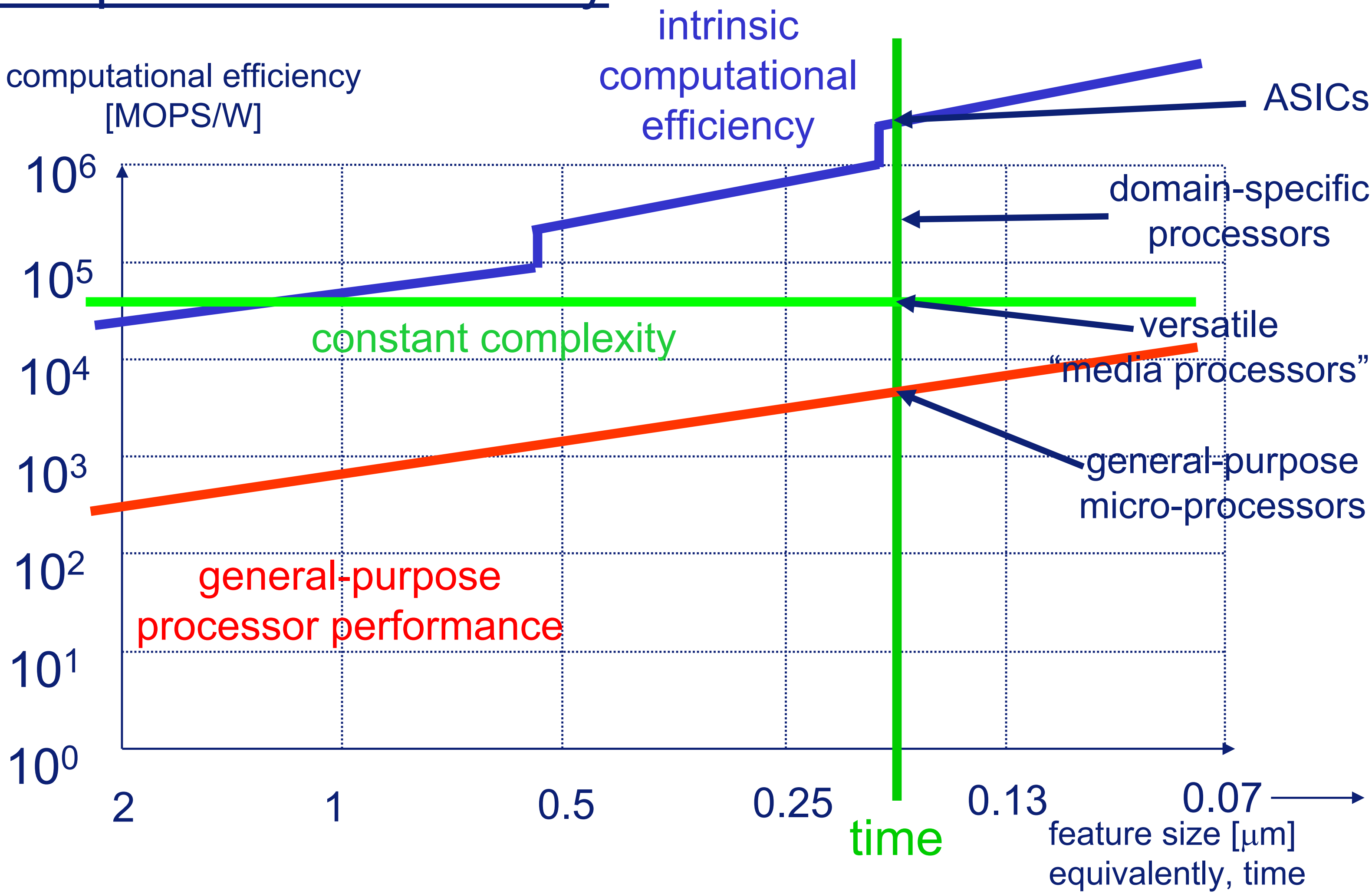
Predictable Systems

Reality, or just an illusion?

