The tension between resource sharing and predictable performance in real-time SoCs

Kees Goossens

Philips Research
MEDEA+/MESA

PHILIPS

overview

- consumer-electronics applications
- example: advanced set-top box & digital TV SoC
- arbitration (resource management) is very important
- why is it hard
- a quality of service approach
- conclusions

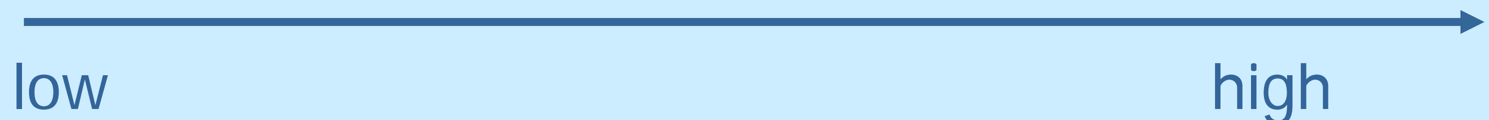
consumer-electronics applications

- convergence of application domains
  - increased functionality and heterogeneity
- higher semantic content/entropy

$\Rightarrow$ more dynamism

- embedded and pervasive applications ("ambient intelligence")
  - real time & safety critical

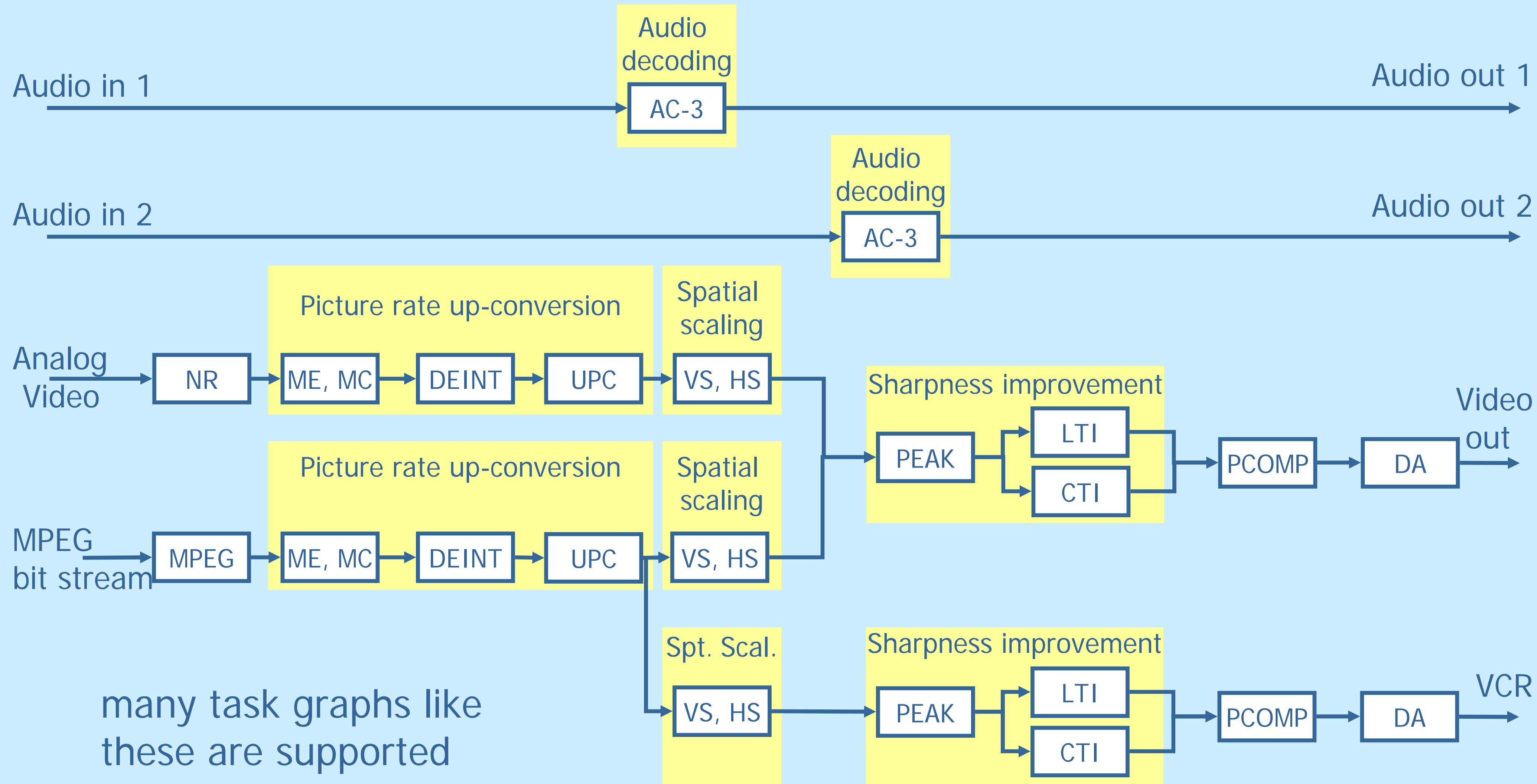
- users expect predictable behaviour
  - e.g. PC, mobile phone, TV, heating system, air bag



low high

- consumer electronics: all of this at low cost

example task graph



application characteristics

- advanced set-top box and digital TV (ASTB) application
  - two video and three audio streams
  - deep pipelines for video processing
  - large temporal processing memories
  - high data rates for high-definition video (e.g. 1920x1080@100i)
  - relatively latency-tolerant, except at in & out
  - but audio & video are synchronised (~55ms lip sync)

	computation	input	output	local	memory
audio	100 MOPS	640 kbps	5 Mbps	5 MBps	50 Kb
mpeg2	4 GOPS	10 Mbps	120 MBps	240 MBps	8 MB
video	100 GOPS	360 MBps	360 MBps	360 MBps	4 MB

