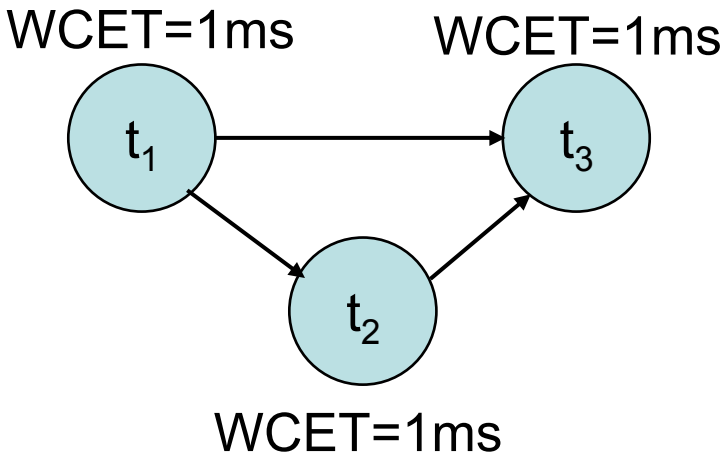




1/Throughput = 2 ms/token $\Rightarrow$ buffer capacity affects throughput!

Throughput analysis

Task-graph

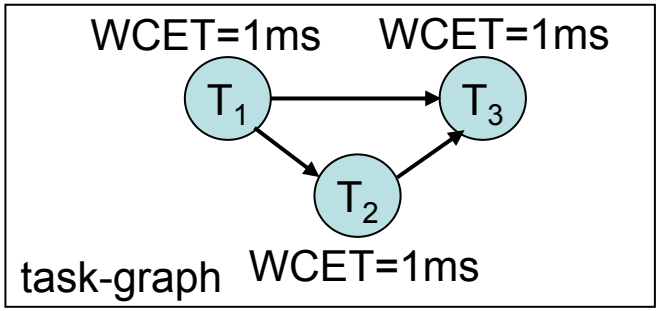


Assume:

- t_1 and t_2 share one processor, each task get a TDM-slice of 4 ms every 8 ms
- Capacity of each buffer is 2 tokens

What is the minimum throughput?

