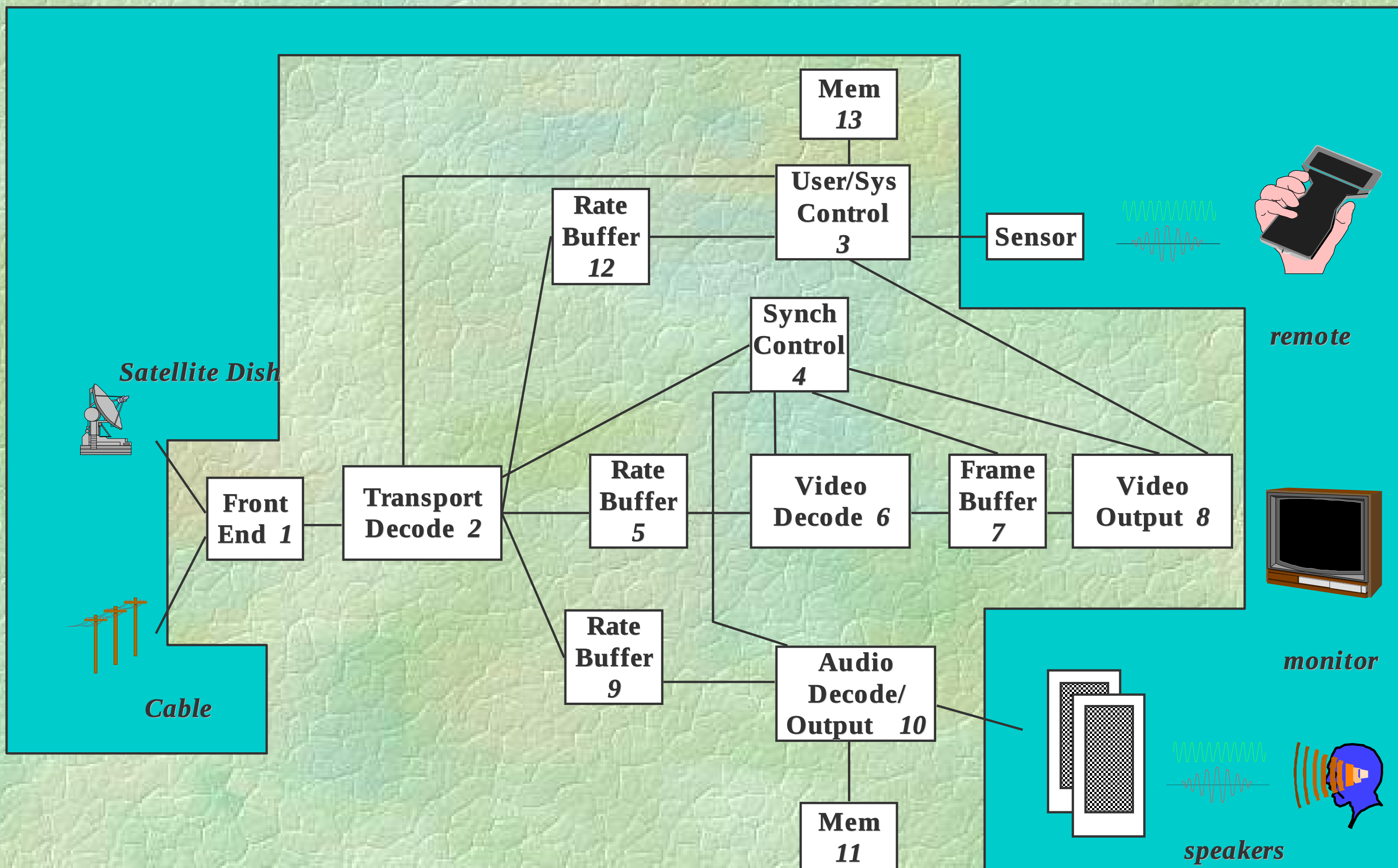
Component-Based Software
State-of-the-Art and Lessons Learned

