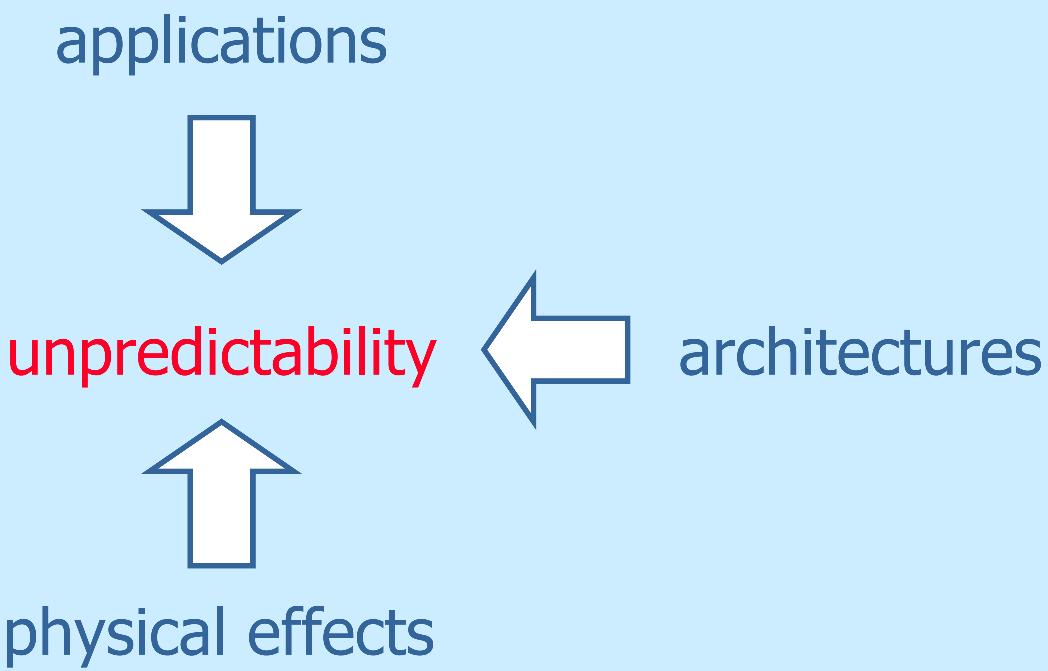
Systems on Chip and Networks on Chip: Bridging the Gap with QoS

Kees Goossens
Philips Research
The Netherlands

PHILIPS

sources of unpredictability



- but we still want to build predictable systems!
- the quality of service concept helps

overview

1. application & user views
   - leads to quality of service (QoS) concept

2. system on chip (SoC) design view
   - leads to networks on chip (NoC)

3. NoCs and QoS: a synthesis to recuperate predictability
   - types of QoS commitment and their costs

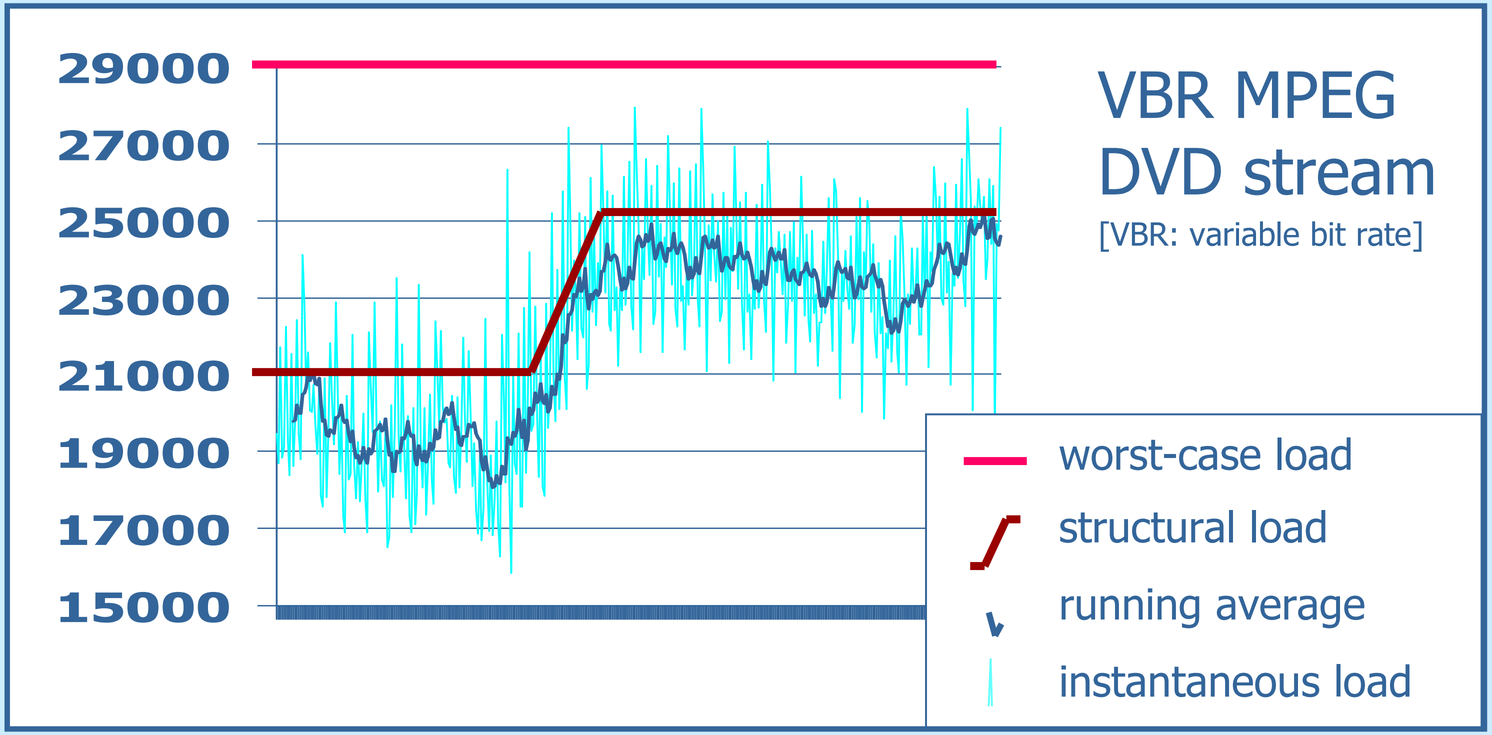
4. the Æthereal approach and architecture

1. application & user views

application-induced unpredictability

1. future applications

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



1. future applications

- 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

- quality of service is resource management for predictability

PHILIPS

1. consumer-electronics requirements

- consumer-electronics media processing is challenging

signal processing	media processing	multi-media
hard real time very regular load	hard real time irregular load	soft real time irregular load
high quality	high quality	"sloppy" quality
worst case typically on DSPs	average case SoC/media processors	average case PC/desktop