application characteristics

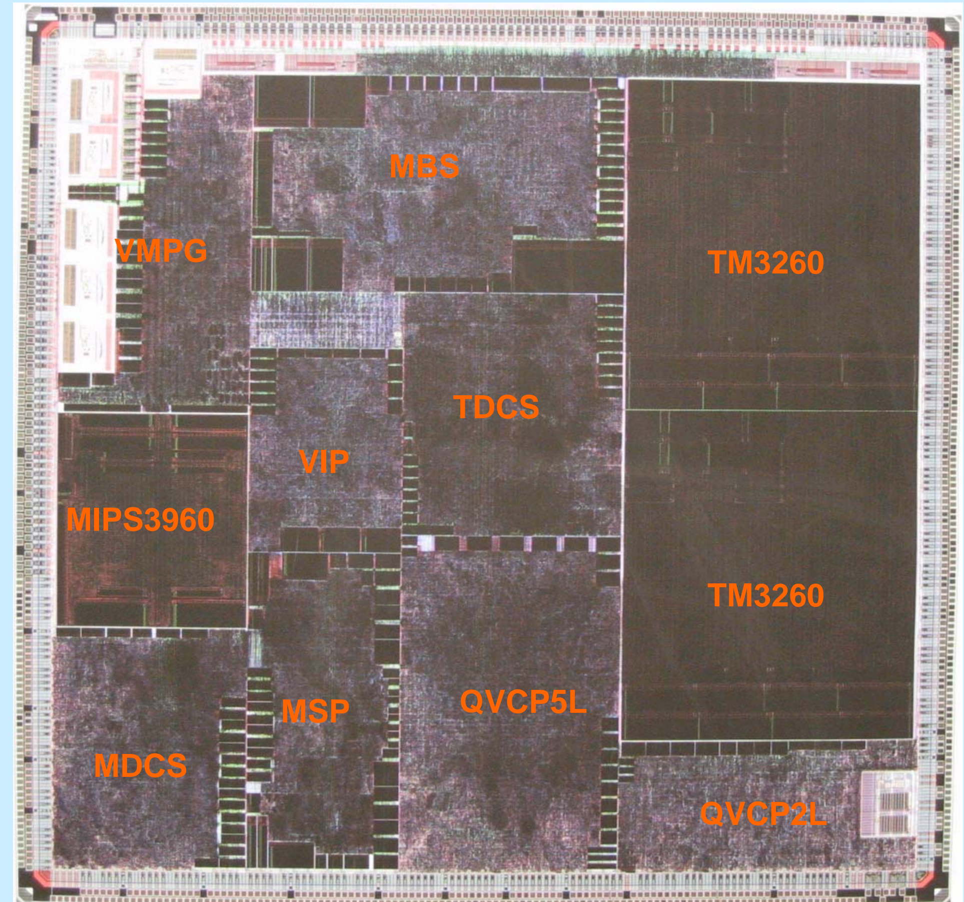
- low to high-performance computation
  - require heterogeneous computation elements
  - general purpose MIPS, application-domain-specific VLIW, function-specific (weakly) programmable cores
- for proprietary and emerging standards
  - require programmable processors
- product differentiation, run-time task graph changes
  - require programmable interconnect
- large temporal memories require off-chip memory
- real-time audio and video
  - require accurate / predictable arbitration

PHILIPS

example SoC

Philips's advanced set-top box and digital TV SoC (Viper2)

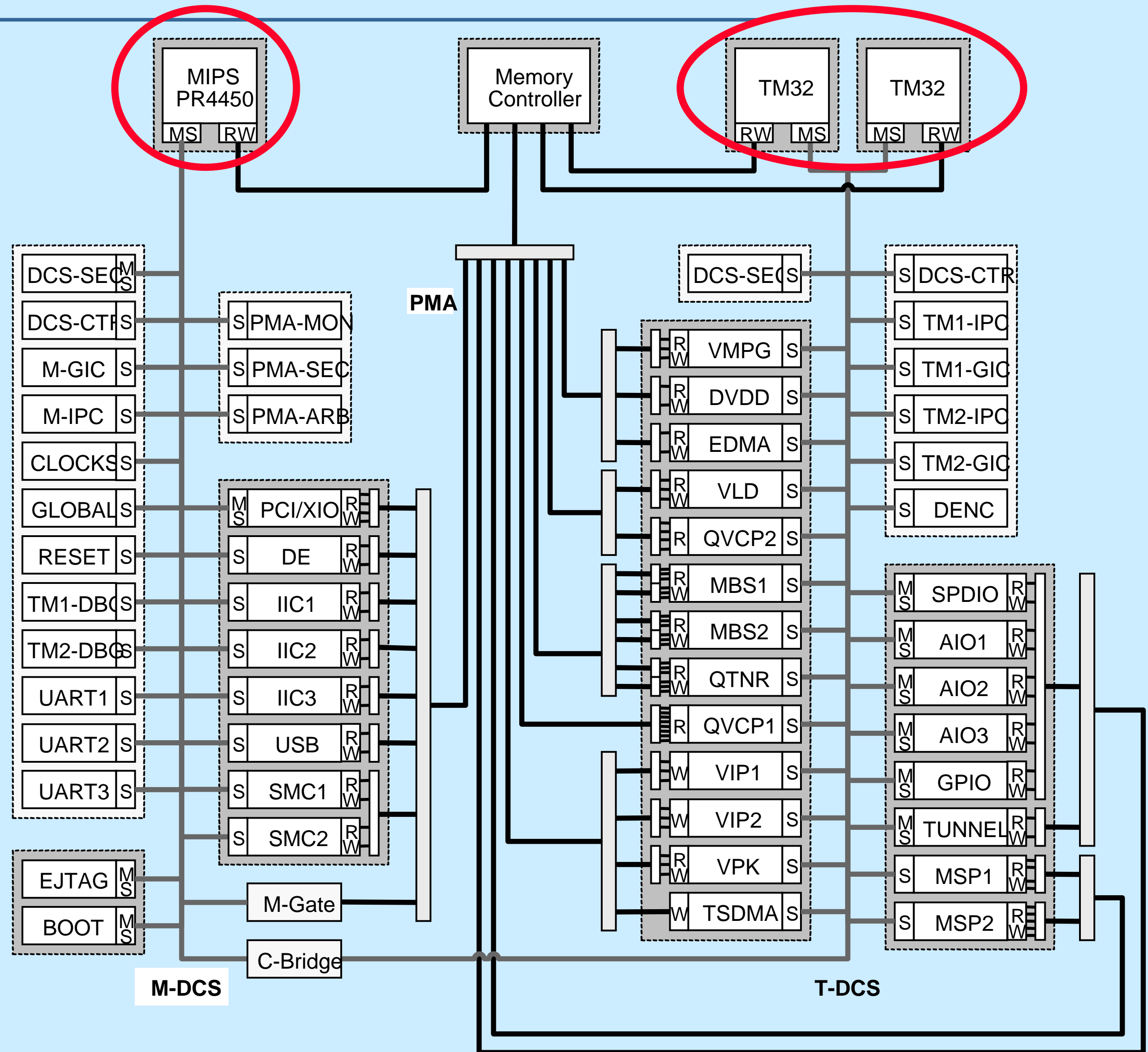
- 0.13 μm
- 50 M transistors
- 100 clock domains
- > 60 IP blocks