Kees Goossens
Philips Research

computational efficiency



computation

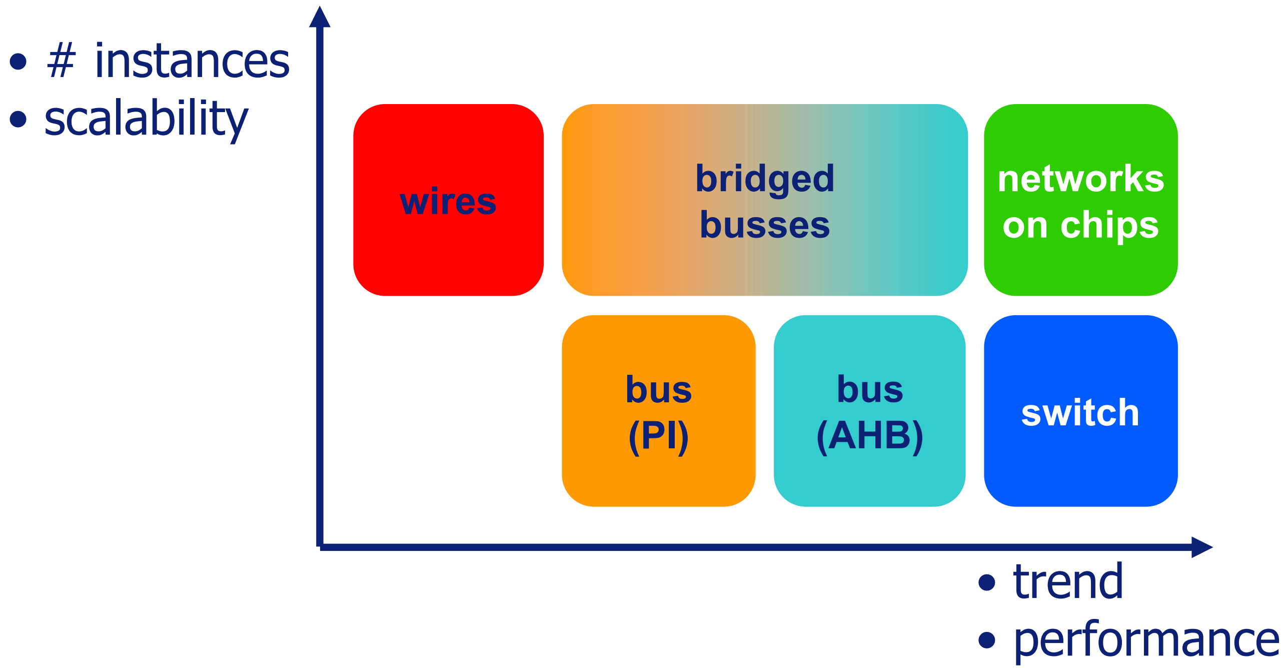
- flexibility becomes more important
- and also more affordable
- complexity requires scalability
- i.e. multiple cores