2. SoCs and NoCs

architecture-induced unpredictability

2. systems on chip

- Moore’s law predicts exponential growth of resources
- but.. someone has to do the work to make it come true

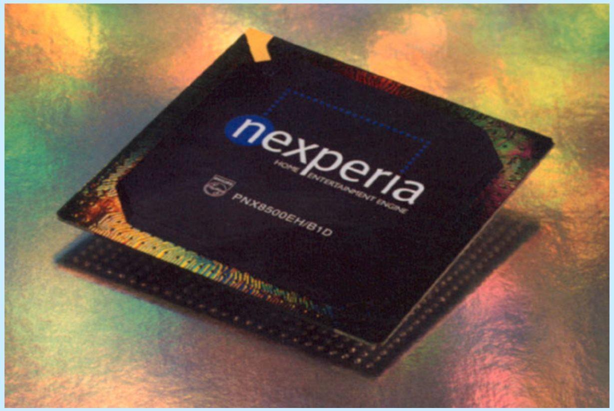
1. deep submicron problems (DSM)
   - wire vs. transistor speed, power, signal integrity
2. design productivity gap
   - IP re-use, platforms, NoCs
   - verification

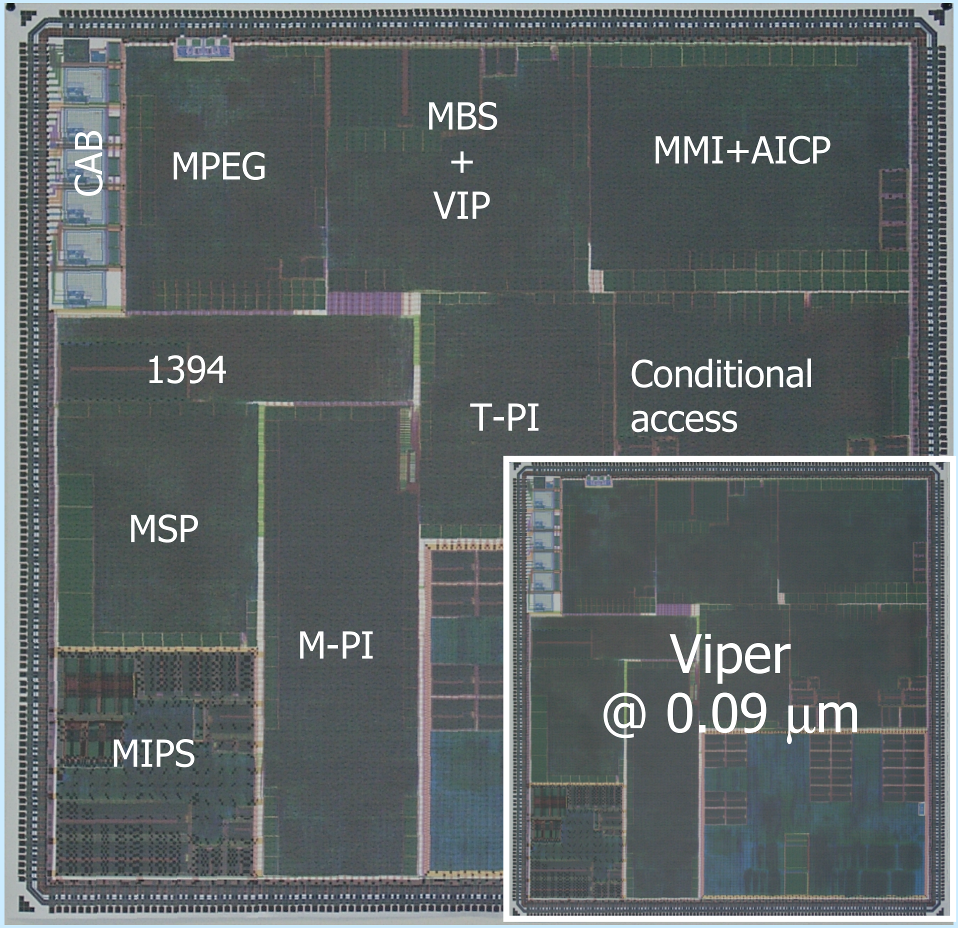
2. example SoC

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

Viper (pnx8500)

- 0.18 µm / 8M
- 1.8V / 4.5 W
- 35 M transistors
- 82 clock domains
- more than 50 IP blocks





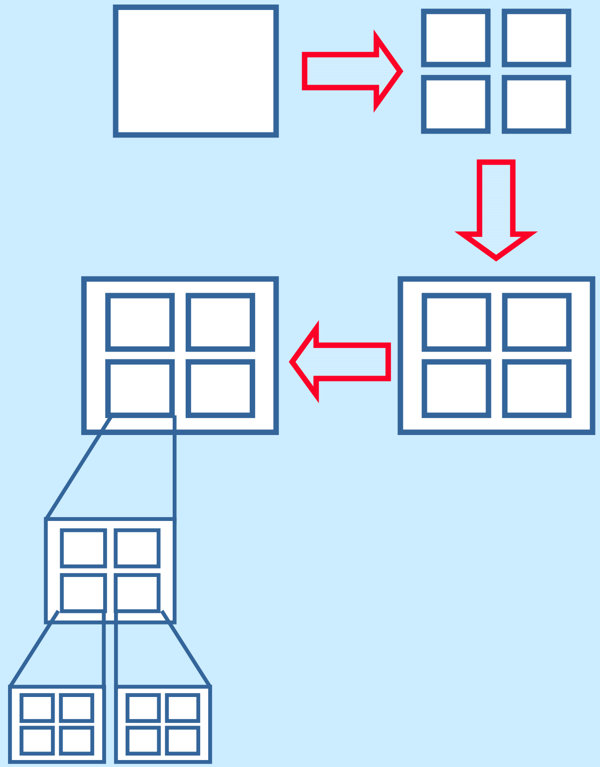
2. composing local solutions

to solve both DSM and productivity gap issues:

- global approaches won't work with exponential problem

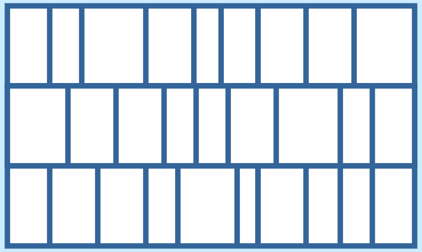
- so
  1. break up problem (modularity)
  2. then compose sub-solutions
  3. in a scalable fashion

  – hierarchy helps
  – abstraction helps

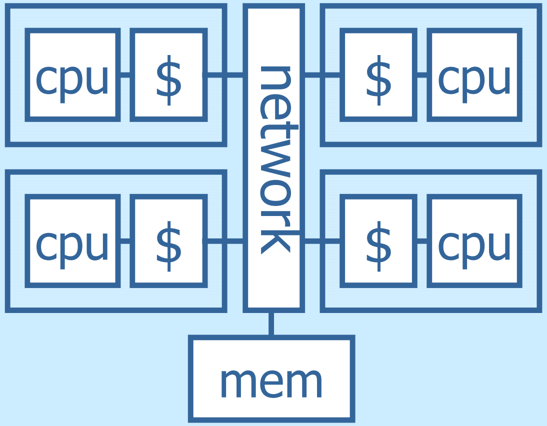


2. composing local solutions - examples

- for timing (closure):
  - globally asynchronous, locally synchronous (GALS)



