

The *Engineering* of Software

Bran Selic

IBM Software Group – Rational Software

bselic@ca.ibm.com

A Great Pioneer Speaks...



Edsger Wybe Dijkstra (1930 – 2002)

- ◆ *“I see no meaningful difference between programming methodology and mathematical methodology” (EWD 1209)*

Two Opinions

“Because [programs] are put together in the context of a set of information requirements, they observe no natural limits other than those imposed by those requirements. Unlike the world of engineering, there are no immutable laws to violate.”

- Wei-Lung Wang
Comm. of the ACM (45, 5)
May 2002

“All machinery is derived from nature, and is founded on the teaching and instruction of the revolution of the firmament.”

- Vitruvius
On Architecture, Book X
1st Century BC

What is Engineering?

◆ *Merriam-Webster Collegiate Dictionary:*

engineering: the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people

- ◆ What does this have to do with software design?
 - “...*no natural limits...no immutable laws to violate*”

What is Software Made of?

Exhibit A: Transmission Delay Effects

- ◆ Possibility of out of date status information

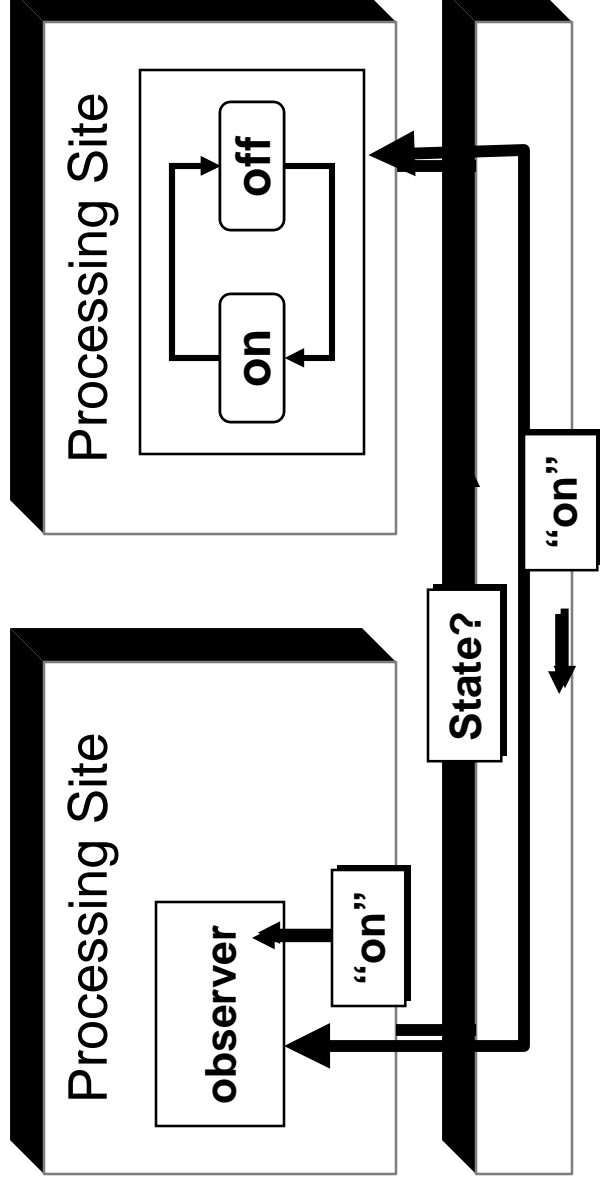
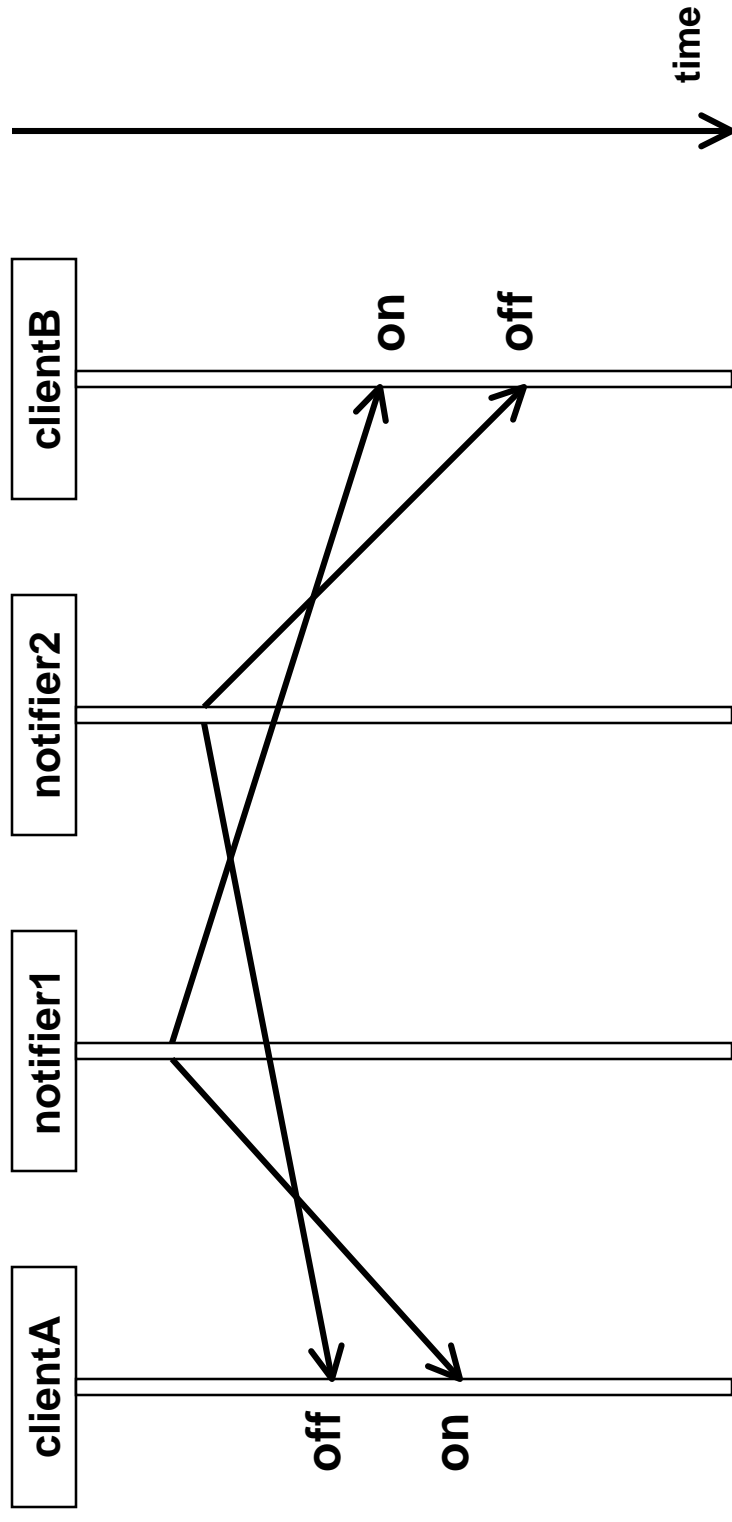