A. Richard Newton

Department of Electrical Engineering and Computer Sciences

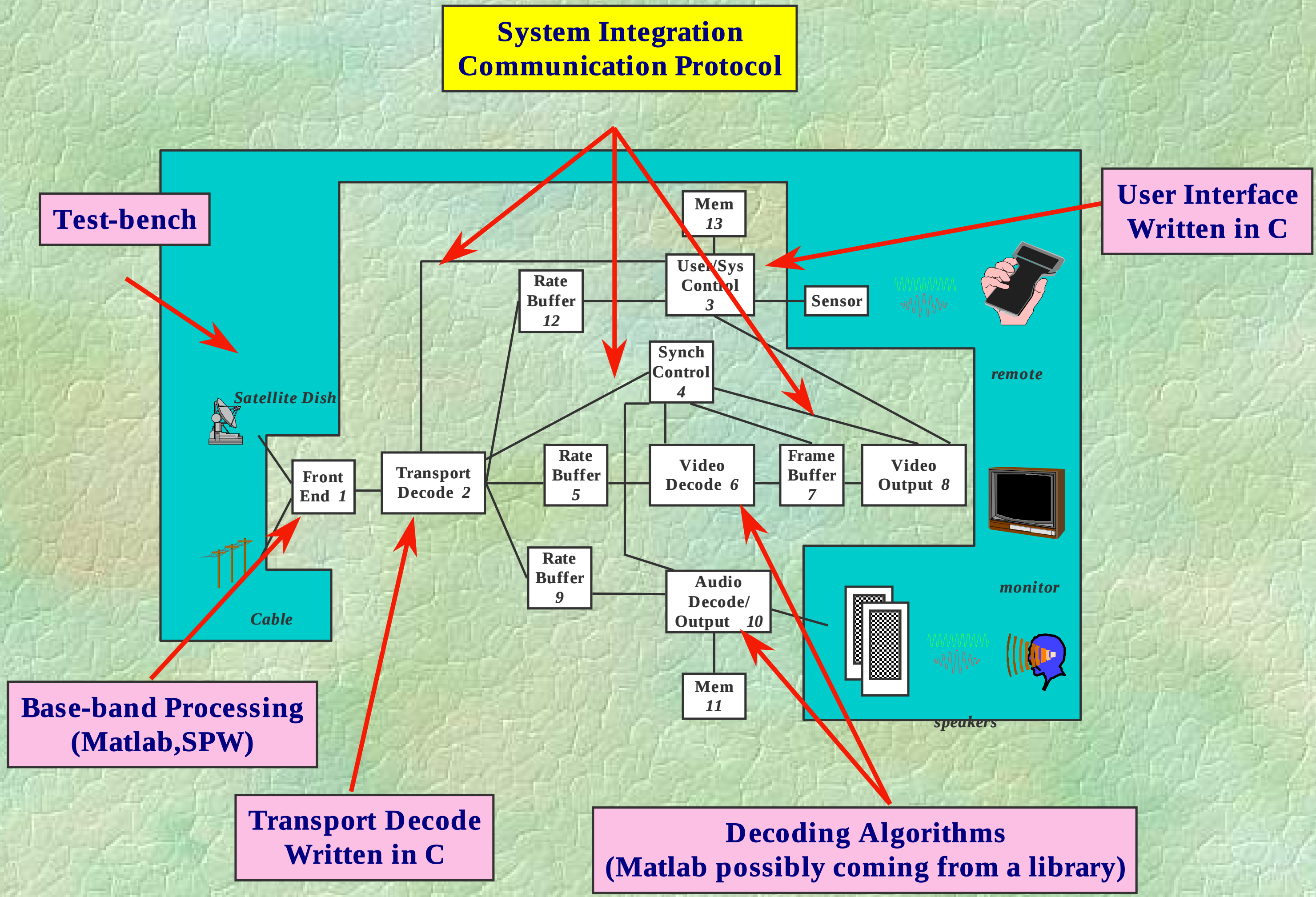
University of California at Berkeley

Example of System Behavior



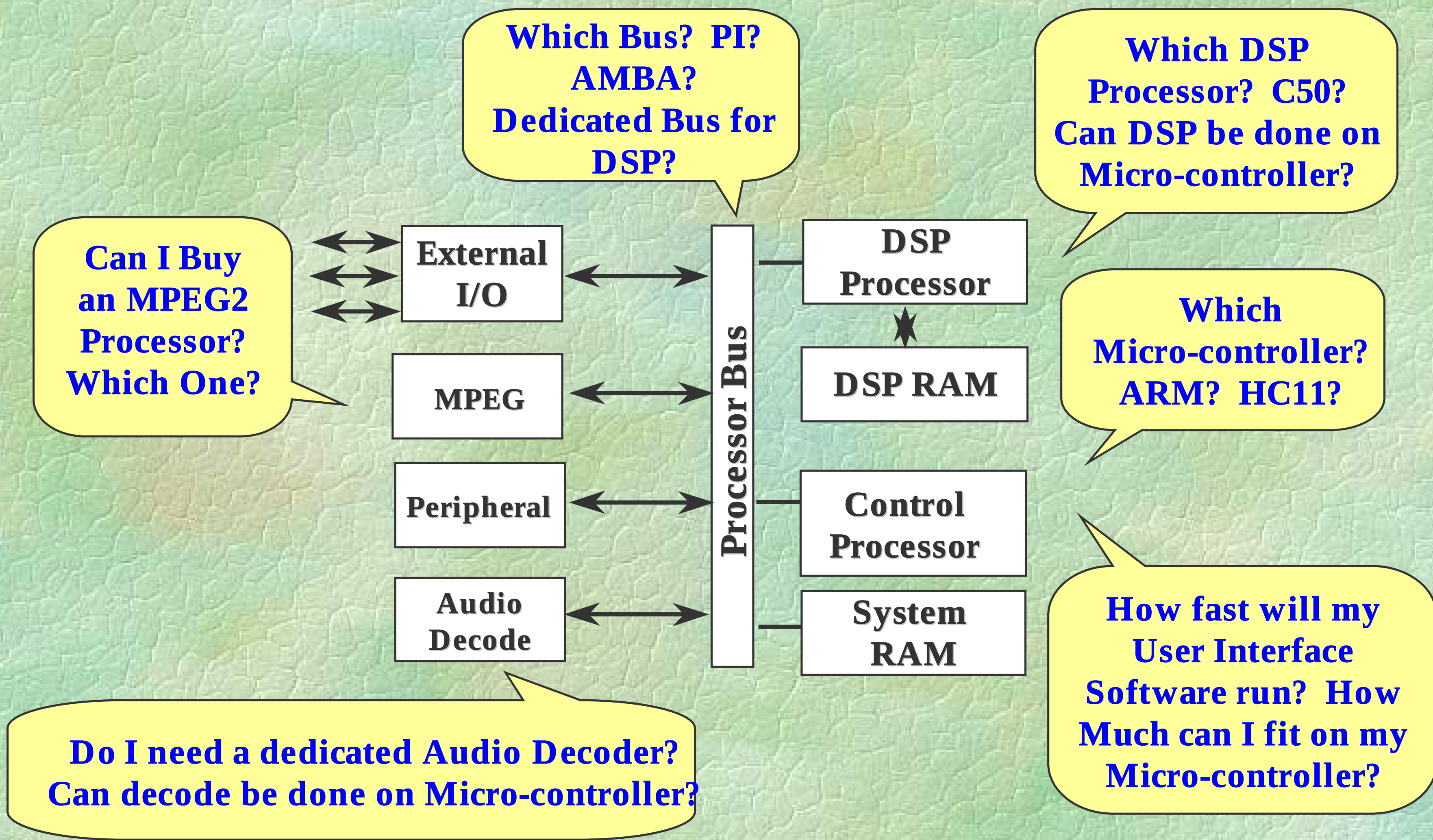