- # instances
- scalability

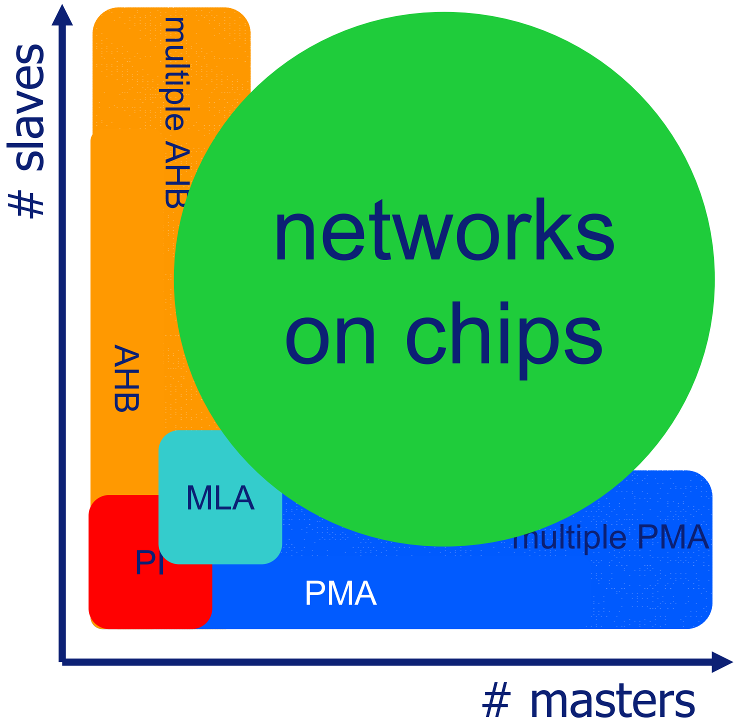
- trend
- flexibility

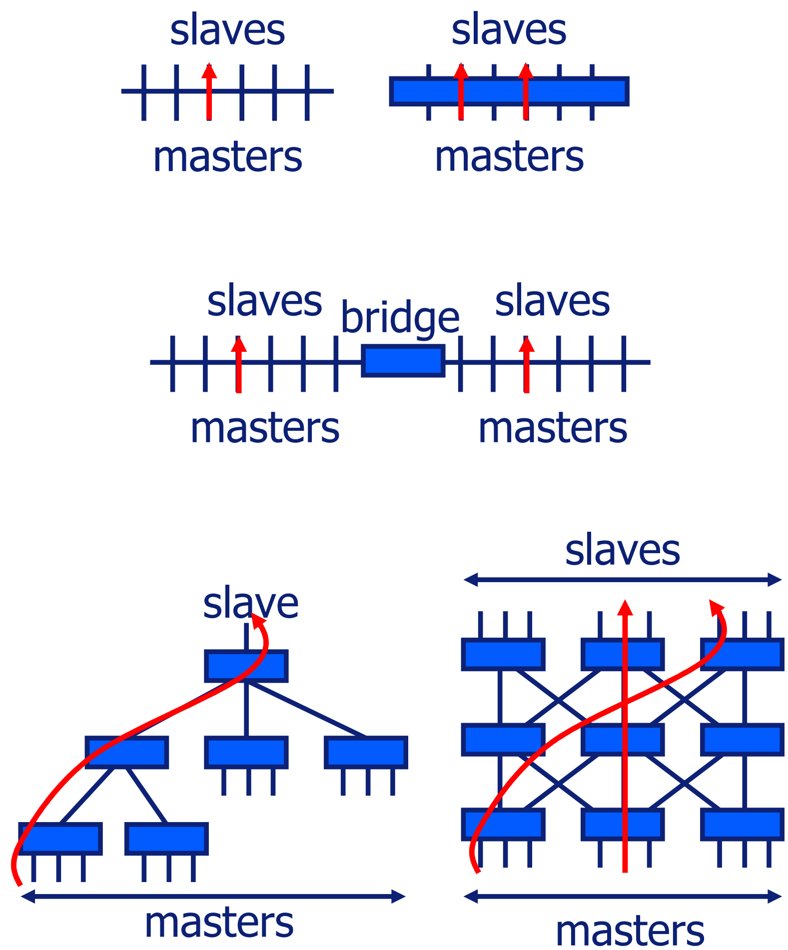
communication

- more IP ⇒ more interconnect
- a similar story…



communication



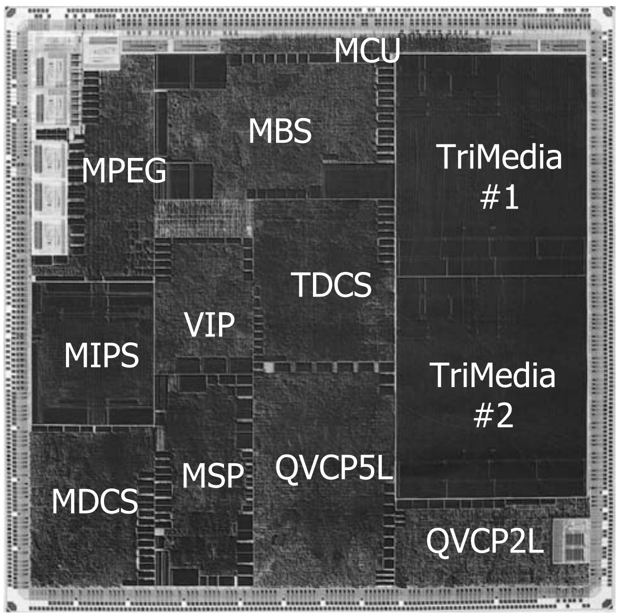


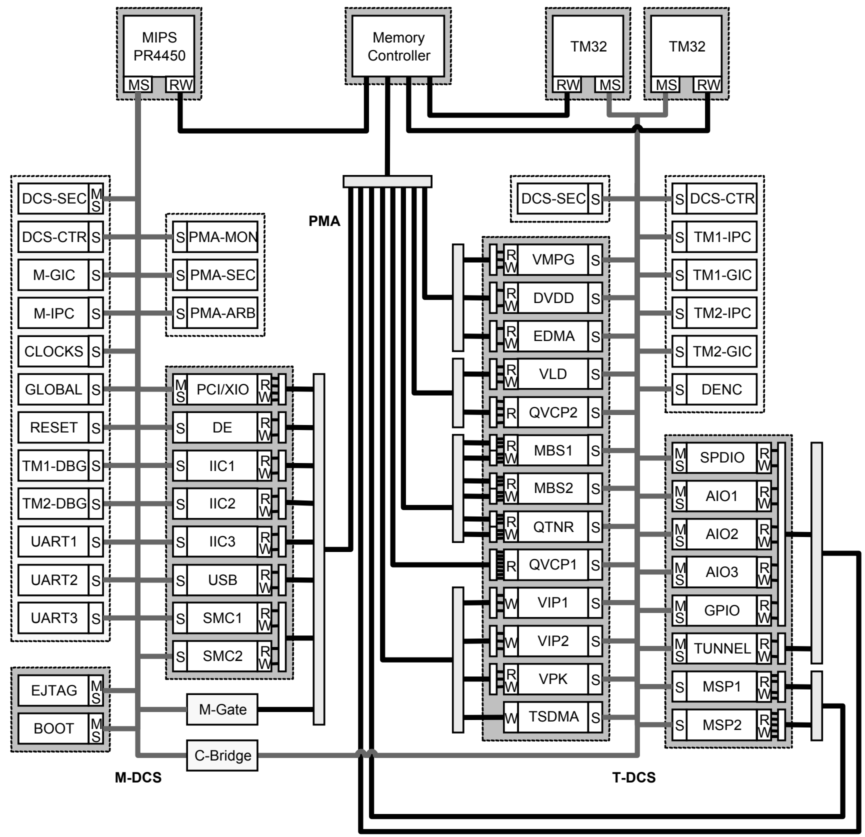
"Interconnect and memory organization in SOCs for advanced set-top boxes and TV," K. Goossens et al. in "Interconnect-Centric Design for advanced SoC and NoC," J. Nurmi, et al. (eds), Kluwer, 2004.