computation

- multiple IP
  - for performance
- heterogeneous
  - 1 CPU for control
  - 2 VLIW for emerging standards
  - 60 fn-specific weakly programmable cores
    - audio
    - video codec(s)
    - pixel processing
    - peripherals

  for efficiency & proprietary stds



performance guarantees

- different kinds of traffic are transported over the interconnect

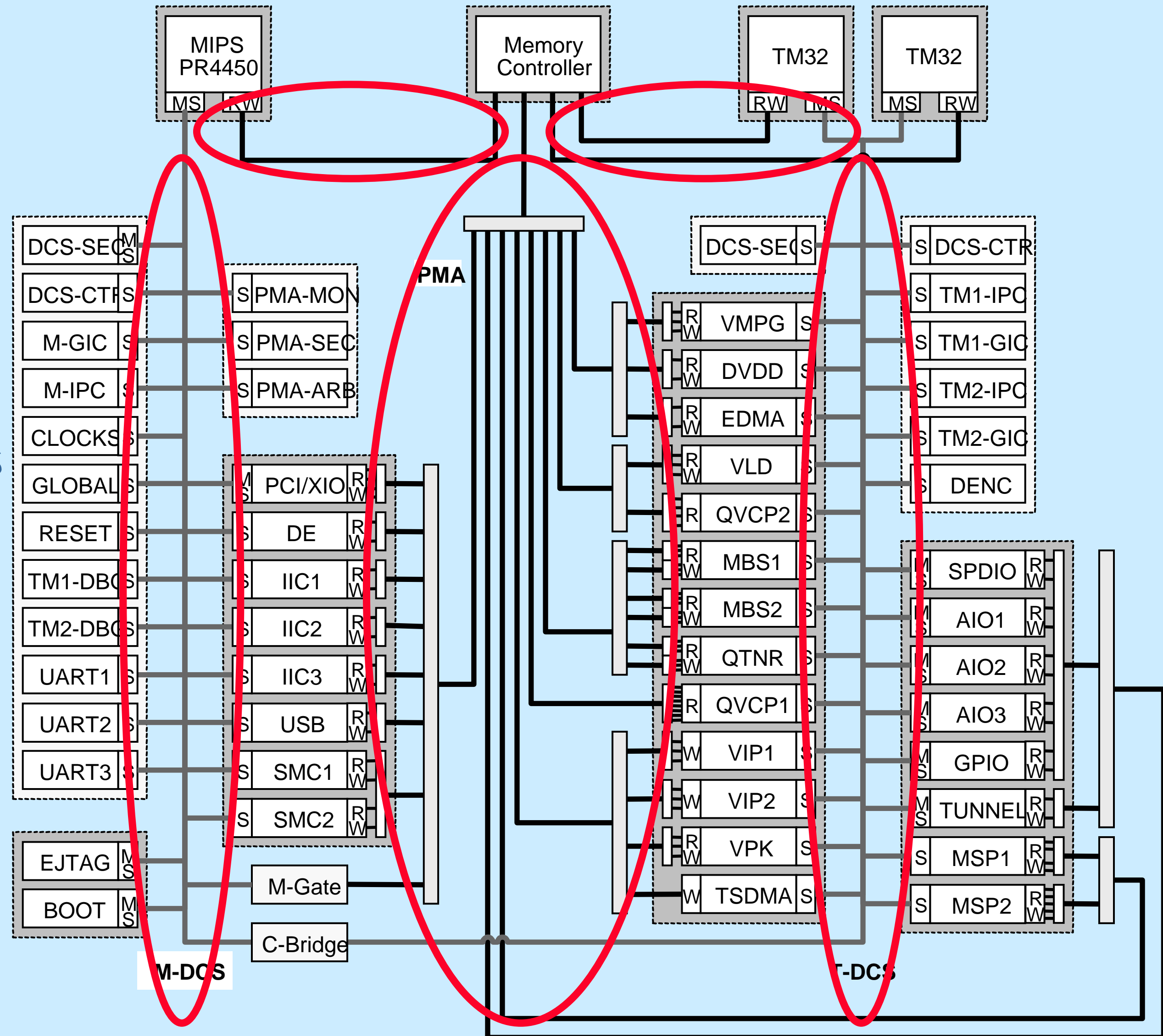
example	data rate	latency	jitter
control	low	low	low
cache misses	high	low	low
"hard real time" video	high	tolerant	low
"soft real time" video	high	tolerant	tolerant
audio/MPEG bitstreams	medium	tolerant	tolerant
graphics (best effort)	tolerant	tolerant	tolerant

- low latency is always hard to guarantee
- with larger data bursts the latency cost can be amortised (cf. $)
- low latency and low jitter requirements can be relaxed by buffering
  - e.g. in the absence of feedback loops, and using line blanking

communication

- multiple interconnects
- flexible/programmable

- heterogeneous
  - dedicated wires for low latency $ misses
  - PMA: for MPEG, audio, video, gfx (high throughput)
  - 2 DCS: control