Source: Prof. Alberto Sangiovanni

IP-Based Design of Behavior



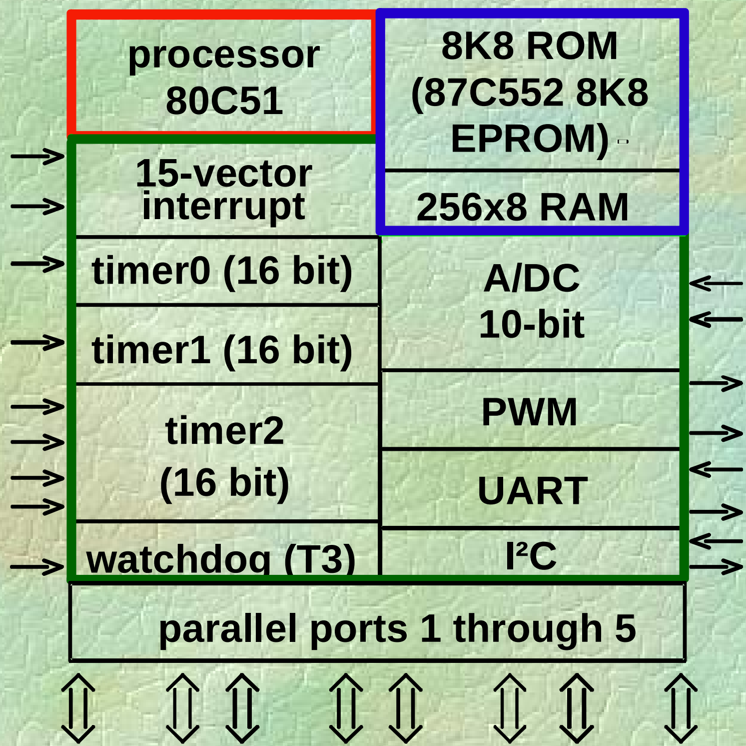
Source: Prof. Alberto Sangiovanni

IP-Based Design of Implementation



Source: Prof. Alberto Sangiovanni

A Complete System-on-a-Chip