putting it all together

Viper2 (PNX8550)

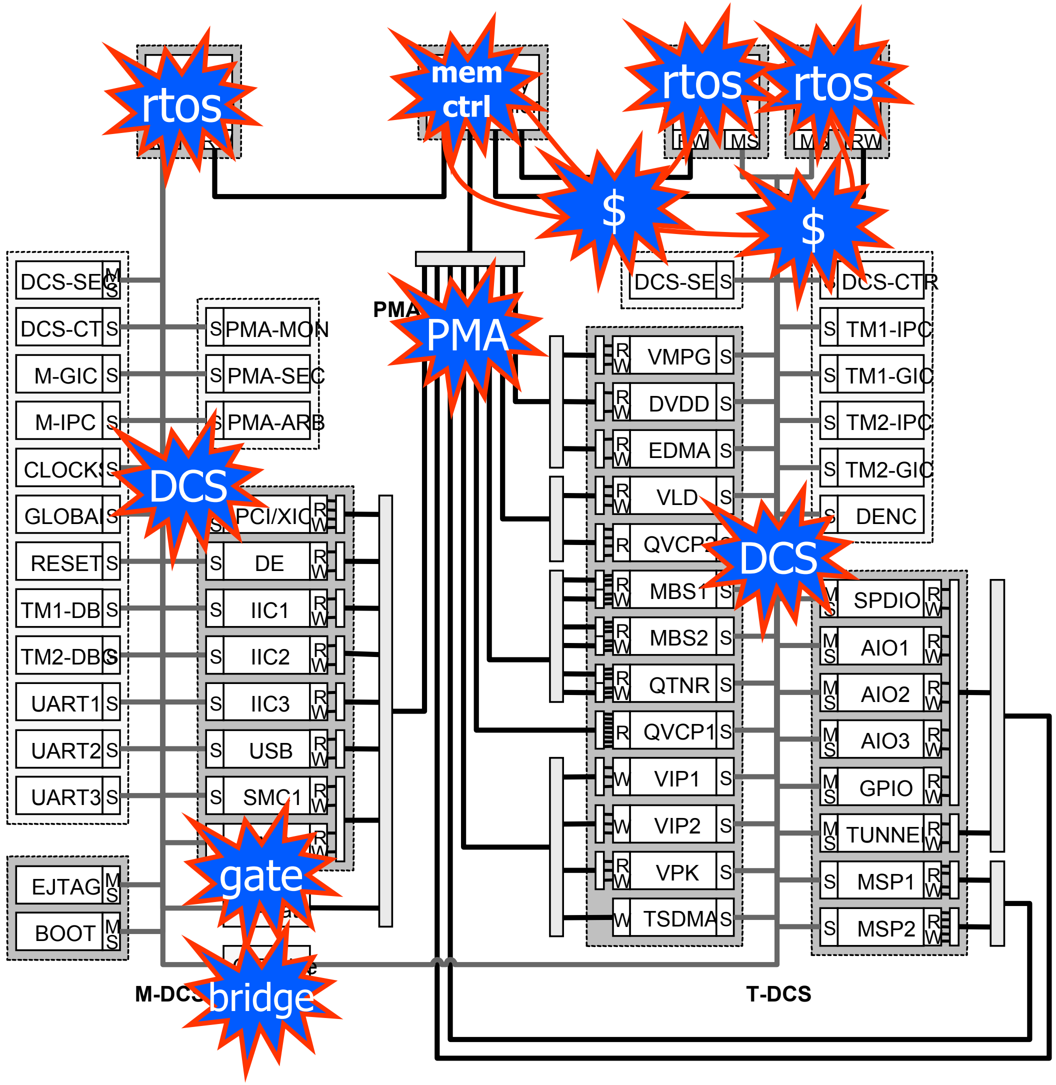
- 0.13 μm
- ~50 M transistors
- ~100 clock domains
- more than 70 IP blocks





unpredictability

- the challenge of designing these SOCs
- many resources
- many arbiters
- interference
- what is the resulting global behaviour?



why care about predictable systems?