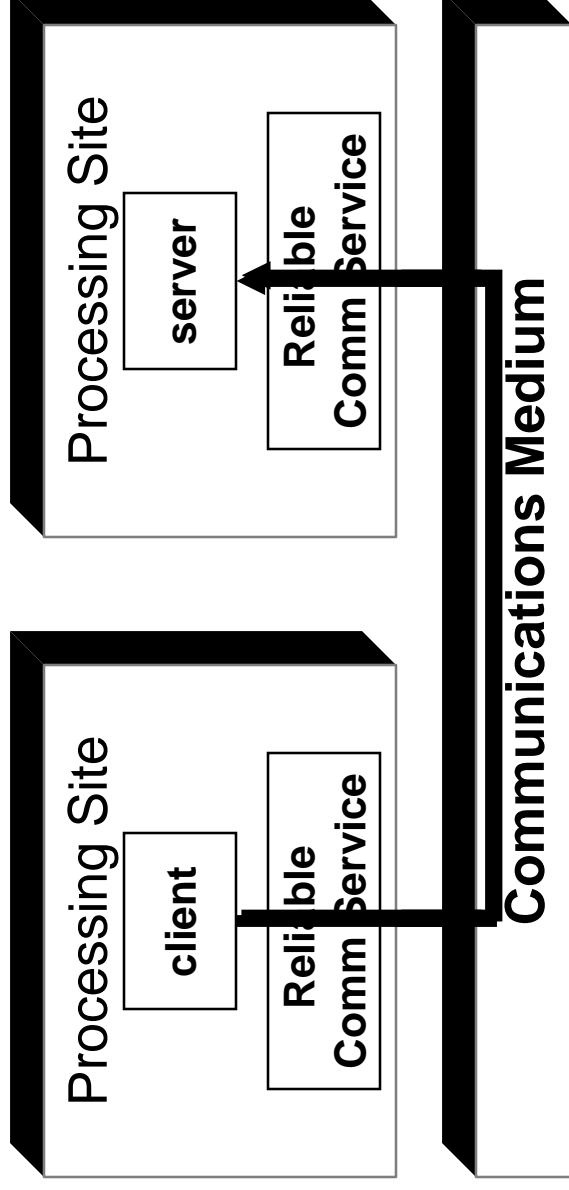
Exhibit B: Relativistic Effects

- ◆ Relativistic effects:
 - different observers see different event orderings (due to different and variable transmission delays)



Distribution Transparency Mechanisms

- ◆ Platform layers that mask out failures from the application
 - e.g., reliable RPC services, relocation transparency,...



Impossibility Result No.1

It is not possible to guarantee that agreement can be reached in finite time over an asynchronous communication medium, if the medium is lossy or one of the distributed sites can fail

- Fischer, M., N. Lynch, and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process" *Journal of the ACM*, (32, 2) April 1985.

Impossibility Result No.2

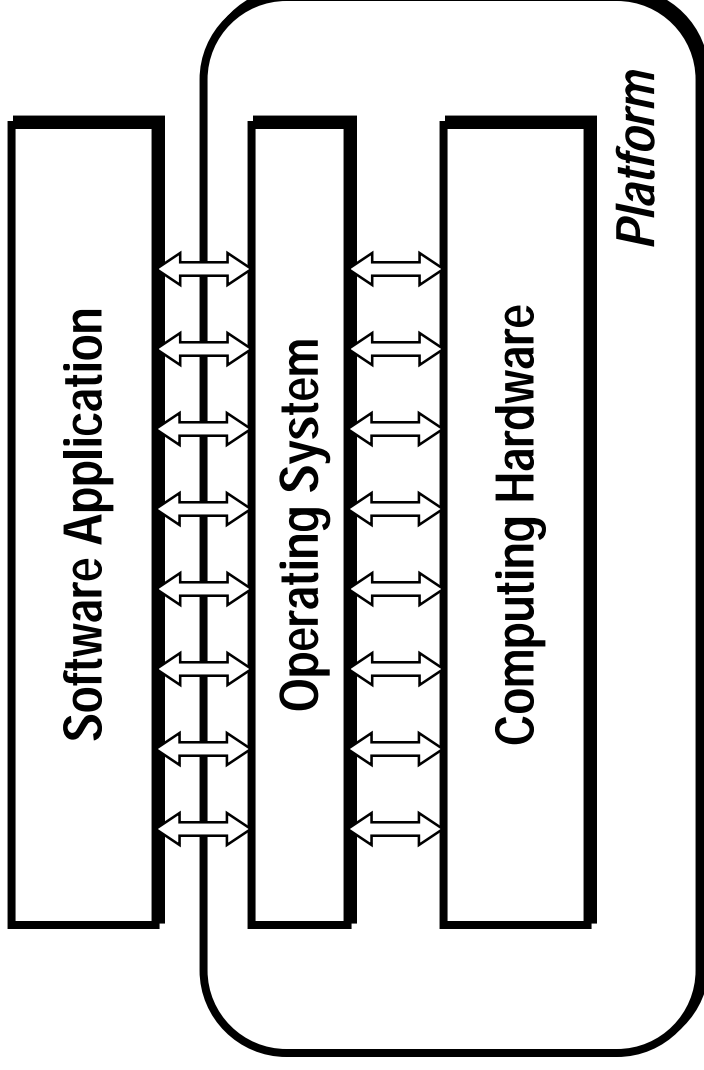
Even when communication is fully reliable, it is not possible to guarantee common knowledge if communication delays are unbounded

- Halpern, J.Y, and Moses, Y., "Knowledge and common knowledge in a distributed environment" *Journal of the ACM*, (37, 3) 1990.

Transparency Mechanisms?