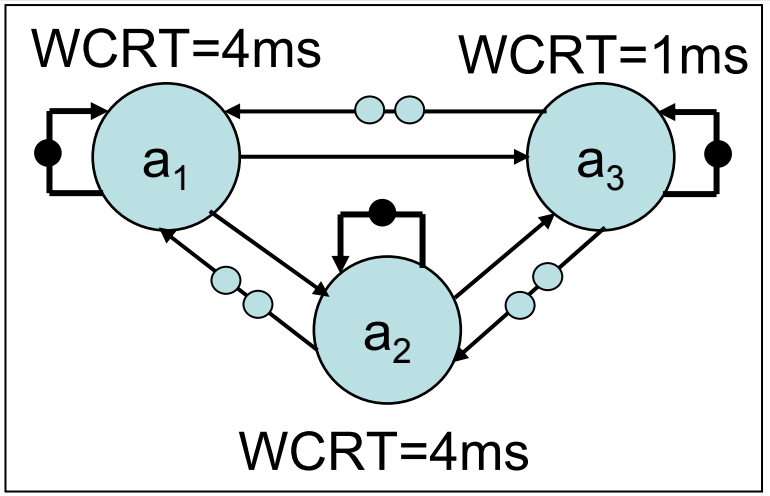
Worst-case response time

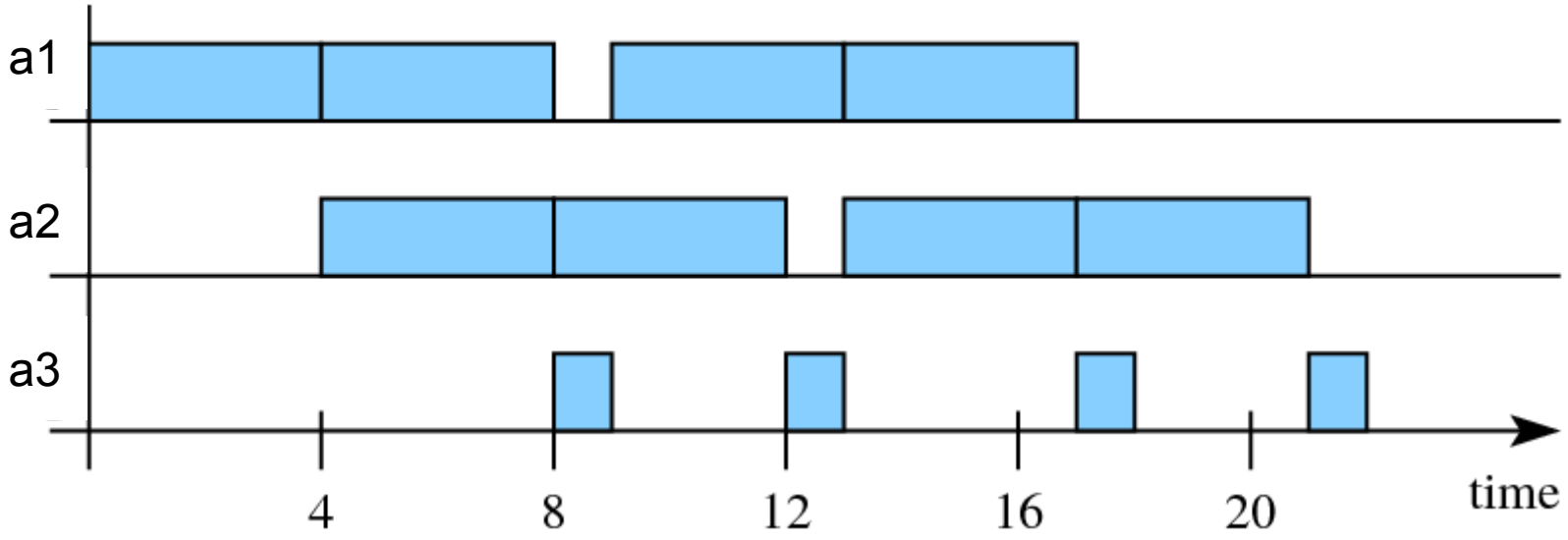


$$wcrt_1 = wcet_1 + (P - B)\lceil \frac{wcet_1}{B} \rceil$$

$$wcrt_1 = 1 + (4 - 1)\lceil \frac{1}{1} \rceil = 4ms$$

Valid schedule





1/Throughput = 4.5 ms/token

Improved response time calculation

- Traditional model
  - Does not capture multiple consecutive executions in one slice
  - Correct from a latency point of view
  - Too pessimistic from a throughput point of view
- If you know that enabling time is before previous finish time, then you do not always need to assume the initial pre-emption

Latency-Rate servers. Stiliadis and Varma. 1998

Improved response time calculation

- Traditional model
  - Does not capture multiple consecutive executions in one slice
  - Correct from a latency point of view
  - Too pessimistic from a throughput point of view
- If you know that enabling time is before previous finish time, then you do not always need to assume the initial pre-emption

$$f(i) = \max(e(i) + P - B, f(i-1)) + P\frac{x(i)}{B}$$

Improved response time calculation

- Traditional model
  - Does not capture multiple consecutive executions in one slice
  - Correct from a latency point of view
  - Too pessimistic from a throughput point of view
- If you know that enabling time is before previous finish time, then you do not always need to assume the initial pre-emption

$$f(i) = \max(e(i) + P - B, f(i-1)) + P\frac{x(i)}{B}$$

Worst-case enabling time
+
initial pre-emption

Improved response time calculation

- Traditional model
  - Does not capture multiple consecutive executions in one slice
  - Correct from a latency point of view
  - Too pessimistic from a throughput point of view
- If you know that enabling time is before previous finish time, then you do not always need to assume the initial pre-emption

$$f(i) = \max(e(i) + P - B, f(i-1)) + P\frac{x(i)}{B}$$

Worst-case enabling time
+
initial pre-emption

Previous finish

Improved response time calculation

- Traditional model
  - Does not capture multiple consecutive executions in one slice
  - Correct from a latency point of view
  - Too pessimistic from a throughput point of view
- If you know that enabling time is before previous finish time, then you do not always need to assume the initial pre-emption

$$f(i) = \max(e(i) + P - B, f(i-1)) + P\frac{x(i)}{B}$$

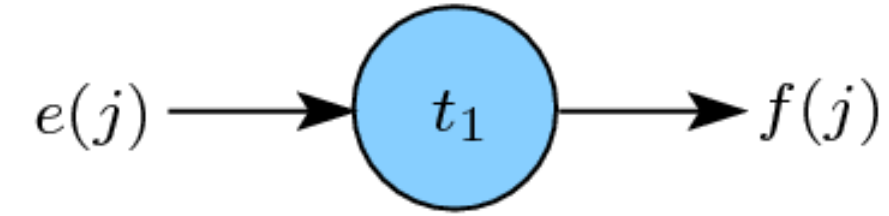
Worst-case enabling time + initial pre-emption