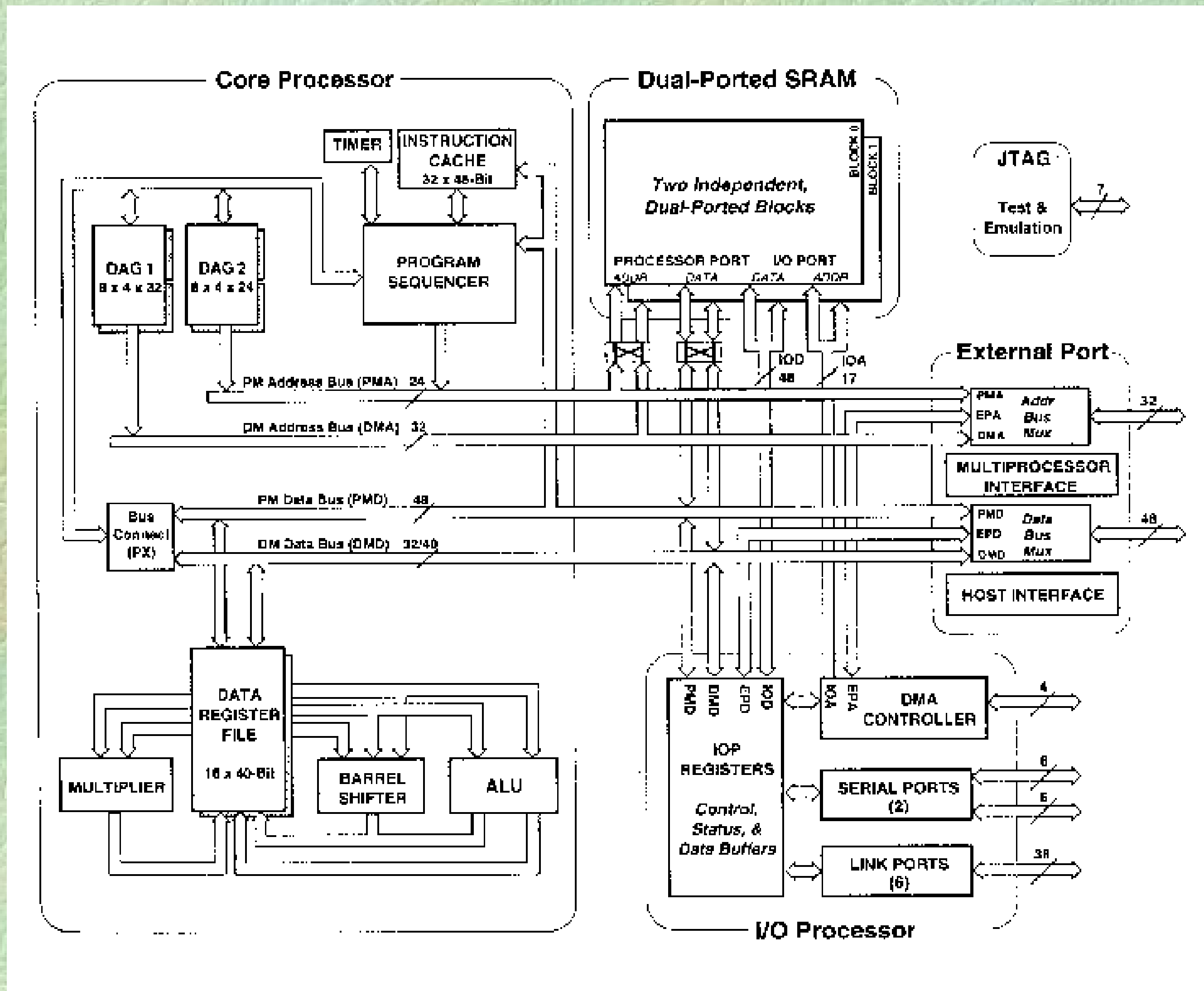
- complete system
- timers, PWM for control
- I²C-bus and par./ser. interfaces for communi-cation
- A/D converter
- watchdog (SW activity timeout): safety
- on-chip memory
- interrupt controller