- for lay-out:
  - IP-level Mead & Conway, e.g. wiring strategies, tiling
- for architectures:
  - chip multiprocessing (CMP), tiling, systolic/cell, …



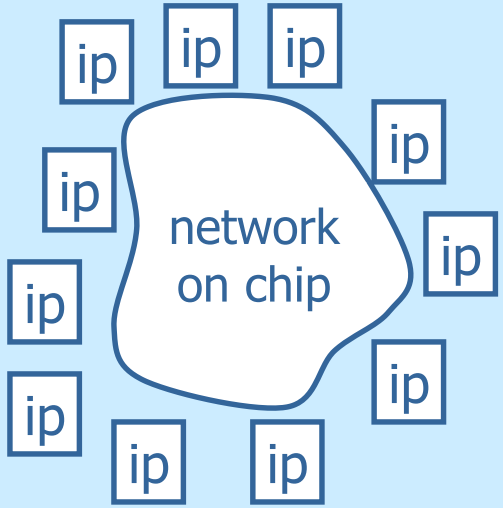
- for programming:
  - e.g. Kahn process networks
    - locally sequential, globally concurrent
    - locally shared memory, globally message passing

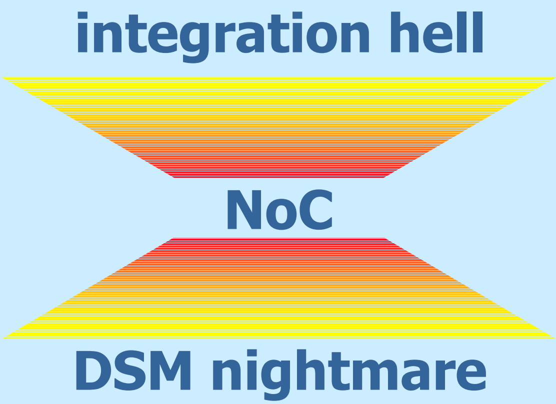
2. networks on chip

- have to connect many local solutions
  - heterogeneity, scalability

- through the decoupling of communication & computation

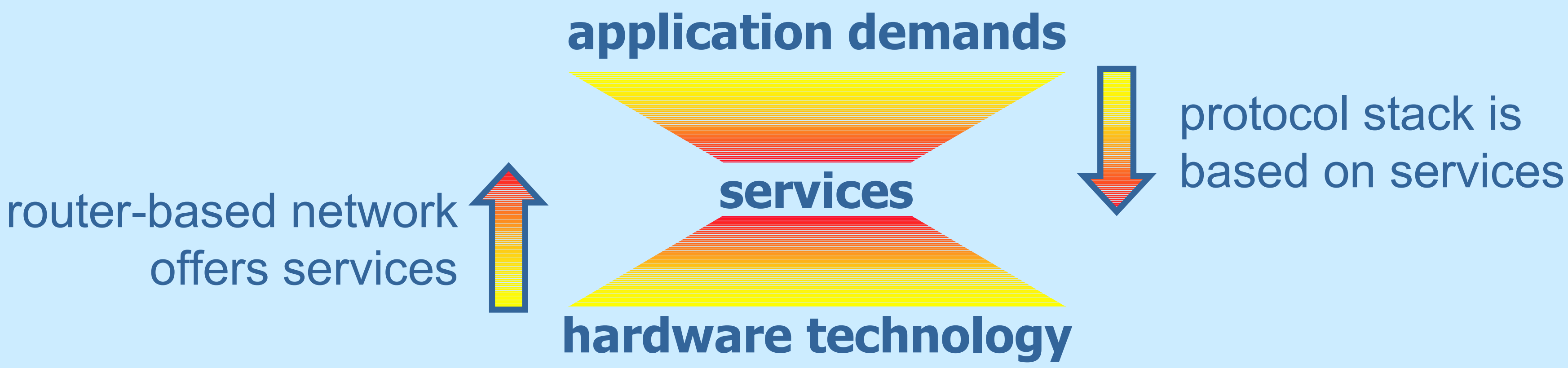
- networks on chip address this challenge
  - from above (protocol stack, IP re-use)
  - from below (DSM)





2. networks on chip

- two-pronged approach
  - deal with communication dynamism
  - protocol stacks enable differentiated services



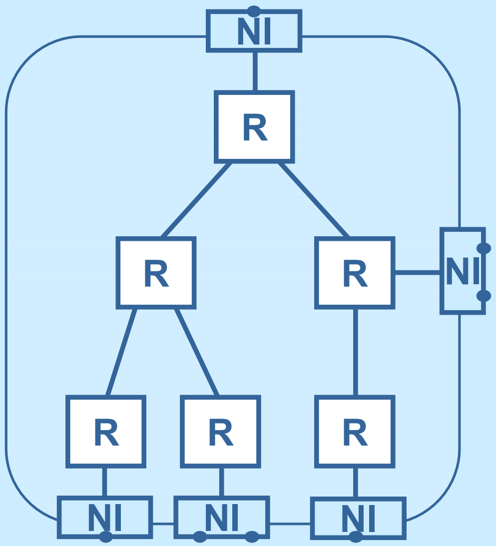
  - scalable, compositional IP composition
  - structure interconnect (wires, lay-out, timing)

2. networks on chip: examples

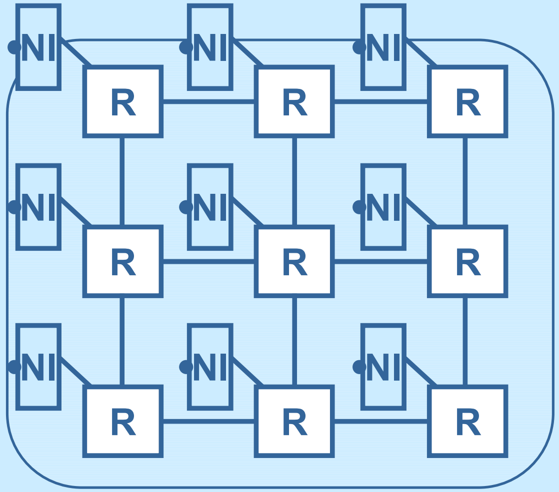
two types of component:

- routers
  - transport data in packets
- network interfaces
  - convert IP view (transactions, e.g. Amba, OCP) to network view (packets)