Previous finish

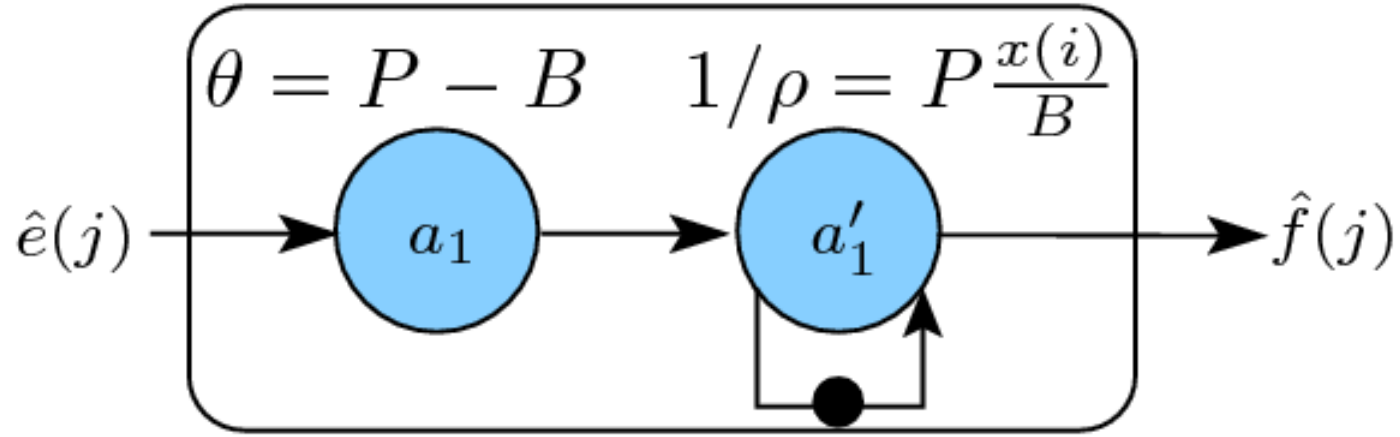
Execution on a P/B times slower processor