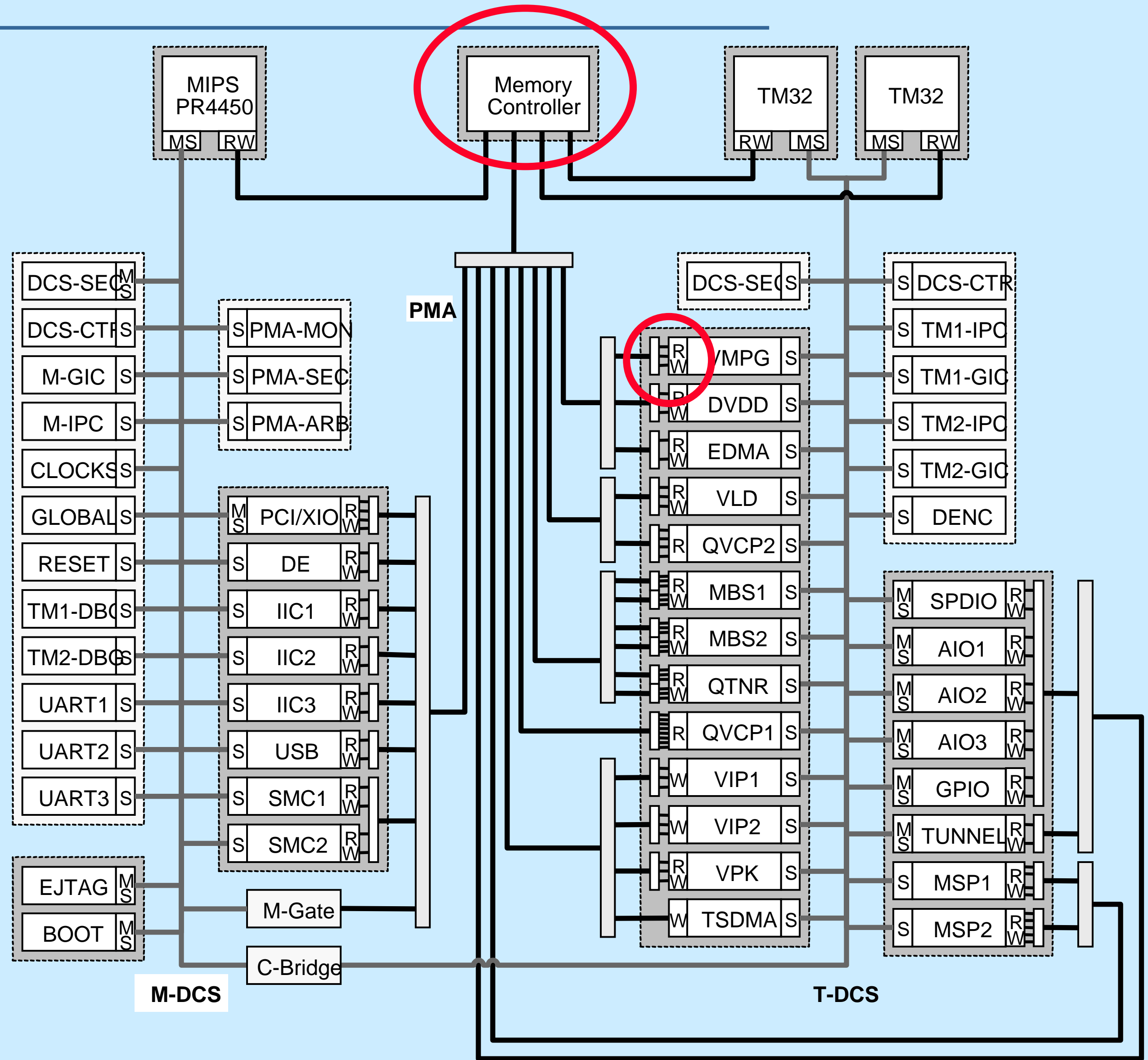


PHILIPS

storage

1 external memory

many small on-chip memories



arbitration

- embedded processors
  - RTOS, cache controller
- memory controller
  - optimises for bandwidth, then low latency
  - round robin with maximum burst length
- control network
  - optimises for low latency
  - round robin, no bursts
- pipelined memory-access interconnect
  - optimises for high data rate (bandwidth)
  - multi-level arbitration for multiple traffic types (low jitter, then jitter-tolerant, then best-effort)

external memory arbitration

- optimise use of bandwidth by efficiently scheduling the use of the interconnect, using different traffic types
- e.g. Viper2

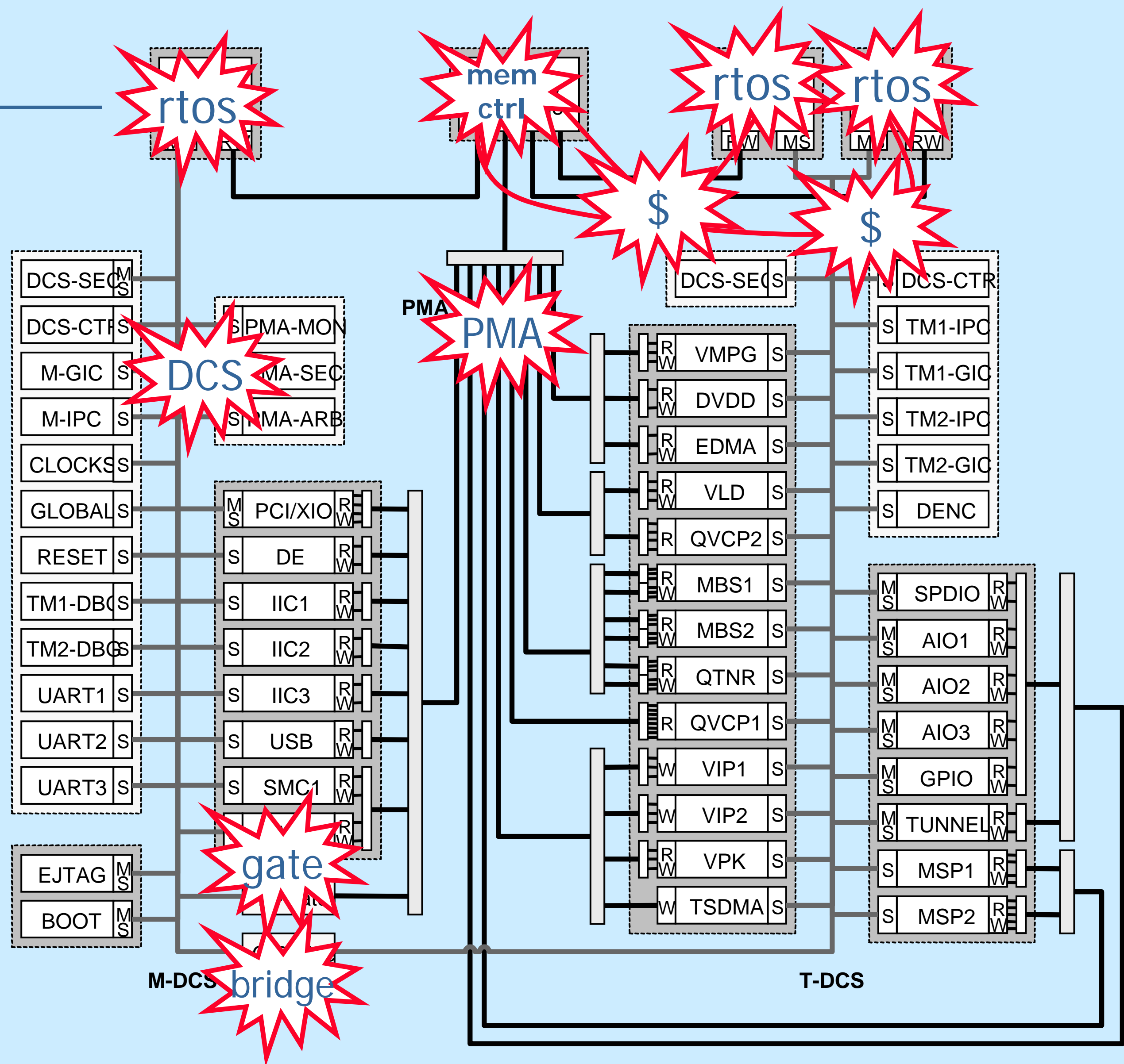
location	traffic	aim	method
DCS	LRLL	low latency	round robin
mem contrl	HRLL	low latency	round robin with cut off
PMA top	low-jitter	HRLT maximum latency	time-division multiplexing
PMA top	jitter-tolerant	HRLT minimum bandwidth	priorities
PMA top	best effort	best effort	round robin
adapters	(all)	coalescing	round robin