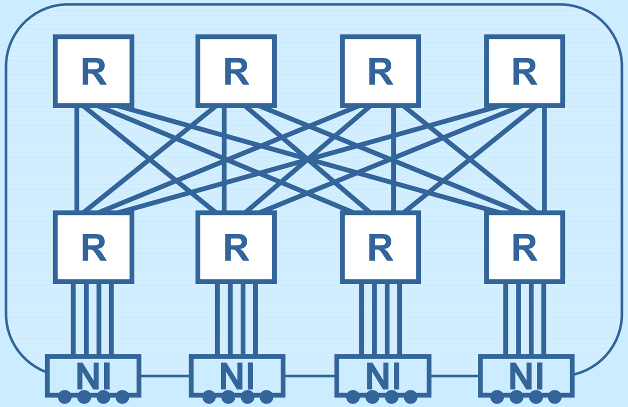
tree



mesh



fat tree



2. networks on chip: advantages

- differentiated services
  - offer different kinds of communication with one network
- scalable
  - add routers and network interfaces for extra bandwidth (at the cost of additional latency)
- compositional
  - add routers/NIs without changing existing components e.g. timing, buffers
- efficient use of wires
  - statistical multiplexing/sharing (average vs. worst-case) ℘ fewer wires ℘ less wire congestion
  - point to point wires at high speed
- communication becomes re-usable, configurable IP

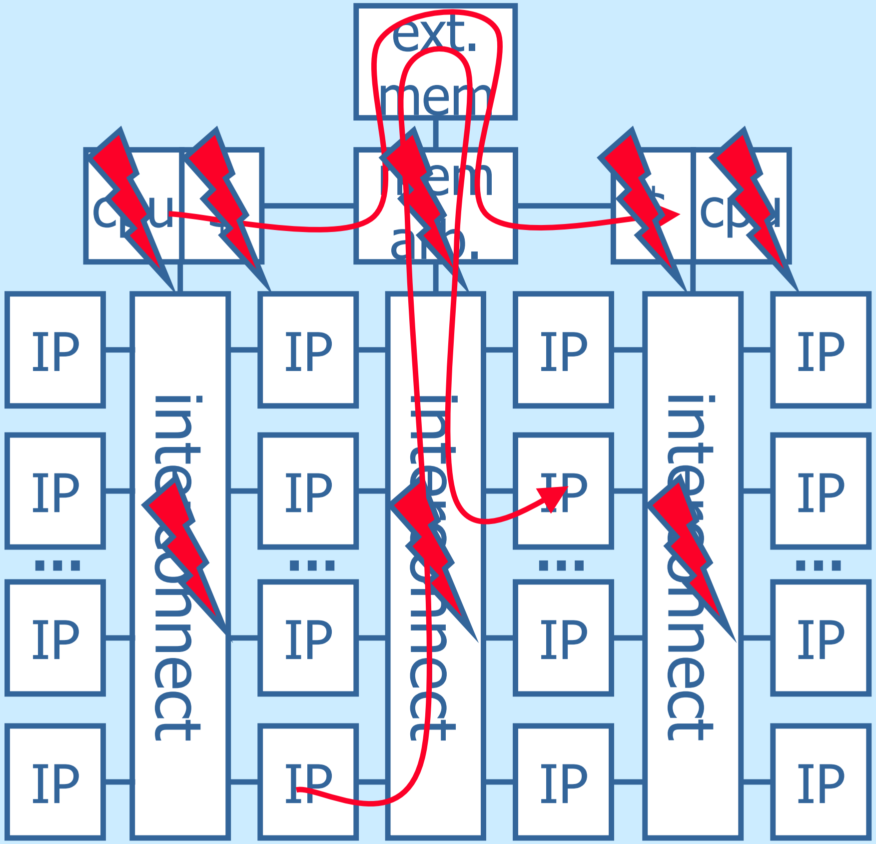
3. NoCs and QoS: a synthesis

managing unpredictability
with quality of service

3. GULP?

local schedulers

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



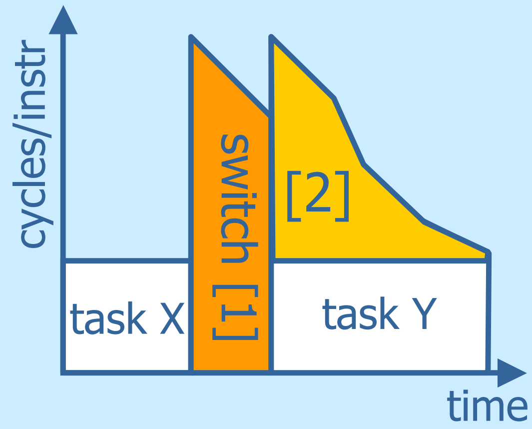
e.g. RR, TDMA, FCFS,
LRU, EDLF, FIFO,
priority, …

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

PHILIPS

3. GULP?

- example
  - CPU @ 225MHz, 64KB I$
  - ~70 cycle latency to external memory
- single task switch ~13K cycles =
  - RTOS overhead + task switch [1] ~6K cycles +
  - cache reload due to pollution (10%) [2] ~7K cycles
- with 20 hard real-time video tasks @ 60Hz
  - 1200 switches X 13K cycles ℘ 7% CPU load
  - what about effective task throughput, latency?



- have to guarantee
  - throughput and latency (for hard real-time IP)
  - throughput (for soft real-time IP)
  - minimal latency (for CPU control tasks)

PHILIPS

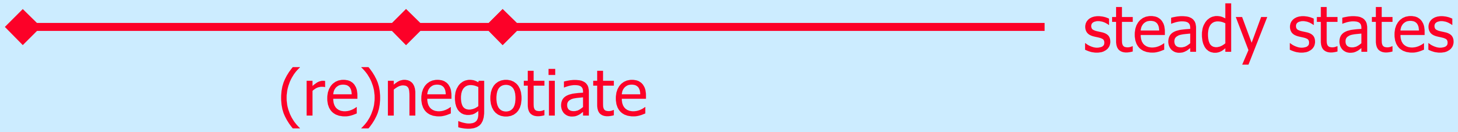
3. GULP?

- so, now we can make SoCs with NoCs,
  using our decoupled recomposed solutions
- get locally predictable,
  globally unpredictable behaviour (GULP)
  - GALS: multiple clock domains leads to uncertainty in time or data
  - power management: combining local autonomous probabilistic managers
  - NUMA: local vs. remote shared memory, dynamic (cache/mem) paging
  - Kahn process networks: how are sequential processes scheduled?
  - interacting schedulers/resource managers
- but the user wanted (global) predictable behaviour..