Conservative dataflow model

task graph

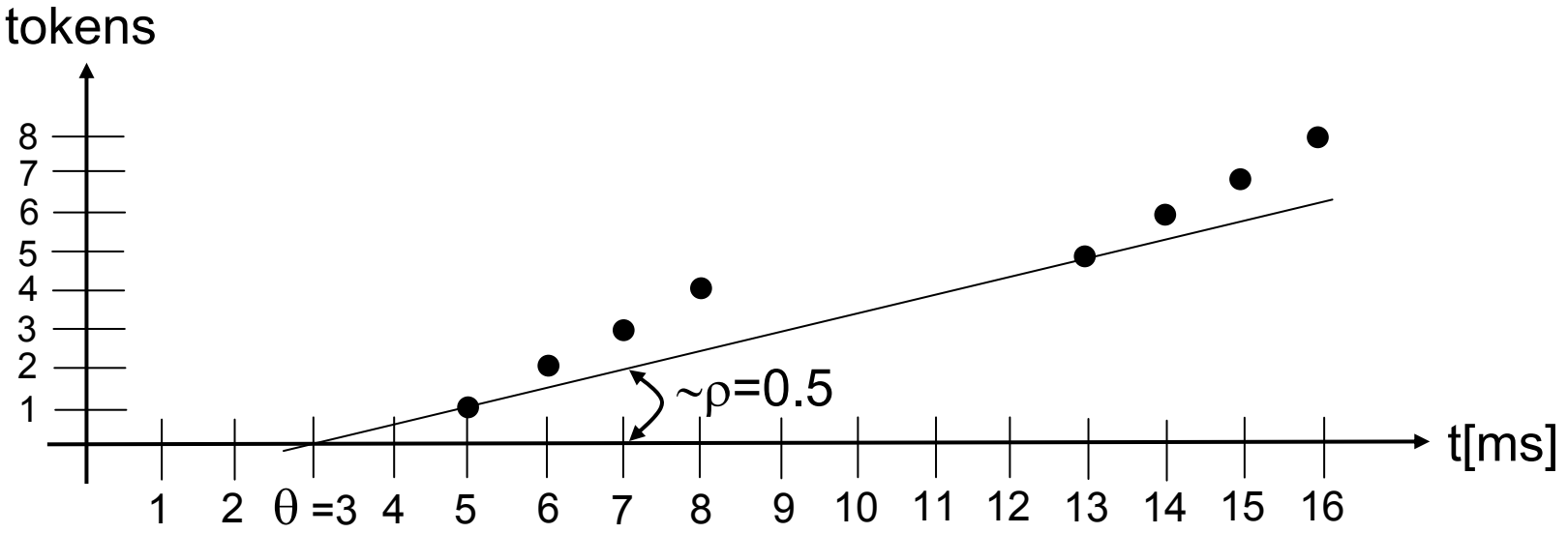


dataflow graph



$$e(j) \leq \hat{e}(j) \Rightarrow f(j) \leq \hat{f}(j)$$

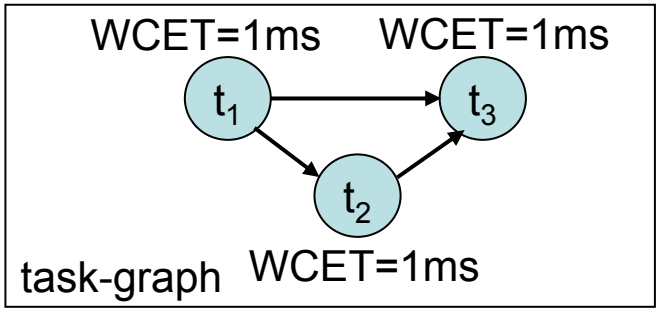
Latency-rate characterization



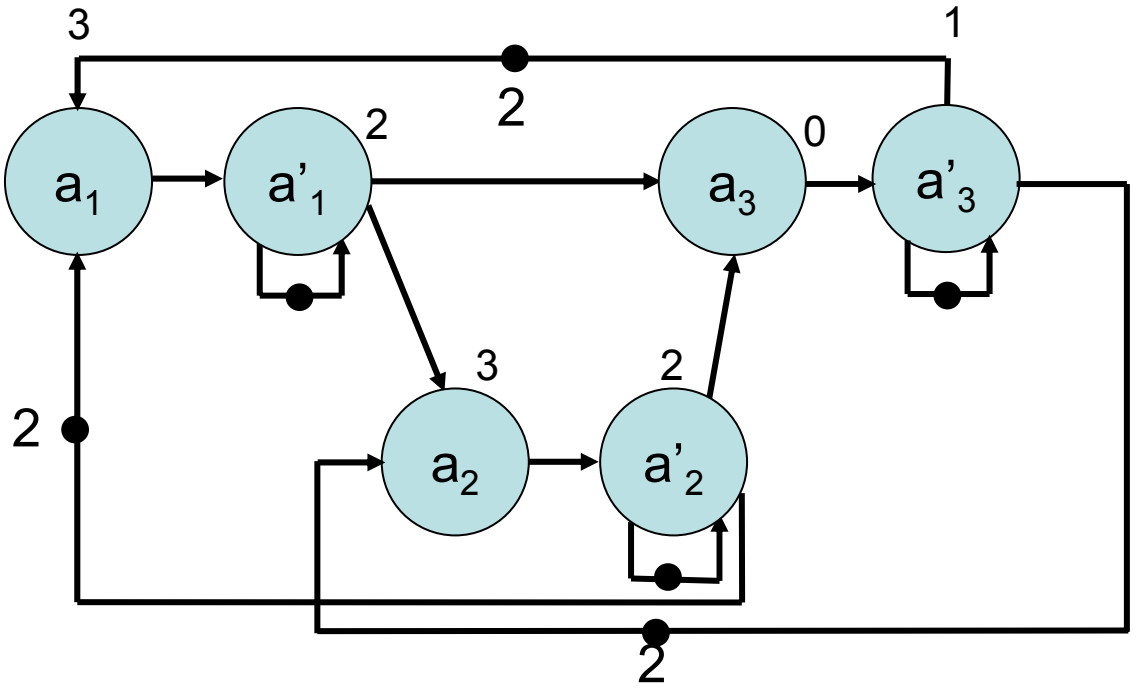
- Latency θ=(P-B) =4-1=3ms
- Rate ρ=B / x(i)P = 4/8=0.5 tokens/ms = 0.5 executions/ms