arbitration

local schedulers

- (RT)OS
  - task switching
  - interrupts
- cache strategy
  - cache pollution
- interconnect
  - busses, bridges
  - networks
- memory controllers
  - external memory

e.g. RR, TDMA, FCFS, LRU, EDLF, FIFO, priority, ...



what is the global behaviour, composed of interacting local solutions?

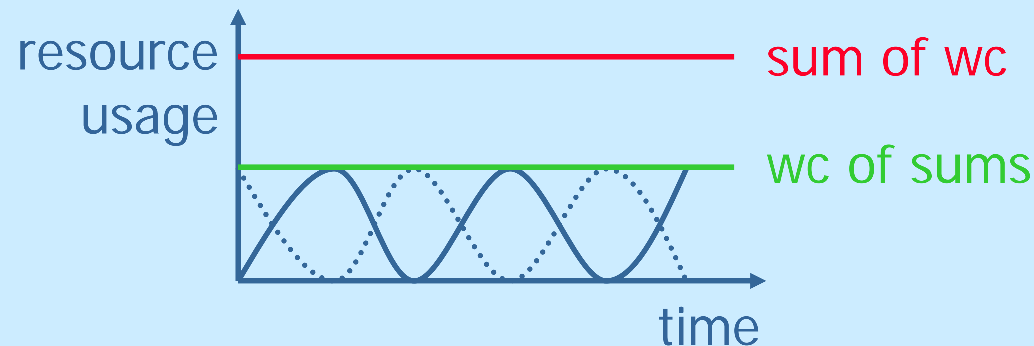
arbitration

- all schemes are optimised for one purpose (latency or bandwidth)
- each scheme uses global centralised arbitration
  [i.e. global in its own sphere, not global in the system as a whole]
  [strictly speaking the control interconnect in two places + bridge]
- the combination and interaction of various schemes is not trivial to understand

arbitration – why care?

- accurately predictable arbitration is required for
  - real-time performance
  - finding global optimum for low cost
    - many task graphs
    - different resources (computation, communication, storage)

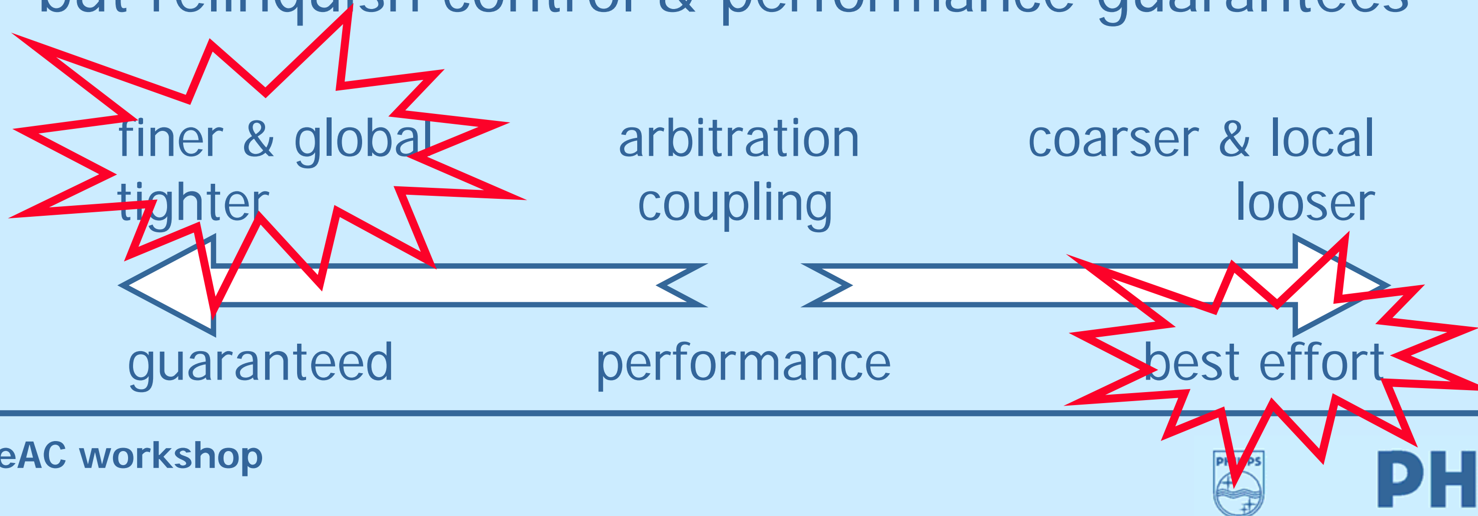
- sum of worst cases >> worst case of sums



the problem

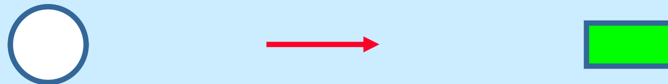
- how to take advantage of this?
  - with guaranteed performance
  - and without global arbitration