- ◆ *All distributed transparency mechanisms require distributed agreement!*
 - Transparency can only be approximated
 - ⇒ *the application may still have to deal with the unpleasant side-effects of distribution*
 - The more transparency is desired the higher the cost (time, resources, complexity)
- ◆ *The end-to-end argument [Saltzer et al.]:*
 - ⇒ *the overhead introduced by transparency mechanisms may outweigh any benefits*

What Software is Made of



- ◆ *Platform* = the complete technological base (SW and HW) required to execute an application
- ◆ The platform is the “construction material” of software, conveying its physical characteristics (speed, capacity, etc.) directly to the application

Platforms and Applications

- ◆ *Q: What effect should a computing platform have on an application?*
- ◆ *A: as little as possible*
...but, no less!
- ◆ *Platform-independent design (MDA)*
 - Separation of concerns (simplifies design)
 - Portability
- ◆ A sound design principle that is far too often misinterpreted as “platform ignorance”

If Transparency is an Idealization...

- ◆ Facts to ponder:
 - In the Internet Age, most interesting applications will be distributed
 - As our dependence on computers increases, the physical characteristics of our software (response time, availability) will become much of a concern
- ◆ *Traditional Programming = Logic*
- ◆ *Physical Programming = Logic + Physics*
 - Like more traditional engineers, software designers must take into account the construction material out of which the logic is spun
 - Dealing with finite resources, finite delays, finite reliability...
- ◆ *“All machinery is derived from nature, and is founded on the teaching and instruction of the revolution of the firmament.”*

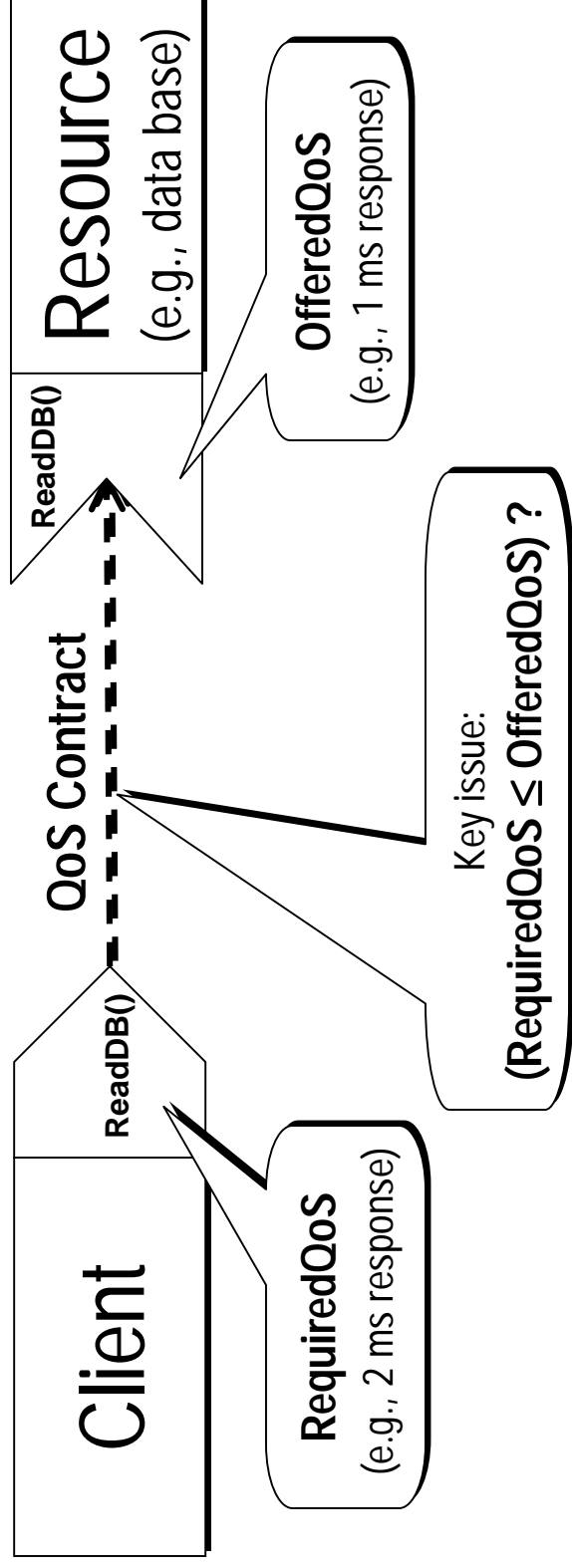
Core Concepts for “Physical” Programming