Throughput analysis

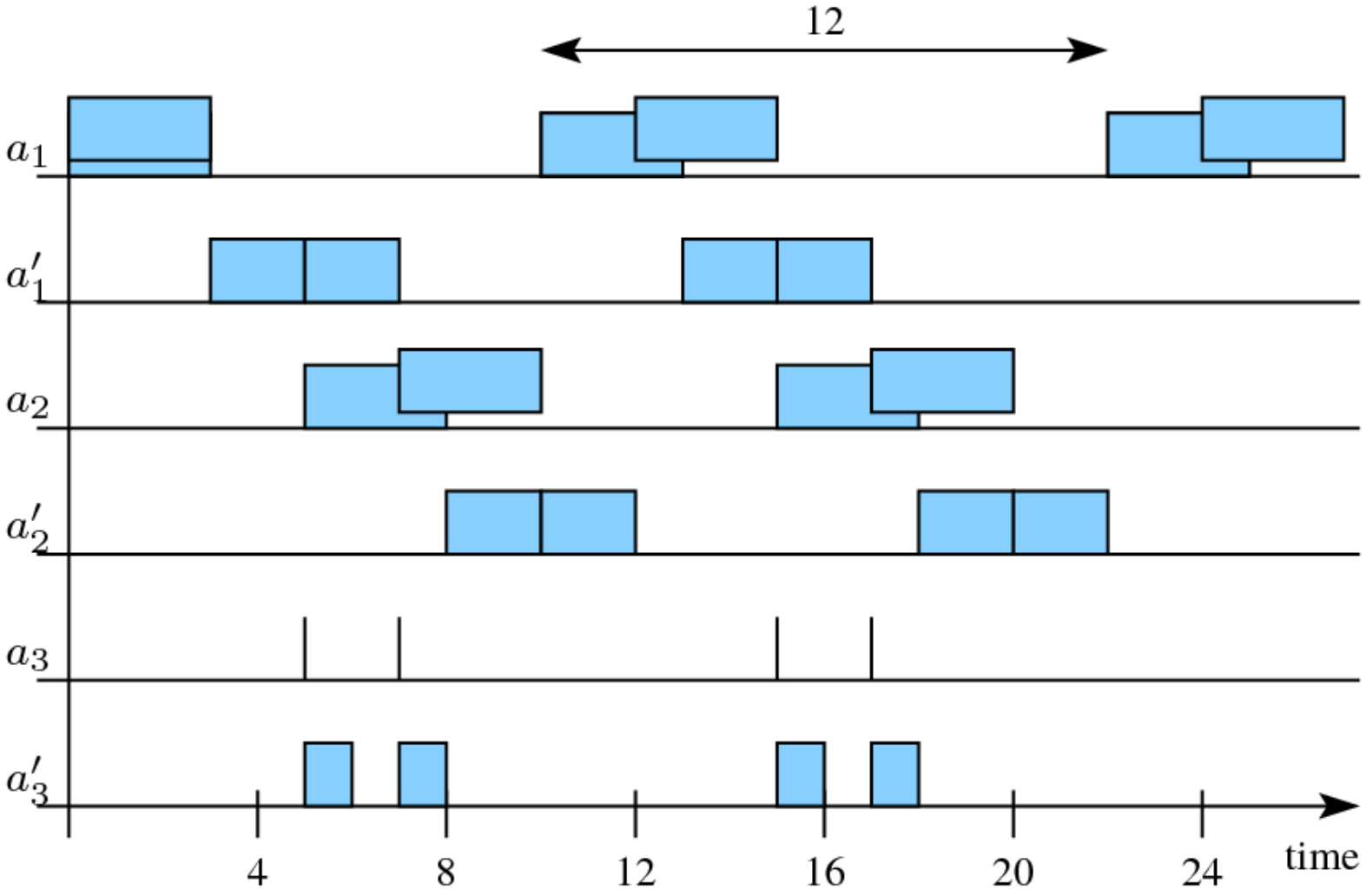
TDM: ρ=0.5 execution/ms, θ=3 ms



Dataflow model



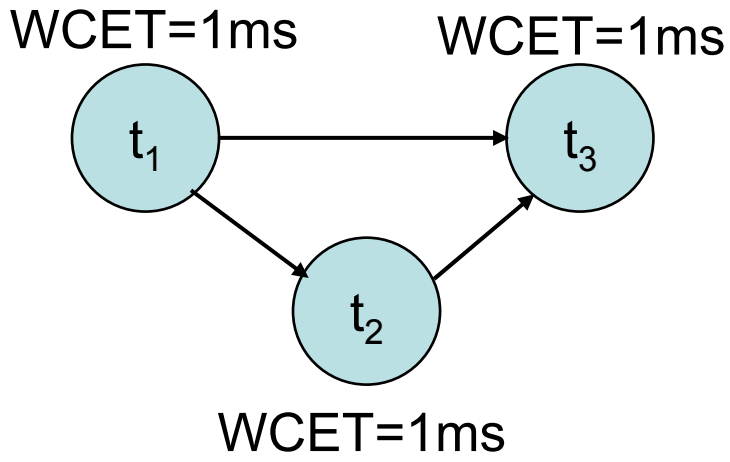
Valid schedule



1/Throughput = 12/2 =6 ms/token

Throughput analysis

Task-graph

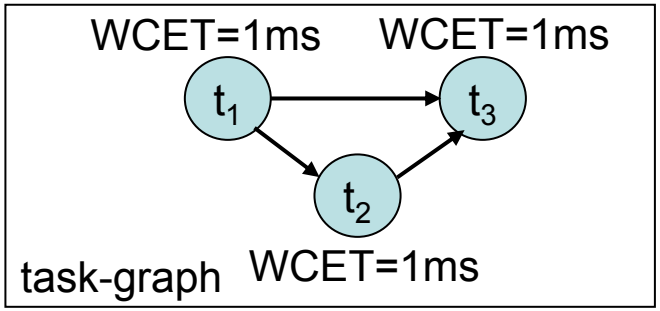


Assume:

- T_1 and T_2 share one processor, each task get a TDM-slice of 4 ms every 8 ms
- Infinite buffer capacity