- arbitrate at finer grain and/or more globally
  - allows better (& guaranteed) optimisation
  - but requires that components are more tightly coupled
- or, make more use of statistical multiplexing
  - allows more loosely coupled components
  - but relinquish control & performance guarantees

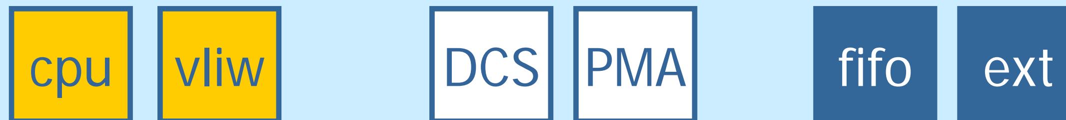


analysis

- three kinds of users (service users)
  - tasks, connections, buffers



- three kinds of resources (service providers)
  - computation, communication, storage

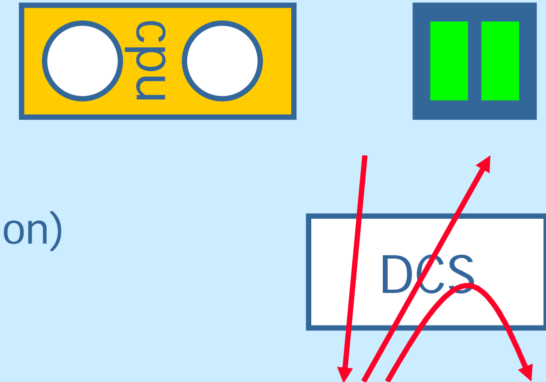


- a SoC may contain multiple, different resources of one kind
  - computation: 1 CPU, 2 VLIW, 60 ASIP & ASIC
  - communication: control (DCS), data (PMA)
  - storage: off chip, on-chip fifo & cache & …

analysis

1) users share a single resource

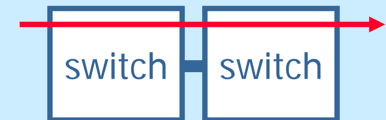
- e.g. tasks on one processor, buffers in one memory, connections using one switch (cf. contention)



- this requires arbitration

2) multiple resources of one kind are combined

- but the users want a single, end-to-end view
- e.g. control busses & bridges, multiple switches in a network (cf. congestion), local buffer & external memory



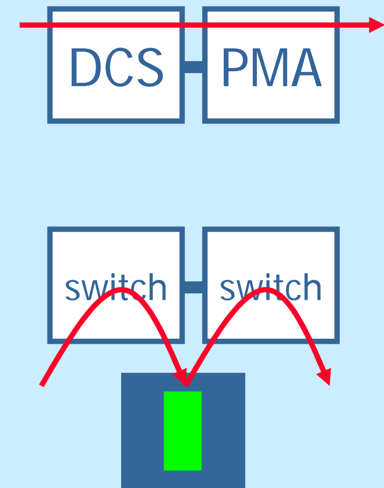
- this requires arbitration and reasoning about multiple arbiters (of one flavour, e.g. DCS)