- applications need it & users expect it
  - real time, embedded, safety critical
- ease design / lower TTM
  - enables compositional design style
  - enables compositional verification
    - functional & performance verification

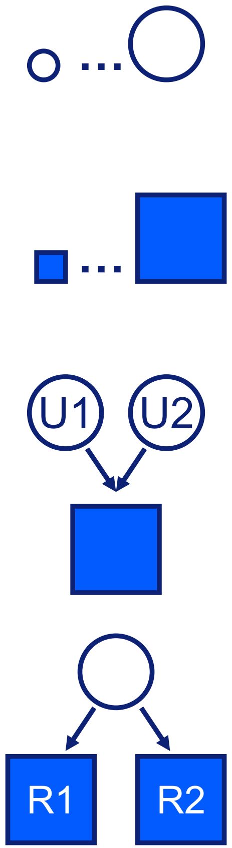
the scene

- SOC consists in resources
  - computation, storage, communication
- application uses resources
  - tasks, buffers, connections

  ~> computation, storage, communication
- where does the unpredictability come from?

the problem

1. resource usage is unpredictable
   – algorithmically difficult, data dependent
2. resources are unpredictable
   – DRAM, cache, power management, wires & gates
3. resources are shared by multiple users
   – require arbitration between users
4. users use multiple resources
   – dependencies & interference between arbiters, which possibly have different optimisation criteria

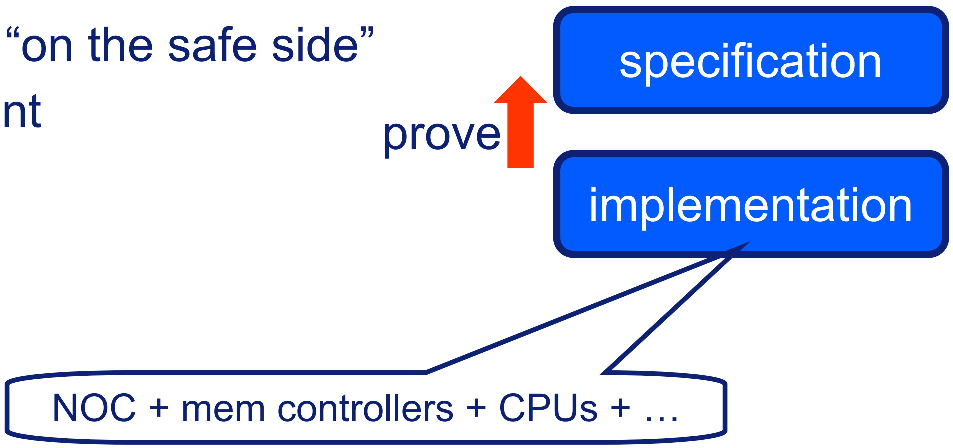


best effort approach

- accept variable performance of resources
- implement an arbiter per resource
- accept interference

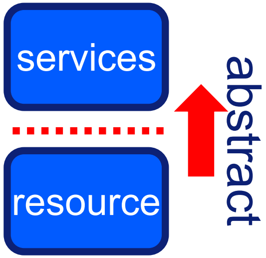
- simulate system specification against monolithic implementation
- fix problems that you find (by tweaking arbiters, or increasing resources)

- overdimensioning to be “on the safe side”
- no resource management

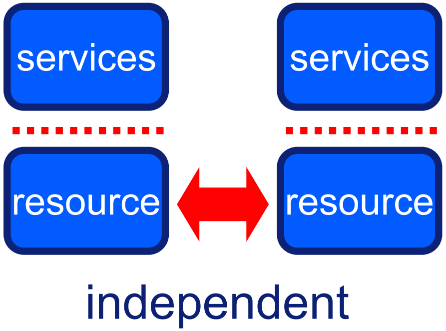


(quality of) service concepts

- services
  - abstract (simple) view on implementation
  - simplify reasoning about resource usage

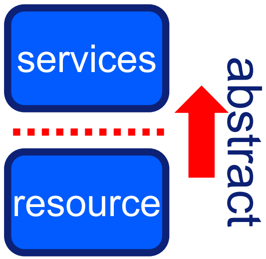


- guarantees
  - enable stronger (easier) reasoning / verification / analysis
  - enable compositional reasoning
- quality of service
  - reduce resource over-reservation
  - increase efficiency