Quality of Service

- ◆ The physical characteristics of software can be specified using the general notion of *Quality of Service (QoS)*:
 - *a specification of how well a service can (or should) be performed*
 - throughput, latency, capacity, response time, availability, security...
 - usually a quantitative measure
- ◆ QoS concerns have two sides:
 - *offered QoS*: the QoS that is available
 - *required QoS*: the QoS that is required to do a job

Resources and QoS Contracts

- ◆ **Resource:**
an element whose ability or capacity is limited, directly or indirectly, by the finite capacities of the underlying physical elements
- ◆ The relationship between resources and resource users

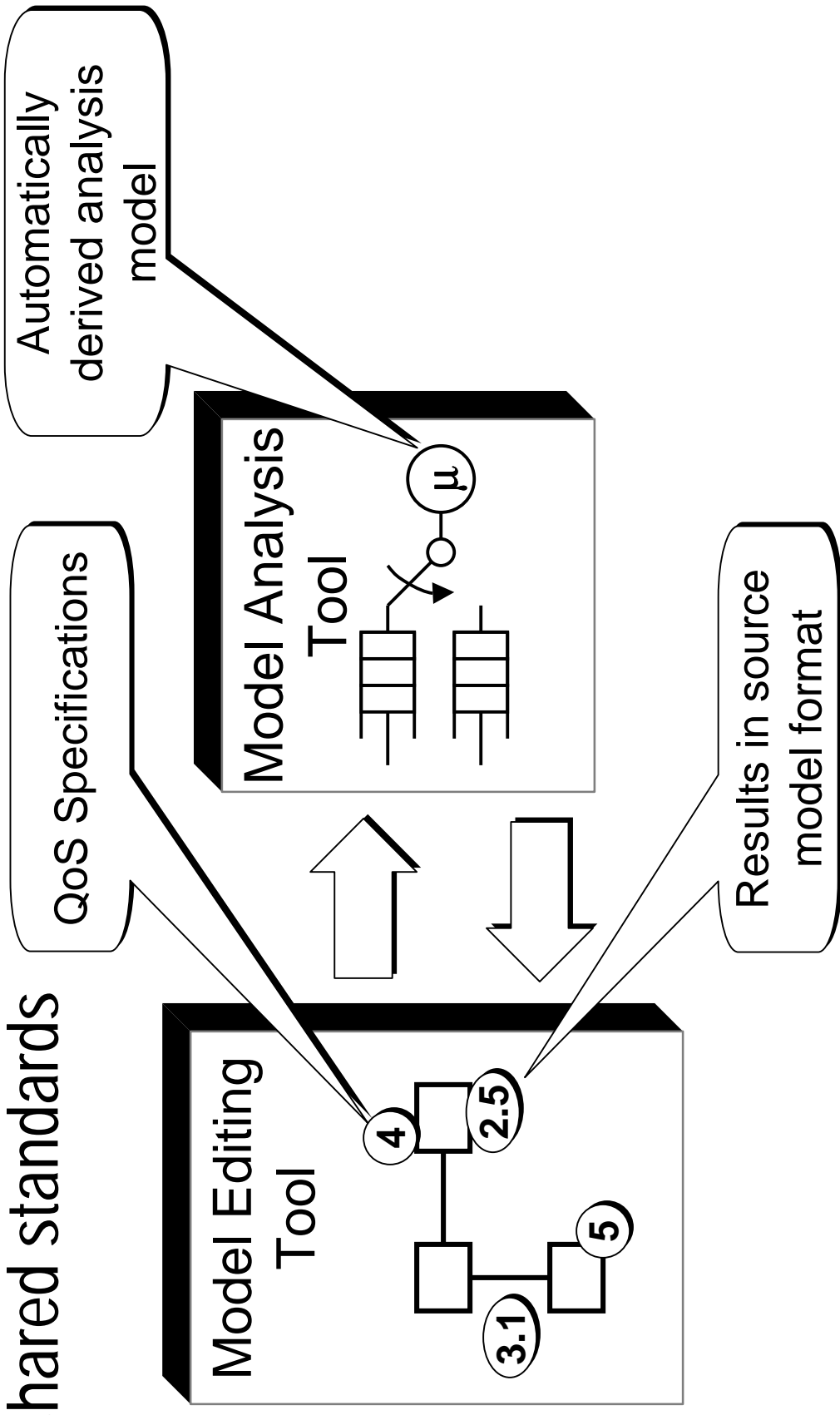


QoS Contract Verification

- ◆ Can QoS contracts be statically checked by a compiler?
 - The good news: Yes (in most cases)
 - The bad news: typically requires complex analysis methods (queueing network analysis, schedulability analysis, etc.)
- ◆ Some issues:
 - In most cases QoS verification cannot be done incrementally – the full system context is required
 - Each type of QoS (e.g., bandwidth, CPU performance) combines differently – no general theory for QoS analysis
- ◆ However, much of this can be automated