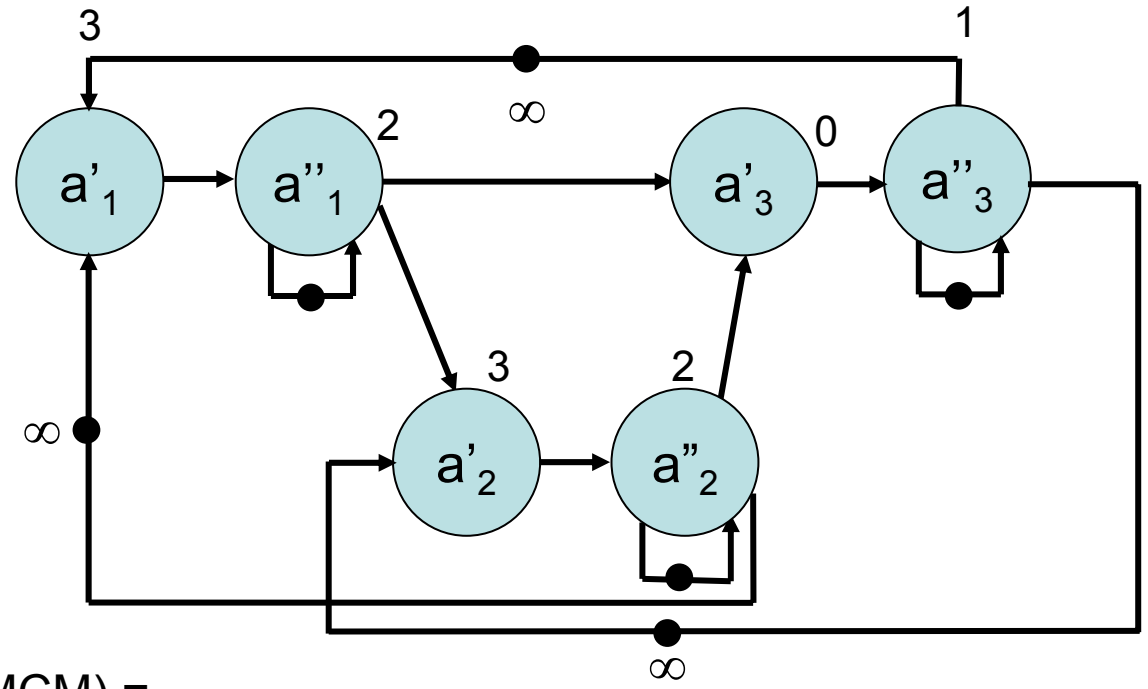
What is the minimum throughput?

Throughput analysis



TDM: ρ=0.5 execution/ms, θ=3 ms

Dataflow model

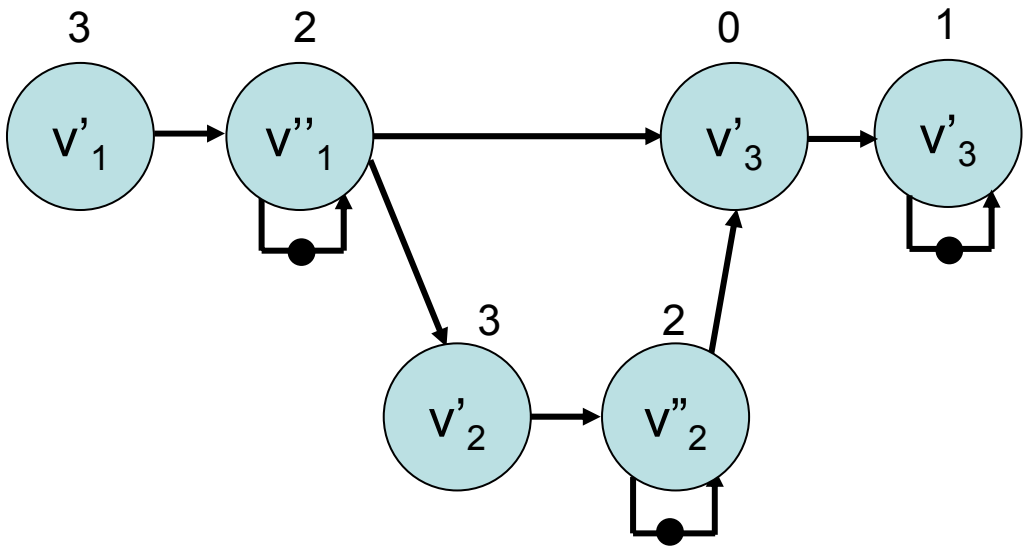


Maximum cycle mean (MCM) =

$\max_{c \in Cg}((\sum_{v \text{ on } c} WCRT(v))/ \text{tokens}) = 2/1 = 2$ ms/execution

Throughput analysis

Dataflow model

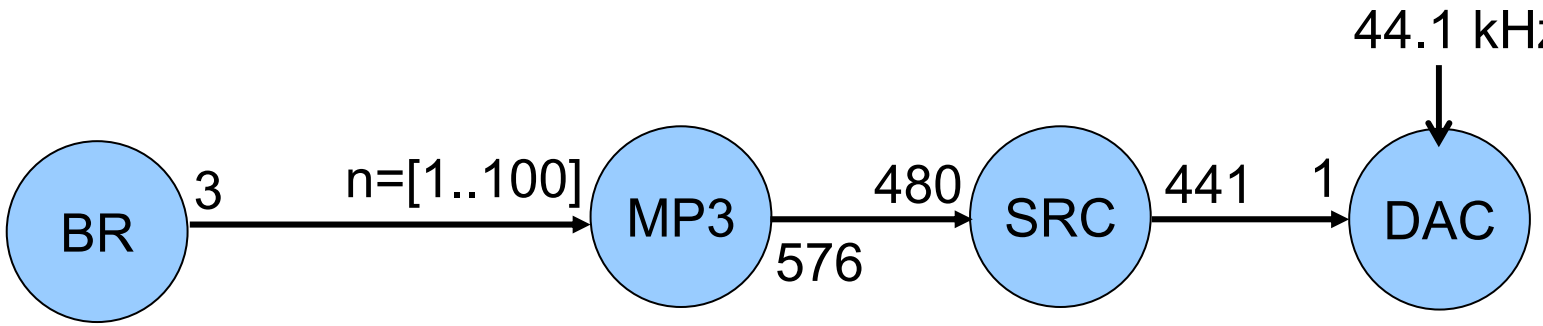


Maximum cycle mean (MCM) =

$\max_{c \in Cg} ((\sum_{v \text{ on } c} WCRT(v))/ \text{ tokens}) = 2/1 = 2$ ms/execution