3. QoS & GULP

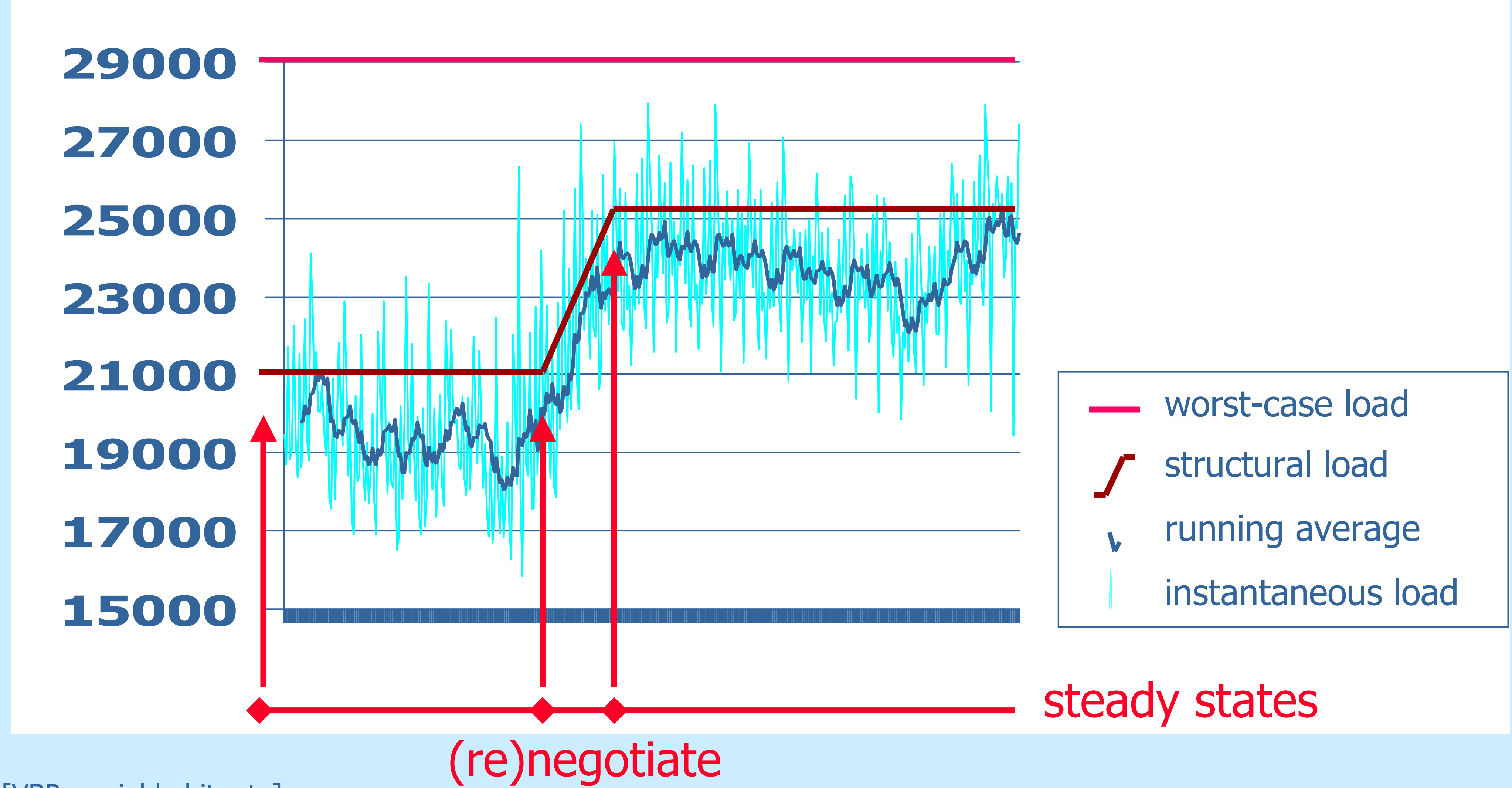
- our tenet is that a quality of service approach is essential to recuperate global predictability
  - the user and application require it
  - it fits well with NoC protocol stack

- 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

3. example: QoS & VBR



[VBR: variable bit rate]

3. quality of service

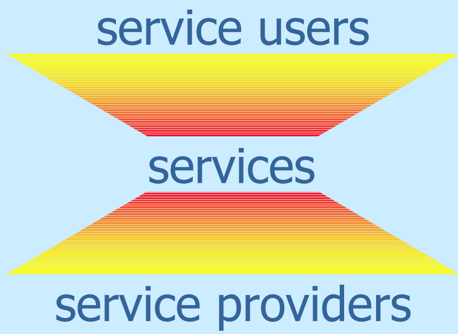
- QoS means reducing uncertainty to negotiation phase
  - for both user and provider
  - requires & enables resource management

- notion of commitment
  - guaranteed versus best-effort service

- types of commitment
  1. correctness e.g. uncorrupted data
  2. completion e.g. no packet loss
  3. bounds e.g. maximum latency

3. some remarks

- the types of commitment are dependent
  - e.g. cannot offer latency bound without completion
  - this has repercussions for protocol stack & architecture
    - data retransmission on unreliable low-swing wires immediately excludes guaranteed latency & jitter



- quality of service must be done at all levels
  - physical: power manager of IP blocks
  - link & network: network and communication links
  - task level: CPU scheduler (RTOS), application software

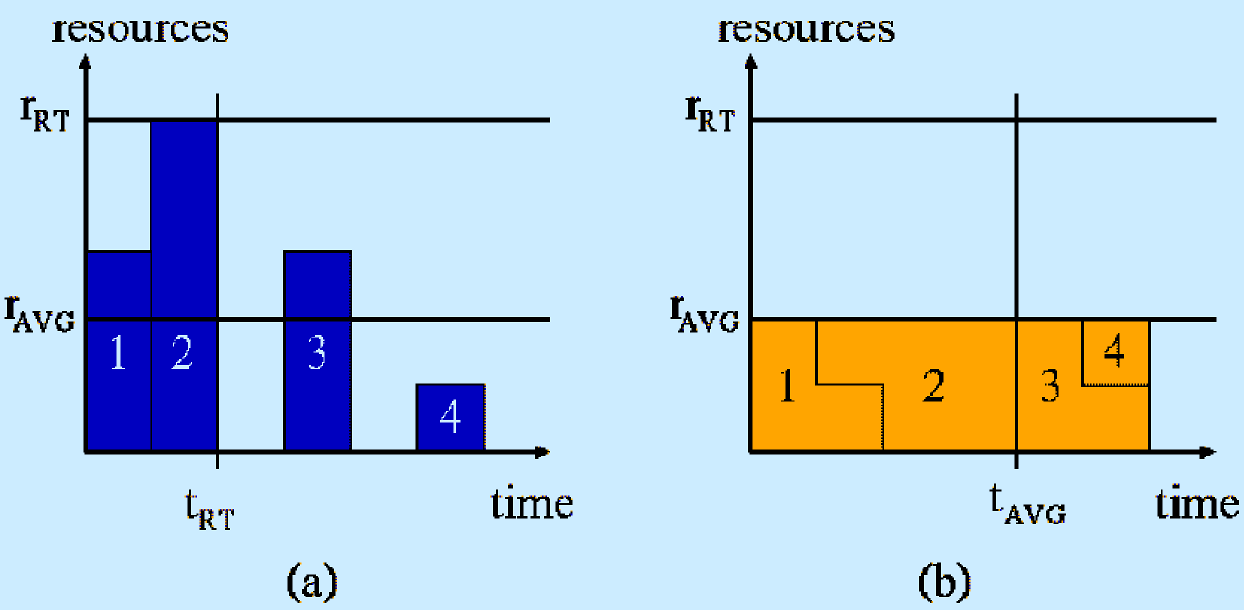
- QoS is pervasive, it cannot be bolted on afterwards