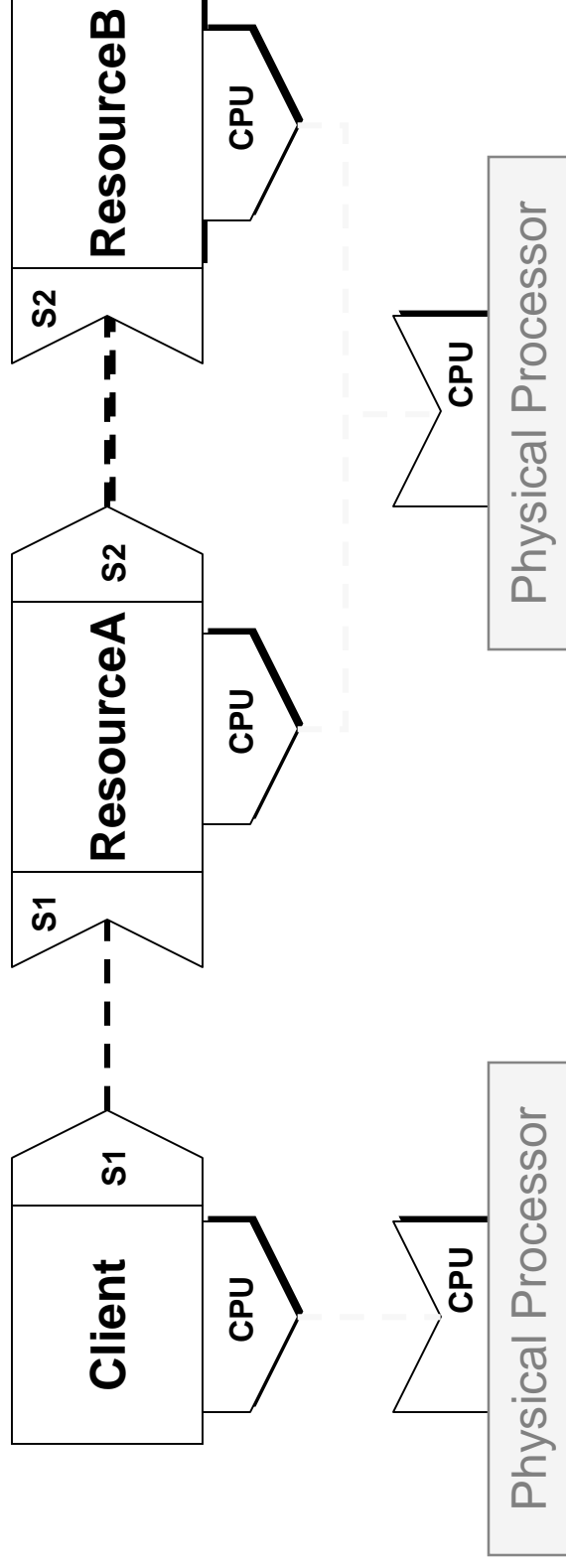
Automating QoS Verification

- ◆ Seamless inter-working of specialized tools based on shared standards



Offered vs. Required QoS

- ◆ Like all guarantees, the offered QoS is *conditional* on the resource itself getting what it needs to do its job
- ◆ This extends in two dimensions:
 - the *peer* dimension
 - the *layering* dimension: for platform dependencies