Generalization: Variable rate dataflow

Recently introduced: variable rate dataflow

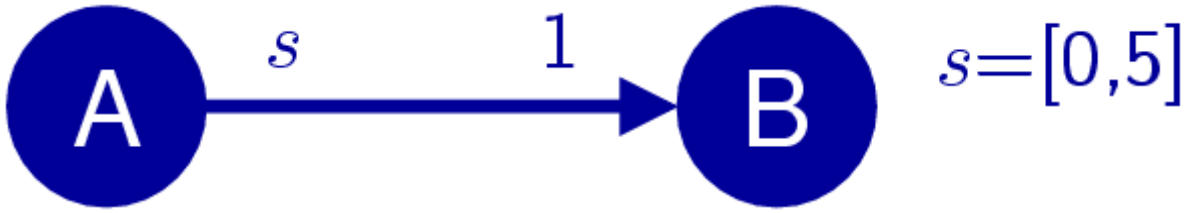


- MP3 decoder actor consumes a data-dependent amount of data
  - Schedule BR actor is a-periodic

[M. Wiggers et.al.,DATE 2008]

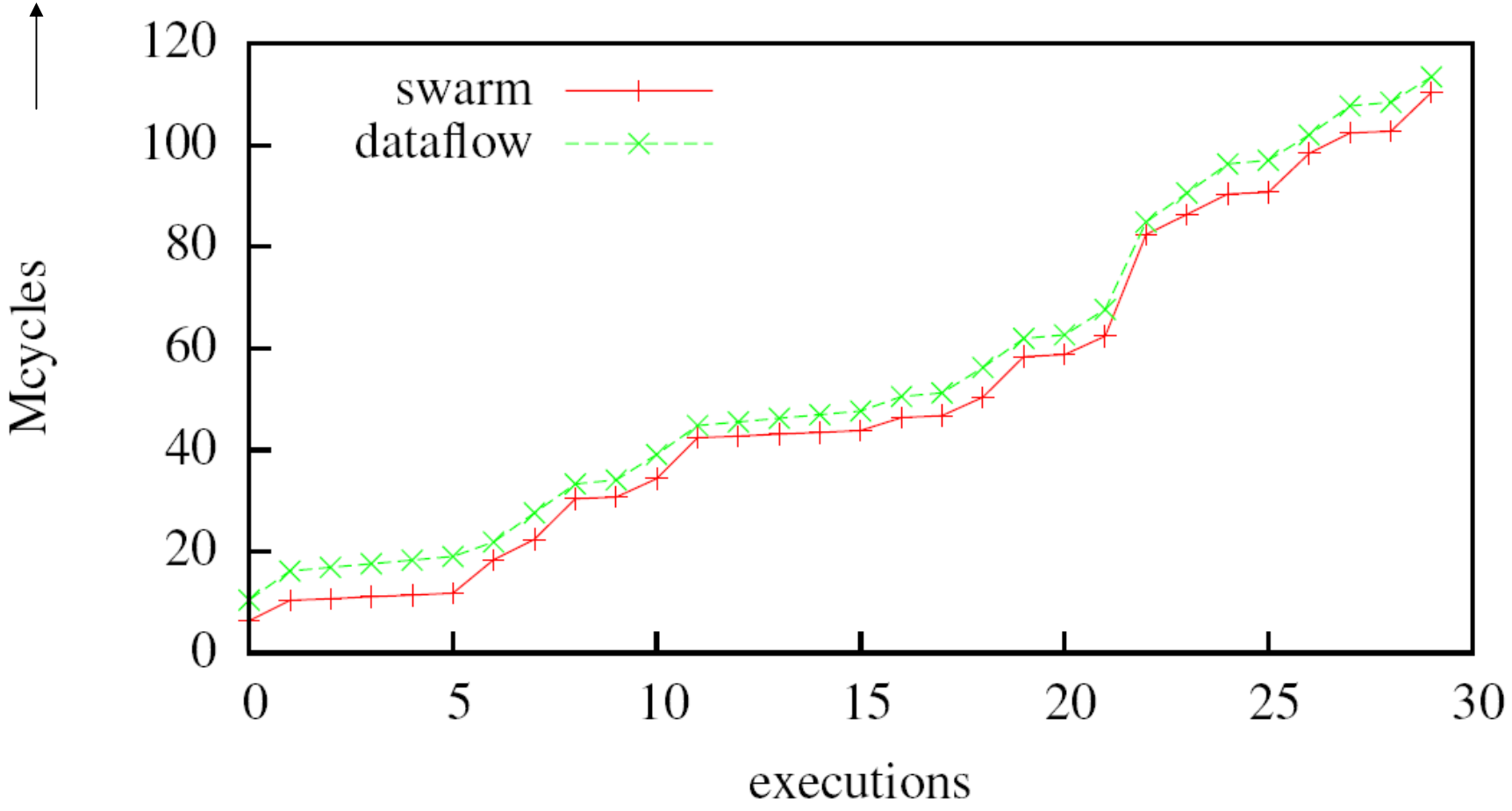
Experiment with VRDF

- Aim: test accuracy of dataflow model
- Set-up
  - Producer-consumer with variable production quantum
  - 2 ARM processors that share one double ported memory
  - Cycle-accurate systemC model (using Swarm)
  - Execution time producer > time slice producer
  - Execution time consumer << time slice consumer
  - Buffer capacity of 8 tokens



Experimental results

Finish times
task B



Summary

- Context:
  - Real-time multi-job stream processing on multiprocessor systems
- Objectives:
  - Real-time guarantees per job
  - Design and characterize each job in isolation
  - Guaranteed set of supported use-cases
- Approach
  - Use budget schedulers
  - Compute budgets with dataflow analysis techniques
- Dataflow analysis
  - Include effects budget schedulers in dataflow models
  - Results of experiment with variable rate dataflow

References

- B. Akesson, K. Goossens, and M. Ringhofer. Predator: A Predictable SDRAM Memory Controller. In *Proc. Int'l Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS),* 2007.
- M. Bekooij, A. Moonen, and J. van Meerbergen. Predictable and Composable Multiprocessor System Design: A Constructive Approach, In *Proc. Bits&Chips Symposium on Embedded Systems and Software*, October 2007, Eindhoven, The Netherlands.
- M. Bekooij, M.Wiggers, J. van Meerbergen, Efficient Buffer Capacity and Scheduler Setting Computation for Soft Real-Time Stream Processing Applications, In *Proc. Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, April 2007.
- J. Buck. Scheduling Dynamic Dataflow Graphs with Bounded Memory using the Token Flow Model. PhD thesis, University of Berkeley. 1993
- A. Kumar, A. Hansson, J. Huisken and H. Corporaal. An FPGA Design Flow for Reconfigurable Network-Based Multi-Processor Systems on Chip. *In Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE),* April 2007.

References