PHILIPS

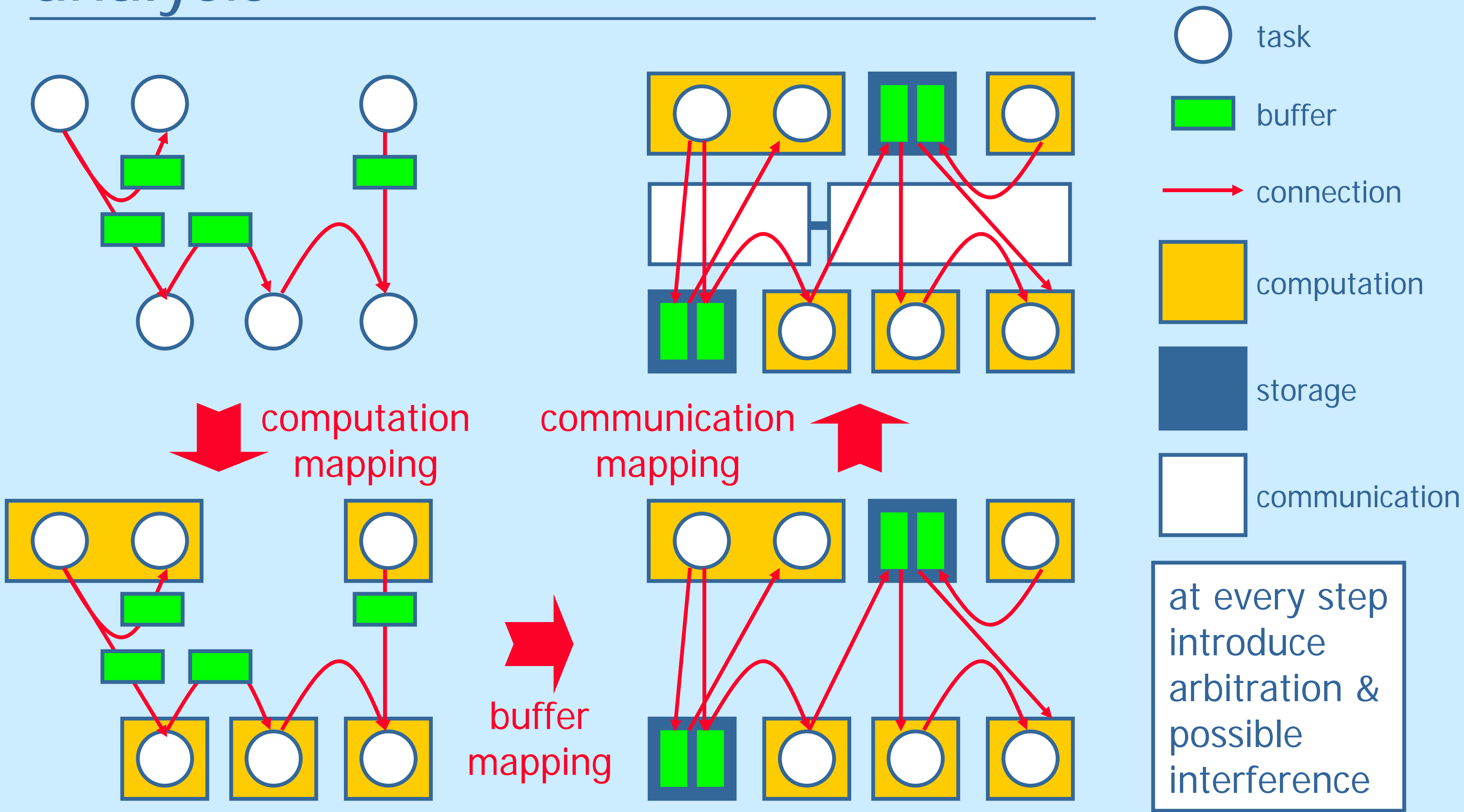
analysis

3) resources of all kinds are combined
- e.g. computation, communication, storage
  but also different communication & computation kinds
- but the users want a single, end-to-end view
  "what is the latency from video input to video output?"

- this requires reasoning about multiple arbiters
  - for multiple resource flavours
    - e.g. DCS & PMAN
  - of multiple resource kinds
    - e.g. storage & communication
- with different optimisation criteria
  (e.g. latency vs. jitter), which gives rise to interference



analysis



analysis - conclusion

- have multiple interacting arbiters with different goals
- but we must reason about the system as whole ("task-to-task" or "end-to-end" performance)
  - for real-time guarantees
  - for low cost
- but not about the whole system simultaneously
  - compositional reasoning is essential
- quality of service concepts enable this

quality of service

- quality of service is nothing more than
  1. stating what service you want (negotiation)
  2. having the provider either commit to or reject your request
  3. renegotiate when your requirements change

- create a series of steady states that are predictable
- QoS means reducing uncertainty to negotiation phase

- notion of commitment
  – guaranteed versus best-effort service

(guaranteed) services are good

- good design practice
  - services make assumptions on partners explicit
  - service contract limits possible interactions (simpler IP)
- composable method
  - reason about services/specification, not the implementation
  - services & design of different IP are independent
    - don't: build everything, test everything, change one thing, test everything, ...
    - but: build & test components independently
  - no interference (cf. caches)
- robust
  - (communication) failure of IP limited to negotiation
  - no overload of (communication) resources
    - local IP failure, not global system failure

example: on-chip communication

- notion of connection
- notion of service as a collection of properties
  - lossless, ordered, x Mb/s min data rate, y max latency, z max jitter, ...
- each property is guaranteed or not (best-effort)

example	data rate	latency	jitter
control	low	low	low
cache misses	high	low	low
"hard-real-time" video	high	tolerant	low
"soft-real-time" video	high	tolerant	tolerant
audio/MPEG bitstreams	medium	tolerant	tolerant
graphics (best effort)	tolerant	tolerant	tolerant

example: on-chip communication

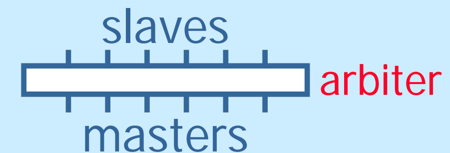
on-chip communication

- was single hop
  - bus, switch with one central, global arbiter
- is multi hop
  - busses & bridges, multiple interconnects (DCS & PMA), multiple switches / routers (networks on chip)
  - with local, distributed arbiters

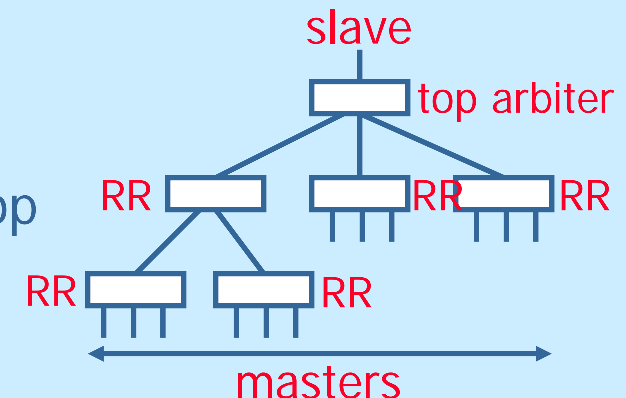
- require global arbitration for end-to-end services, e.g. latency
- this is harder in multi-hop interconnects

QoS – how to give guarantees

- single-hop interconnects (bus, switch)
  - global, central arbiter: easy, can be made sophisticated

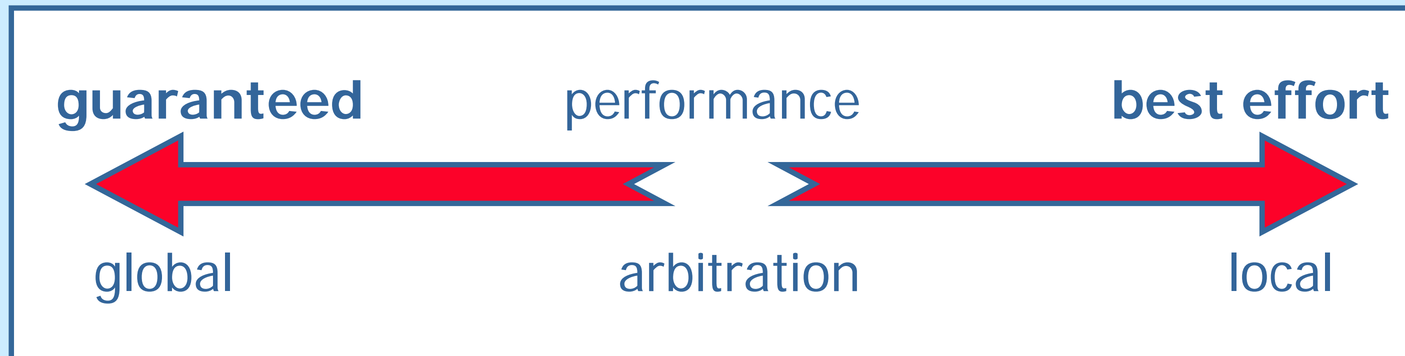


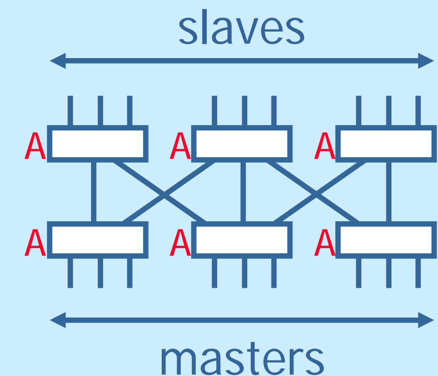
- multi-hop interconnects
  - single master or single slave (e.g. PMA)
    - hierarchical arbiter: relatively easy
      e.g. sequence of round robins,
      followed by sophisticated arbitration at the top