3. some remarks

- the “statistical guarantees” oxymoron
  - e.g. guaranteeing >0% packet arrival implies QoS
    - have to keep track of percentage lost
  - post hoc analysis of behaviour of architecture is no guarantee
    - unless boundary conditions of analysis are enforced (and then resources have to be managed ℘ QoS)

3. the cost of QoS

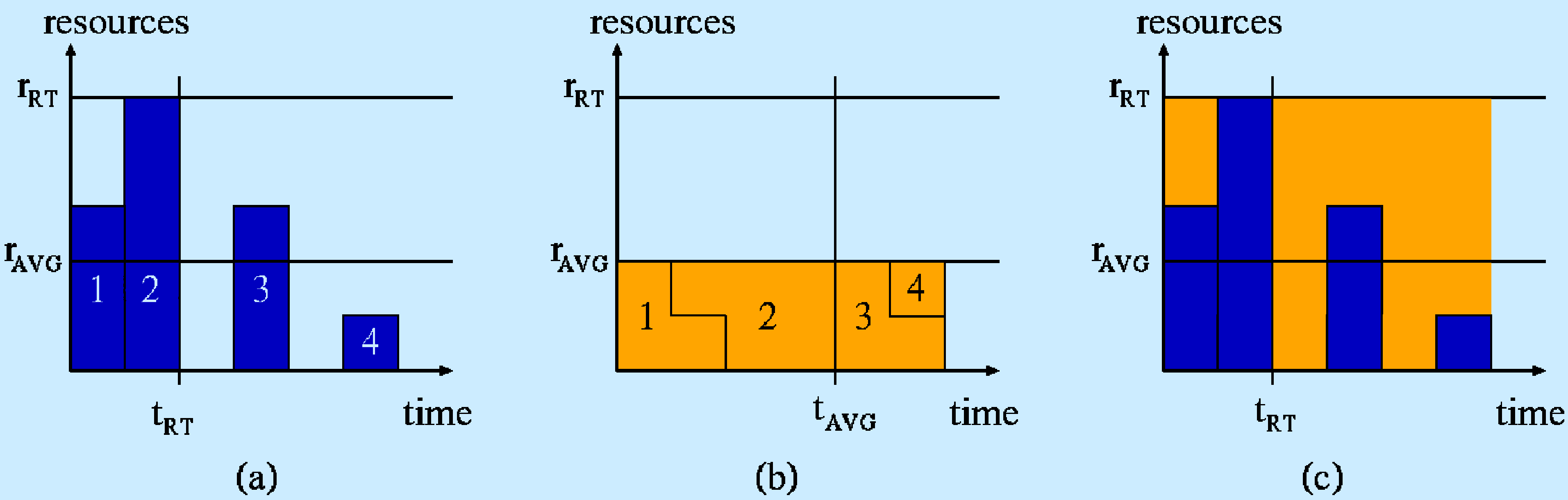
a) guaranteed bounds require worst-case resource dimensioning
b) completion requires at least average-case resources



PHILIPS

3. the cost of QoS

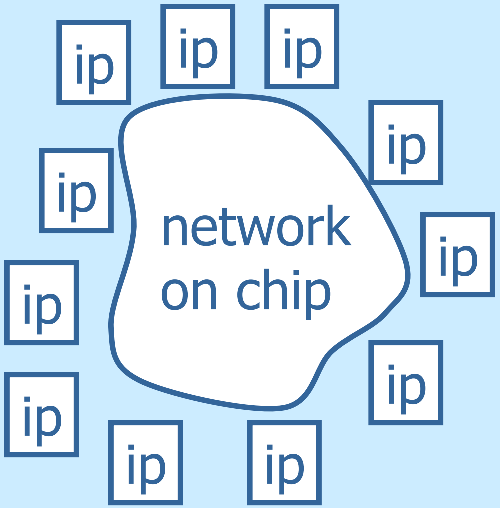
- best-effort services can have better average resource utilisation at the cost of unpredictable/unbounded worst-case behaviour
- the combination of best-effort & guaranteed services (c) is useful!



3. quality of service

- IP integration is the problem
- SoC design becomes communication centric
- the NoC is the focus of the architecture

- to make SoCs predictable, NoCs must offer QoS



predictable
composition
with QoS

4. NoCs and QoS: the Æthereal approach

4. Æthereal context

- consumer electronics
  - reliability & predictability are essential
  - low cost is crucial
  - time to market must be reduced

- NoCs and QoS helps on all accounts

- NoCs are focal point of SoCs
  - ℘ QoS is essential for NoCs

- hence the Æthereal NoC offers differentiated services
  - to manage (and hence reduce) resources
  - to ease integration (and hence decrease TTM)

4. NoC services

- request communication services using connections
  - opening & closing affect resource reservations

- with properties
  - data integrity (uncorrupted data transfer)
  - transaction ordering
    - un/ordered per slave/connection
  - transaction completion
  - flow control
    - data loss or not
  - delivery bounds
    - throughput, latency, jitter

commitment

correctness

completion

bounds

4. architecture decisions

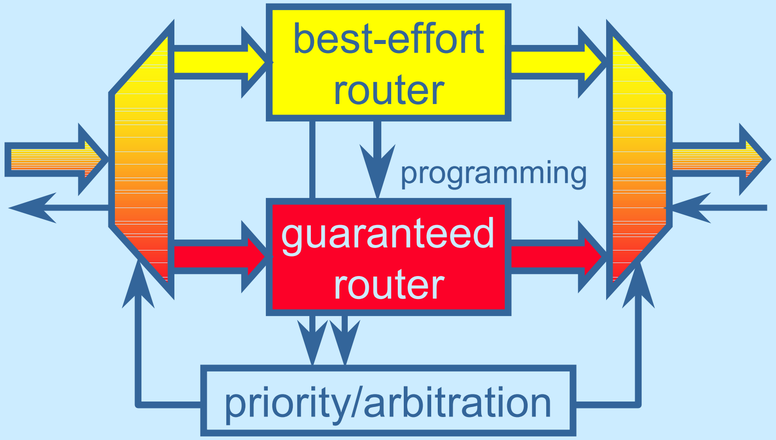
- we expect lossless, (partially) ordered connections with or without throughput guarantees to be most popular
- hence, to reduce costs, we implement this natively, i.e.
  - don't drop data in the network(*)
  - don't reorder data in the network