- A. Hansson, K.G.W. Goossens, M.J.G. Bekooij and J. Huisken. CoMPSoC: A Composable and Predictable Multi-Processor System on Chip Template. *ACM Transactions on Design Automation of Electronic Systems*. *To appear* E.A. Lee. *Consistency in Dataflow Graphs*. IEEE Transactions on Par. and Distr. Systems. 1991
- E.A. Lee and T. Parks. *Dataflow Process Networks*. Proc. of the IEEE. May 1995
- A. Moonen et.al. A Multi-Core Architecture for In-Car Digital Entertainment, GSPX publication 24-27 October 2005: In *Proc. Int'l Conference on Global Signal Processing (GSPx),* October 2005.
- O. Moreira and M. Bekooij. Self-Timed Scheduling Analysis for Real-Time Applications, In *EURASIP Journal on Advances in Signal Processing, 2007*
- O. Moreira and M. Bekooij. Scheduling Multiple Independent Hard Real-Time Jobs on a Heterogeneous Multiprocessor, In *Proc. Int'l Conference on Embedded Software (EMSOFT), September* 2007
- S. Sriram and S. S. Bhattacharyya. Embedded Multiprocessors: Scheduling and Synchronization. Marcel Dekker Inc., 2000

References

- D. Stiliadis and A. Varma. Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms. In *IEEE/ACM Transactions on Networking*, 6(5):611–624, October 1998.
- S. Stuijk, T. Basten, M.C.W. Geilen and H. Corporaal. Multiprocessor Resource Allocation for Throughput-Constrained Synchronous Dataflow Graphs, In *Proc. Design Automation Conference (DAC),* June 2007.
- M. Wiggers, M. Bekooij, and G. Smit. Modelling Run-Time Arbitration by Latency-Rate Servers in Data Flow Graphs. In *Proc. Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, April 2007.
- M. Wiggers, M. Bekooij, P. Jansen, and G. Smit. Efficient Computation of Buffer Capacities for Cyclo-Static Real-Time Systems with Back-Pressure. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2007.
- M.H. Wiggers, M.J.G. Bekooij and G.J.M. Smit. Computation of Buffer Capacities for Throughput Constrained and Data-Dependent Inter-Task Communication. In *Proc. DATE*. April 2008
- M.H. Wiggers, M.J.G. Bekooij and G.J.M. Smit. Buffer Capacity Computation for Throughput Constrained Streaming Applications with Data-Dependent Inter-Task Communication. In Proc. *RTAS*. April 2008

Questions?