QoS – how to give guarantees

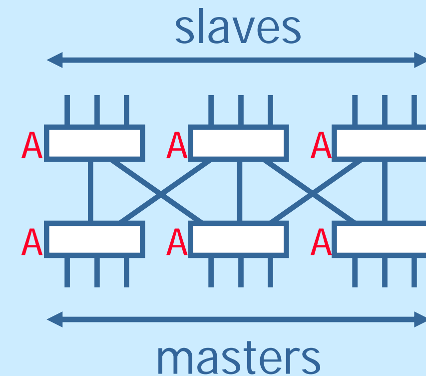
- multi-hop interconnects
  - multi-master & multi-slave
    - global & distributed arbiter: harder to implement (Æthereal, Nostrum)
    - local & distributed arbiter: hard to give good guarantees (most other packet-switched NOCs, e.g. SPIN)





QoS – how to give guarantees

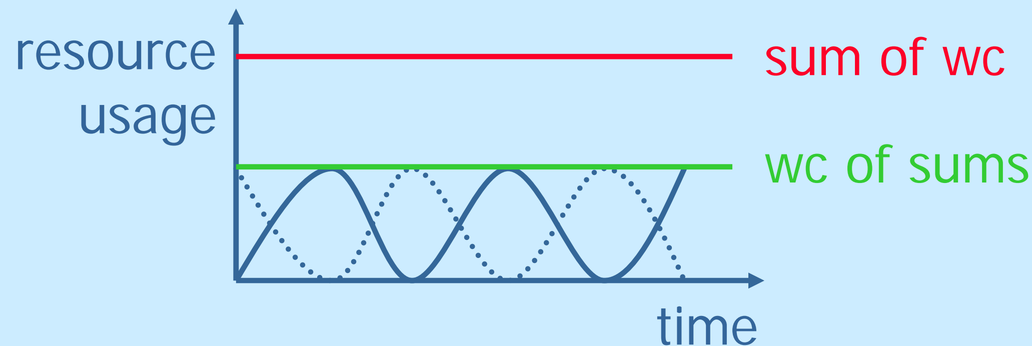
- distributed arbitration
  - local arbiters work independently
  - hence, it's hard to offer end-to-end guarantees
    - like latency & throughput



- solutions
  - tightly couple all local arbiters
    - e.g. contention-free routing: cheap (Æthereal)
  - give looser guarantees
    - e.g. rate-based scheduling: expensive
  - "overdimension and cross your fingers (best-effort)"
    - no guarantees, not necessarily cheaper

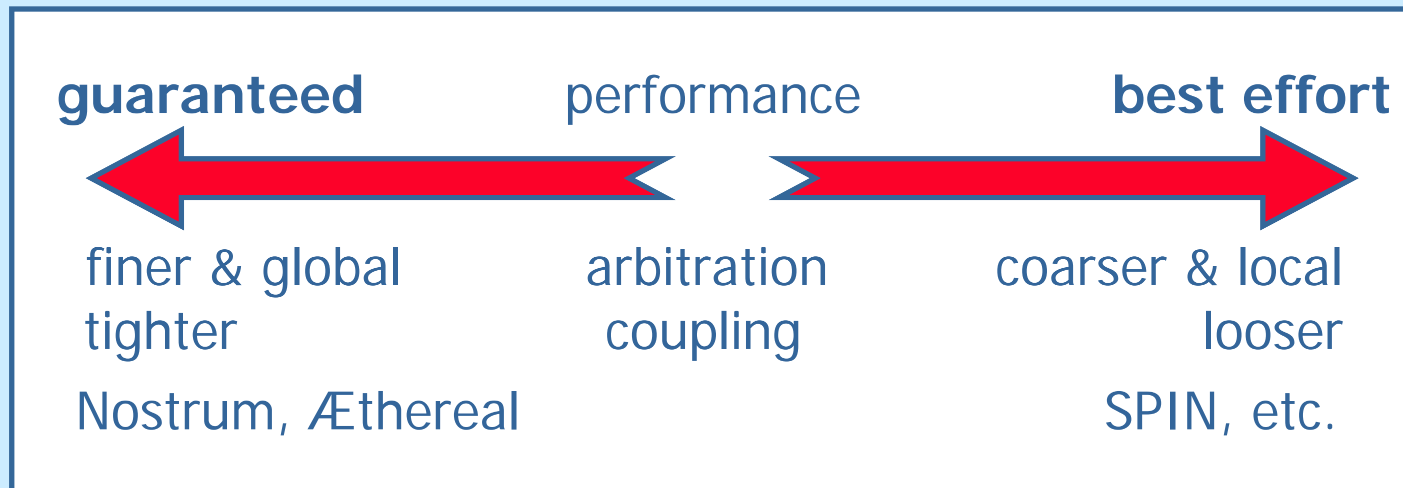
QoS – how to reduce cost

- finding global optimum for low cost
  - many task graphs
  - different resources (computation, communication, storage)
- sum of worst cases >> worst case of sums



QoS – how to reduce cost

- arbitrate at finer grain and/or more globally
  - allows better (& guaranteed) optimisation
  - but requires that components are more tightly coupled
- or, make more use of statistical multiplexing
  - allows more loosely coupled components
  - but relinquish control & performance guarantees



conclusions

- SoCs contain many heterogeneous resources that require multiple arbiters
- performance verification is very hard with multiple arbiters
- quality of service concept abstracts from arbiter implementations
- guaranteed services make SoC design composable & robust
- this approach is used in the Æthereal network on chip
- should be used for all resources, incl. computation & storage