Philips 83 C552: 8 bit-8051 based microcontroller

control dominated systems

Source: Prof. Rolf Ernst

ADSP21060 SHARC (similar to processor: TMS320C40)



data dominated systems

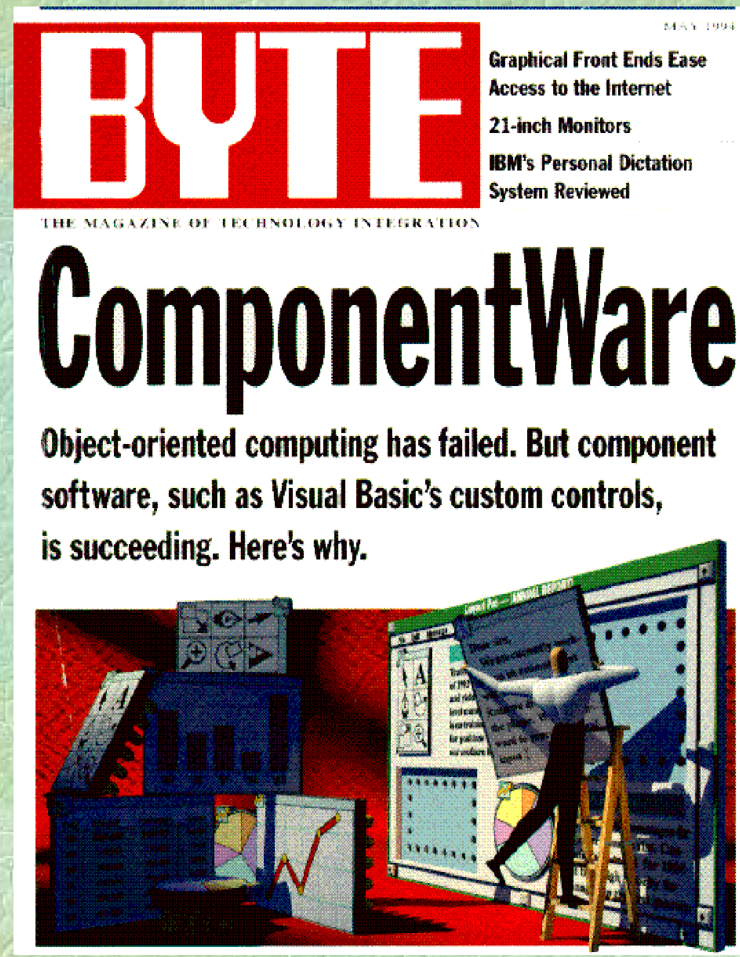
Source: Prof. Rolf Ernst

Observation: Top-Down versus Bottom-Up

- Clearly, design is some combination of top-down and bottom-up aspects
  - In a top-down implementation emphasis, optimal partitioning drives the process
  - In a bottom-up implementation emphasis, optimal combination of existing components drives the process
- Implies efficient evaluation, composition, and deployment of components (from a variety of sources) is the key emphasis in bottom-up implementation styles

What is *ComponentWare* in a Software Context?

- Visual Basic: ActiveX controls
- Windows Programmers: DLLs
- JavaBeans
- CORBA
- Microsoft COM (DCOM, COM+)
- i.e."Binary-Level" reuse



COM Component Market (excluding Microsoft) $300M in 1997, going to $3B in 2000

(Objects are almost never sold, bought, or deployed!)