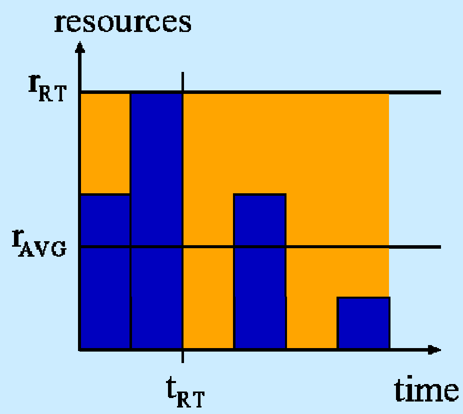
  ℘ no retransmissions & filtering of duplicates
  ℘ no reorder buffers
- not dropping data complicates congestion & deadlock issues

(*) excluding network interfaces

4. architecture decisions

- conceptually, two disjoint networks
  - a network with throughput+latency guarantees (GT)
  - a network without those guarantees (best-effort, BE)





- we have a several types of commitment in the network
  - combine guaranteed worst-case behaviour with good average resource usage

4. best-effort router architecture

- worm-hole routing
- input queueing
- source routing
  - source decides on path to follow through network

- other options (all feasible, area-wise)
  - input queuing with virtual-cut-through routing
  - virtual output queuing & iSLIP with worm-hole routing
  - output queuing with worm-hole routing

4. guaranteed-throughput router

- contention-free routing
  - synchronous, using slot tables
  - time-division multiplexed circuits
- store-and-forward routing
- headerless packets
  - information is present in slot table

4. architecture decisions

- to offer guaranteed latency or bandwidth over finite interval
  - cannot drop data
  - must bound contention and congestion

- rate-based scheduling
  - has high buffer costs (deep fifos)
- deadline-based scheduling
  - even higher buffer costs (deep priority queues)
- contention-free routing
  - low buffer costs (shallow fifos)

- NB, all require some notion of time

4. contention-free routing

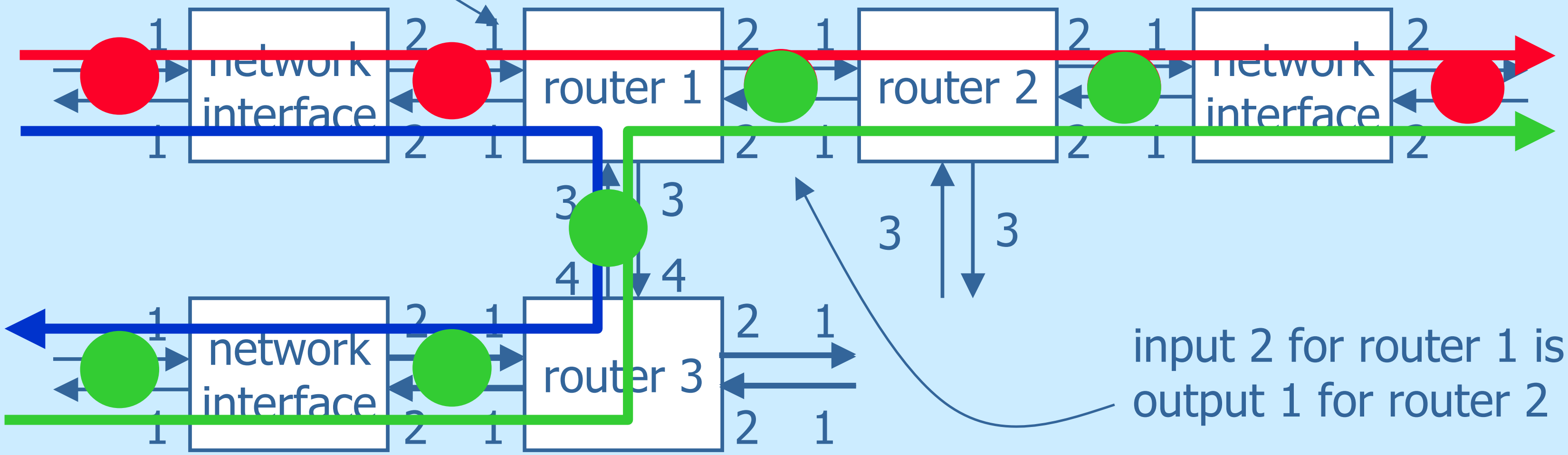
- latency guarantees are easy in circuit switching
- emulate circuits with packet switching
- schedule packet injection in network
  such that they never contend for same link at same time
  - in space: disjoint paths
  - in time: time-division multiplexing
  - or a combination

slot table
of router 1

o1	o2	o3	o4
-	i1	-	-
-	i3	-	i1
-	-	-	-
-	-	-	-

o1	o2	o3	o4
-	-	-	-
-	i1	-	-
-	i1	-	-
-	-	-	-

the input routed to the
output at this slot



input 2 for router 1 is
output 1 for router 2

o1	o2	o3	o4
-	-	-	i1
-	-	-	-
i4	-	-	-
-	-	-	-

use slots to

- avoid contention
- divide up bandwidth

4. programming model

- use best-effort packets to set up connections
  - set-up & tear-down packets like in ATM (asynchronous transfer mode)

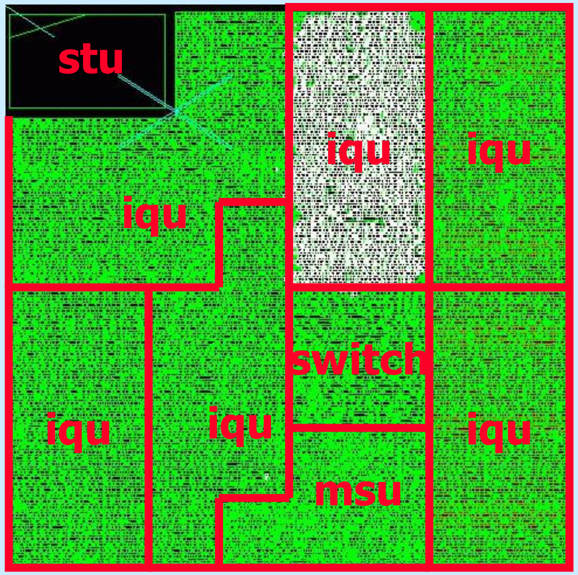
- distributed, concurrent, pipelined
- safe: always consistent
- compute slot assignment compile time, run time, or combination

- connection opening is guaranteed to complete (but without a latency guarantee) with commitment or rejection

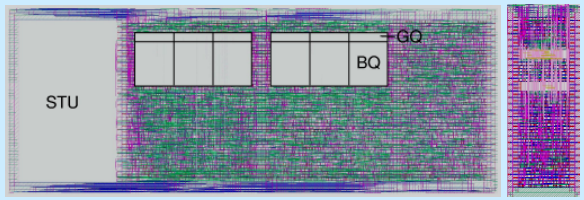
4. architecture insights

- memories (for packet storage)
  - register-based fifos are expensive
  - RAM-based fifos are as expensive
    - 80% of router is memory
  - special hardware fifos are very useful
    - 20% of router is memory