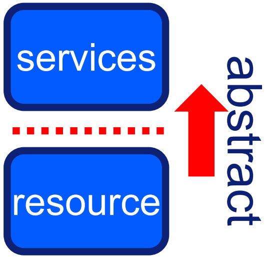
(quality of) service concepts

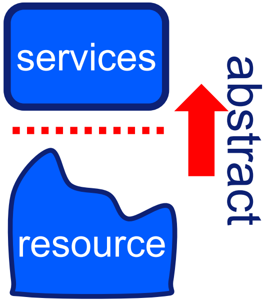
- services
  - abstract (simple) view on implementation
  - simplify reasoning about resource usage
  - hide internal dynamism & arbitration
    - offer 10 MB/s, hide contention & congestion in NOC, DRAM
    - offer 10 MIPS, hide RTOS & scheduling on CPU
    - offer performance level, hide calibration, voltage & frequency scaling

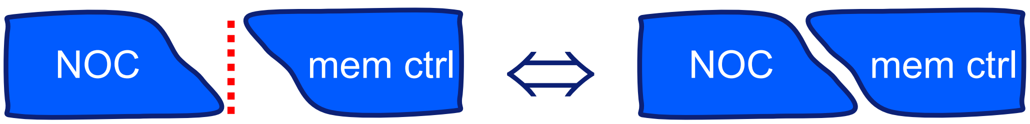


(quality of) service concepts

- services
  - hiding too much may make resource usage inefficient
  - services & implementation must be matched
    - e.g. don't offer rate-based throughput with TDMA
    - higher-level (better) services cost more

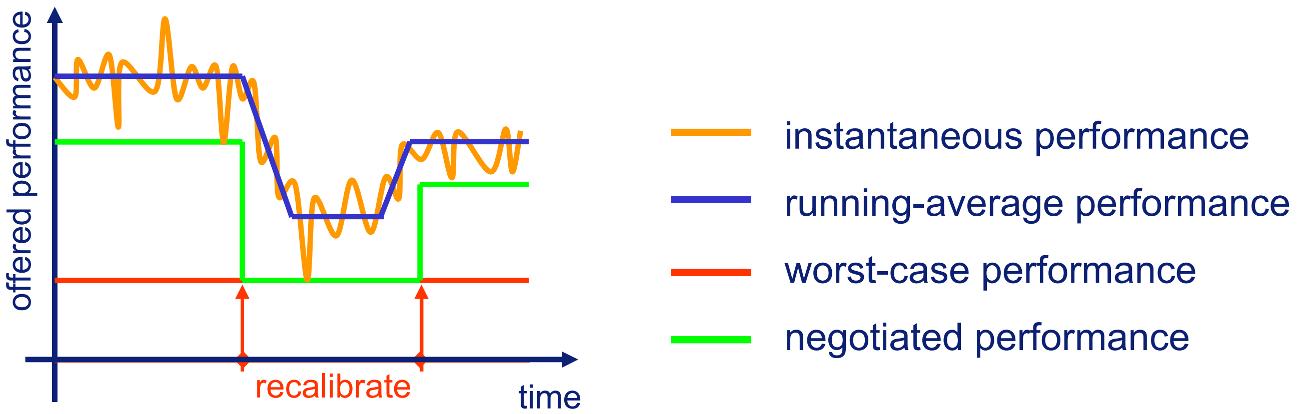






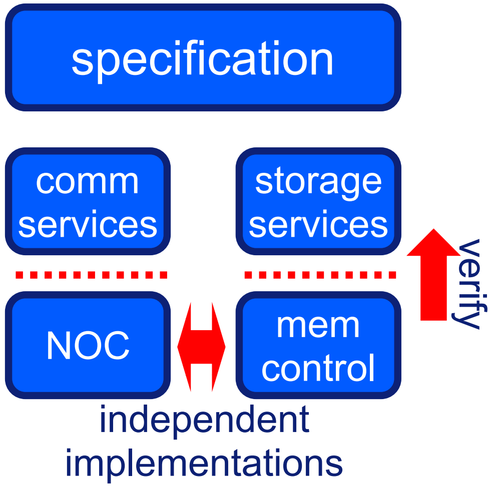
(quality of) service concepts

- service for unpredictable resources
  – resources with (algorithmic) performance of resource:
    - modify algorithm
      – DRAM: ok
      – cache: use as scratch pad
      – power management: ok
  – calibrate variable (hardware) performance of resource