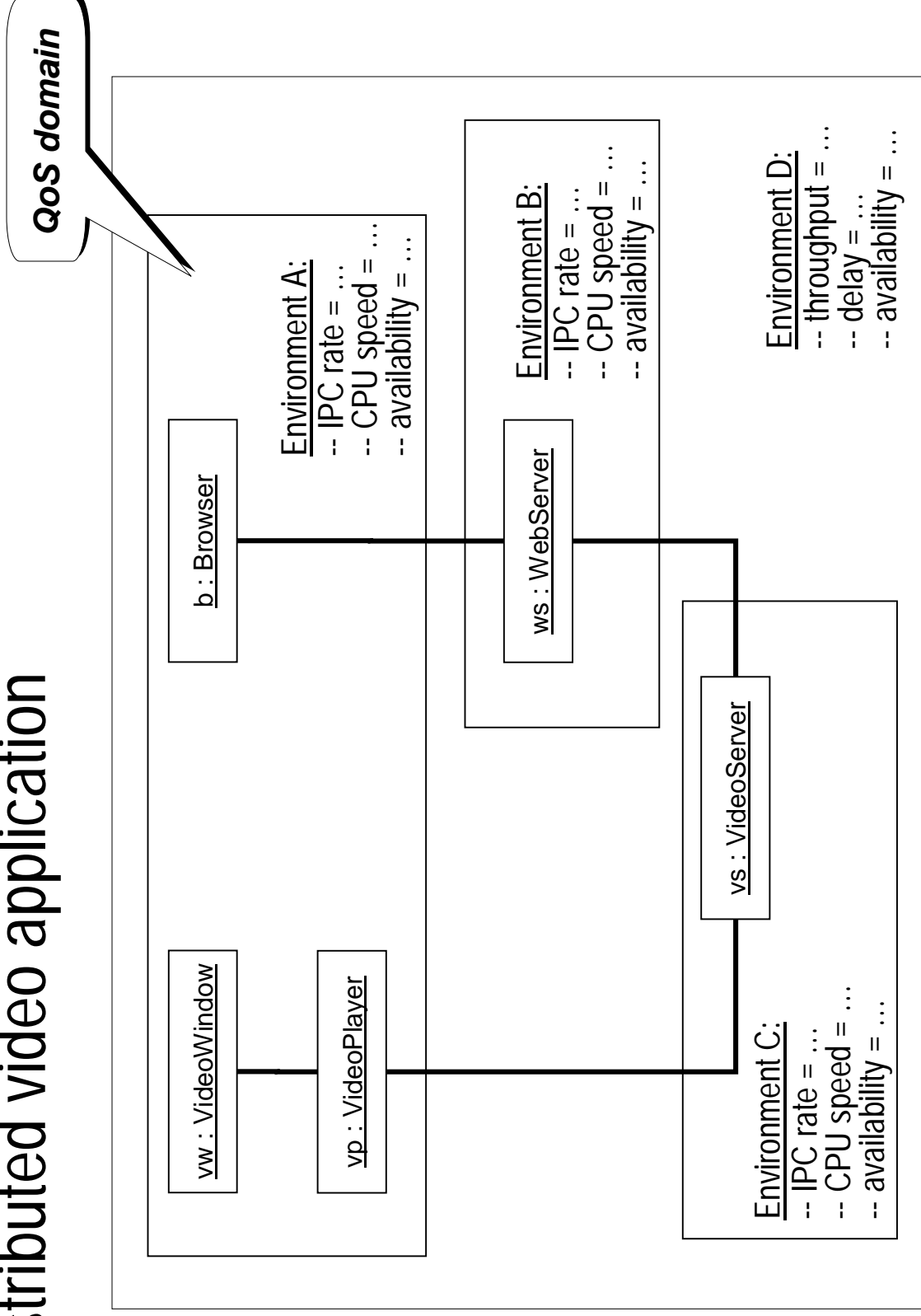
Platform-Aware Platform-Independent Design

Platform Awareness

- ◆ Awareness of possible platform effects
 - Performance, availability, ...
 - E.g., can two components fail independently of each other?
 - E.g., how reliable is the communication between two components?
- ◆ Eliminates the need for worst-case assumptions that the end-to-end argument implies

Specifying Platform Assumptions

◆ Distributed video application



QoS Domains

- ◆ A domain in which certain QoS values apply uniformly:
 - CPU performance
 - communications characteristics (delay, throughput, capacity)
 - failure characteristics (e.g., availability, reliability)
 - etc.
- ◆ The QoS values of a domain can be compared against those of any concrete platform to determine its suitability

Conclusions

- ◆ Most interesting systems of the future will be embedded in the real (physical) world
 - E.g., Internet-based systems
 - Ignoring platform characteristics will lead to significant project failures
 - ...yet, platform ignorance has been raised to the level of a design principle!
- ◆ Engineering-oriented software development is based on use of
 - Models and modeling techniques
 - Formal analysis techniques
 - ...including the use of quantitative analyses (e.g., QoS-based methods)
- ◆ These developments are the necessary conditions for the fundamental advance needed to improve the quality of software development