- speed of memories
  - registers are fast enough
  - RAMs may be too slow
  - hardware fifos are fast enough



routers based on
register-file and hardware fifos
drawn to approximately
same scale (1mm2, 0.26mm2)



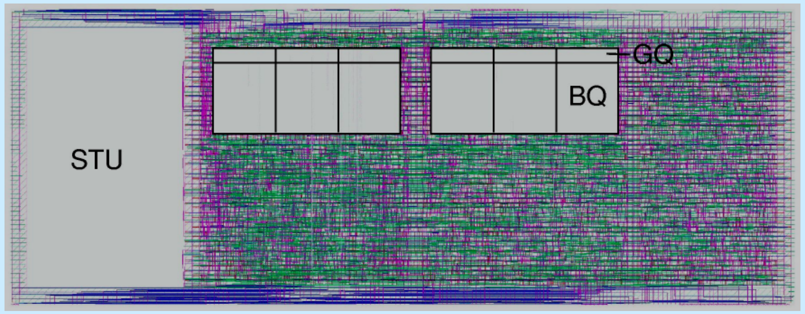
4. architecture insights

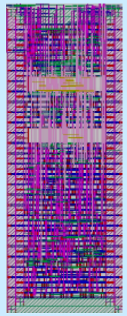
- router must be scalable, but up to a point
  - in terms of area (switch, input or output buffers)
  - in terms of speed (arbiter, memories)
  - don't go beyond 8x8 routers (for fat tree)?
- switch is less of a problem than expected
- latency of router is essential
  - increase data rate
  - increase arbitration rate
- minimise number of hops in network
  - topology choice
- think about trade-off hops versus router latency

4. router results

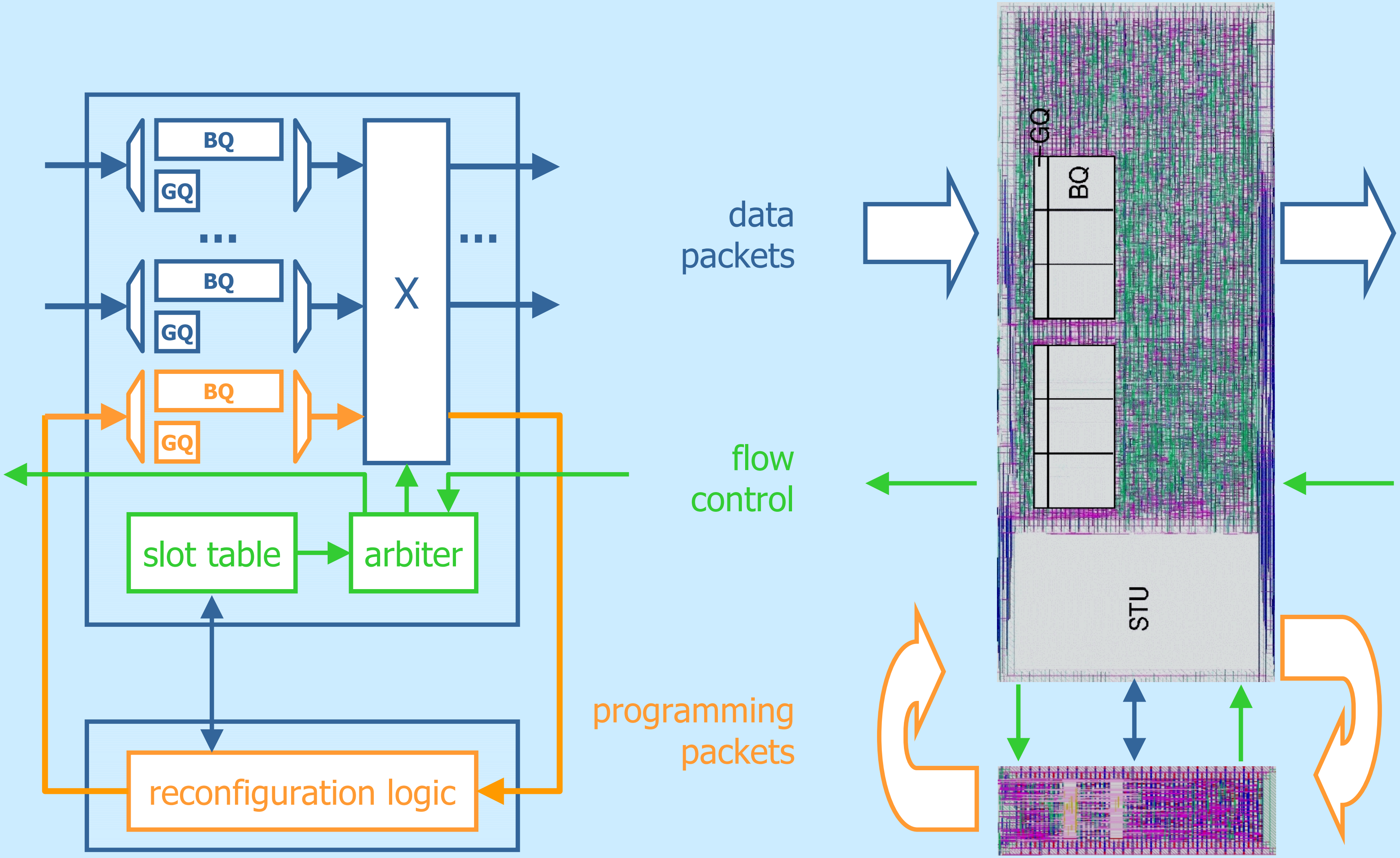
a prototype router:

- 5 input and 5 output ports (arity 5)
- 0.25 mm2 CMOS12
- 500 MHz data path, 166 MHz control path
- flit size of 3 words of 32 bits
- 500x32 = 16 Gb/s throughput per link, in each direction
- 256 slots & 5x1 flit fifos for guaranteed-throughput traffic
- 6x8 flit fifos for best-effort traffic





4. router architecture



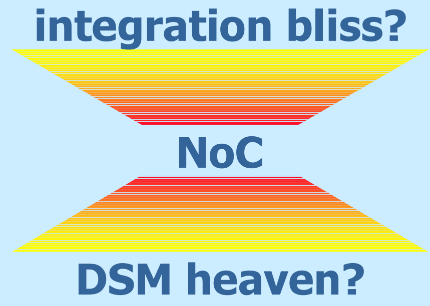
PHILIPS

5. conclusions

5. conclusions

- future applications are more dynamic and embedded
- users wants predictable and reliable behaviour
- QoS bridges this apparent contradiction

- future SoC design relies on NoCs to
  - solve DSM issues
  - close the design productivity gap



- QoS can be provided by the NoC protocol stack (services)
- but, the NoC architecture must take this into account

- there is an increasing awareness of the need for predictable system design and the role NoCs can play

5. conclusions

- the Æthereal NoC
  - offers differentiated services
  - with different types of commitment
- the Æthereal architecture
  - aims to marry guaranteed worst-case behaviour with good average resource usage
- Æthereal's prototype routers show feasibility of NoCs