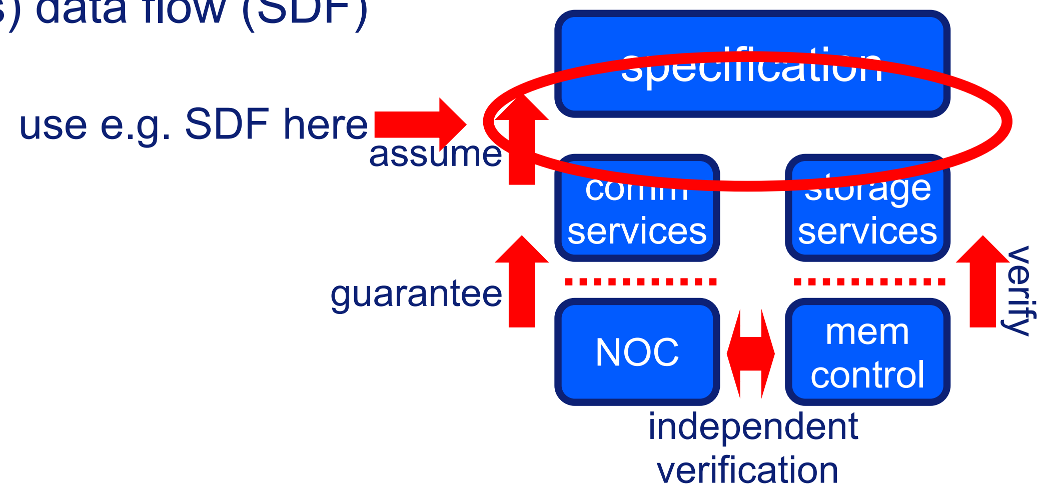
(quality of) service concepts

- guarantees
  - enable stronger (easier) reasoning / verification / analysis
    - “your data may arrive” vs. “your data will always arrive in 100ns”
    - stronger assumptions (services) ease proving the specification
  - usually entail resource reservation & management
  - makes IP/subsystems independent of rest of system



(quality of) service concepts

- guarantees
  - enable compositional reasoning
    - proofs of independent resources/sub-implementations are independent
    - “assume/guarantee” reasoning
  - must reason about all the different services
    - preferable using a common model
      e.g. (synchronous) data flow (SDF)



(quality of) service concepts

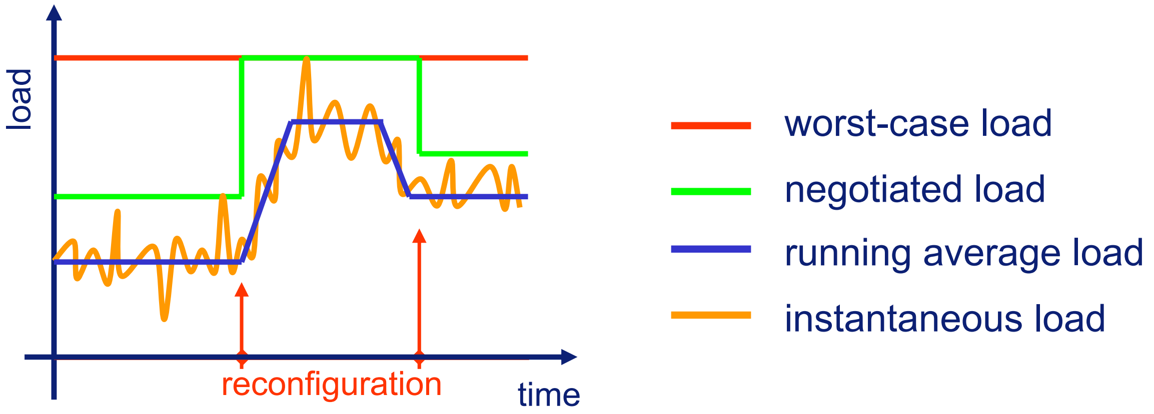
- quality of service
  - renegotiation for variable resource usage
    - reduce resource over-reservation
    - increase efficiency
  - alternatively, use multiple service classes
    - differentiated services, guaranteed & best-effort

the solution

1. resource usage is unpredictable
   - ✓ use QoS to characterise resource usage
2. resources are unpredictable
   - ✓ use calibration & predictable design
3. resources are shared by multiple users
   - ✓ use resource management & services
4. users use multiple resources
   - ✓ concerns are separated through guaranteed services

so, is predictability just an illusion?

- unpredictable resource usage
  - algorithms
    - worst-case is ok for many audio/video applications
    - reconfigure between steady states
  - we're looking into (synchronous) data flow (SDF)
    - worst-case execution times enable system-level analysis

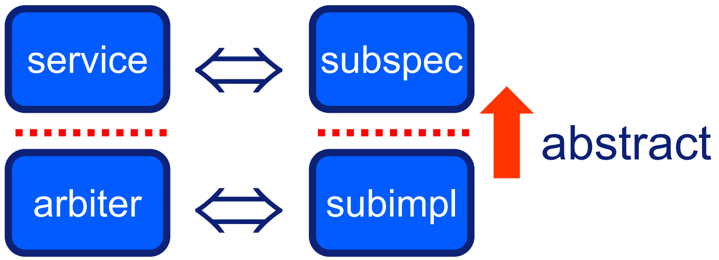


just an illusion?

- unpredictable resources
  - DRAM can be made predictable
  - process variation can be dealt with by calibration
  - power management: use calibration & make predictable
  - cache: not easy
    - use as local memory

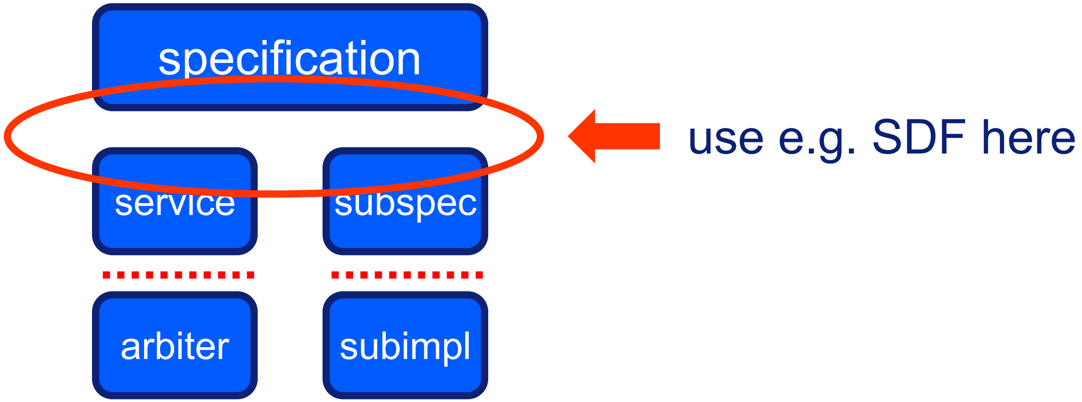
just an illusion?

- resource sharing / arbitration
  - for each service / interface
  - pick an arbiter that you can abstract well (e.g. TDMA, RR)
    - also to get good implementation-service match



just an illusion?

- multiple resources / interference
  - all services must work / be analysable together
    - e.g. NOC & RTOS services
    - use e.g. SDF as the common model to reason about services

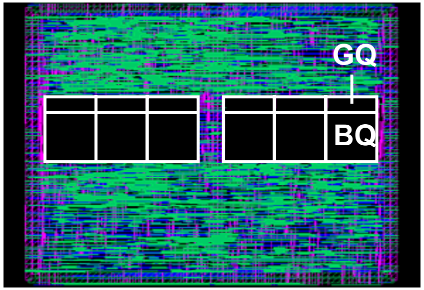


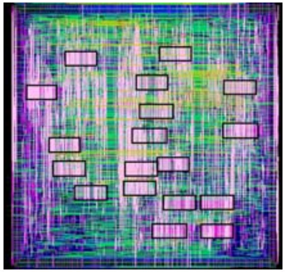
concrete example

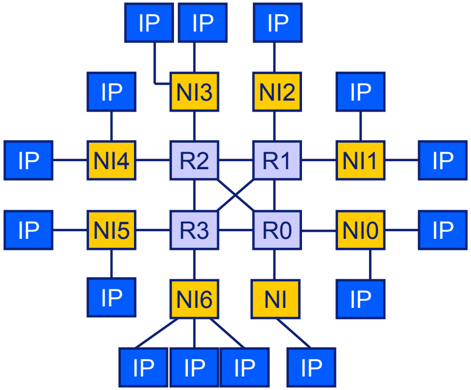
- Æthereal network on chip
  - decouple IP implementations through separation of computation & communication
  - focussed on guaranteed communication services
  - also offer best effort for high resource utilisation
  - fast performance verification of communication
  - decouple interconnect & IP verifications

foundations of the Æthereal flow

- parametrised building blocks
  - router
    - arity, buffer sizes
  - network interface (NI)
    - slot table size
    - #ports & their type
    - #connections per port
    - buffer sizes per connection
- they can be flexibly
  - instantiated
  - connected
  - programmed







Æthereal NOC design flow

- ✓ fast automatic generation and verification
- ✓ guaranteed performance without simulation

- ✓ simplifies back-end flow
- ✓ complies with & enhances platform
  - compliant / backward compatibility
  - future proof

- ✓ quickly verify applications on chip
- ✓ run-time re-configurable
  - like any IP, using memory-mapped IO

- NOC dimensioning

  ⇓
- NOC configuration

  ⇓
- NOC verification

  ⇓
- NOC simulation

conclusions

- trend towards multiple shared resources
- as a result
  - increased arbitration and interference
  - difficult to check if system meets its (RT) specification

- guaranteed services and QoS are essential for
  - compositional system design
  - compositional (performance) verification

- predictable systems require QoS-aware
  - resources (underlying hardware: calibration, storage, computation, communication architectures)
  - resource users (especially software)

PHILIPS