Definition of a Software Component

"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties"

Components are for Composition

Nomen est Omen

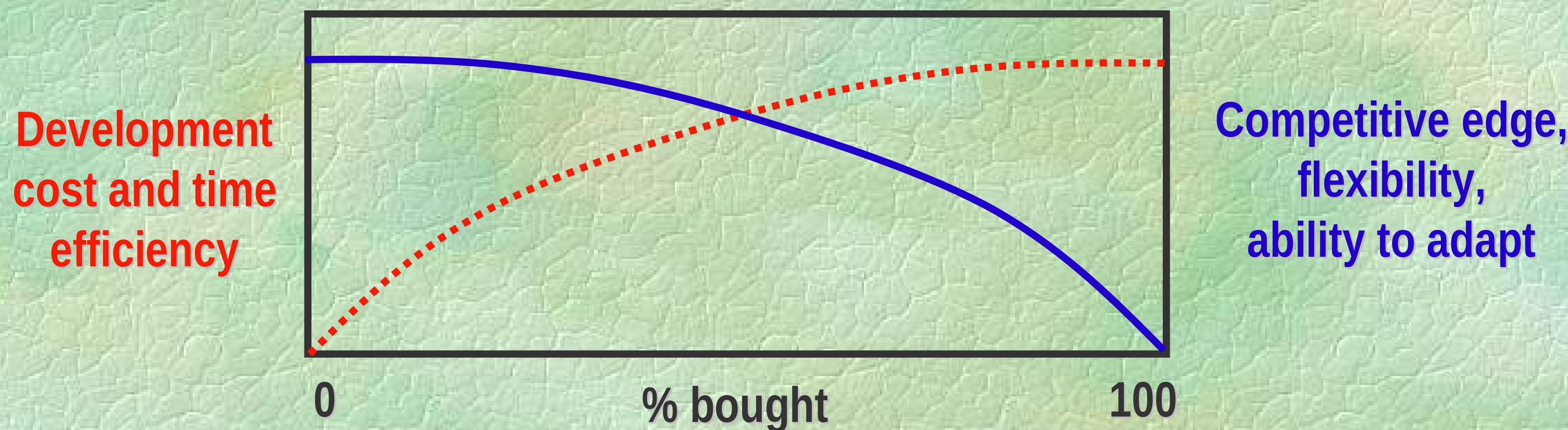
- Enables prefabricated ‘things’ to be reused by rearranging them in ever-new composites
- Aren’t most current software abstractions designed for composition as well?
- Isn’t reuse the driving factor behind almost all compositional abstractions?
- Software components are “binary units of independent production, acquisition, and deployment that interact to form a functioning system” Szyperski, “Component Software,” 1998

Component Software

- Requirement for independence and binary form rules out many software abstractions
- Early literature on components (motivated by the 'software crisis'), refers to them as "Software IC's" (McIroy, 1968; Cox 1986).
- Why components?
  - All other engineering disciplines introduce components as they mature
  - And they continue to use them

Component Software

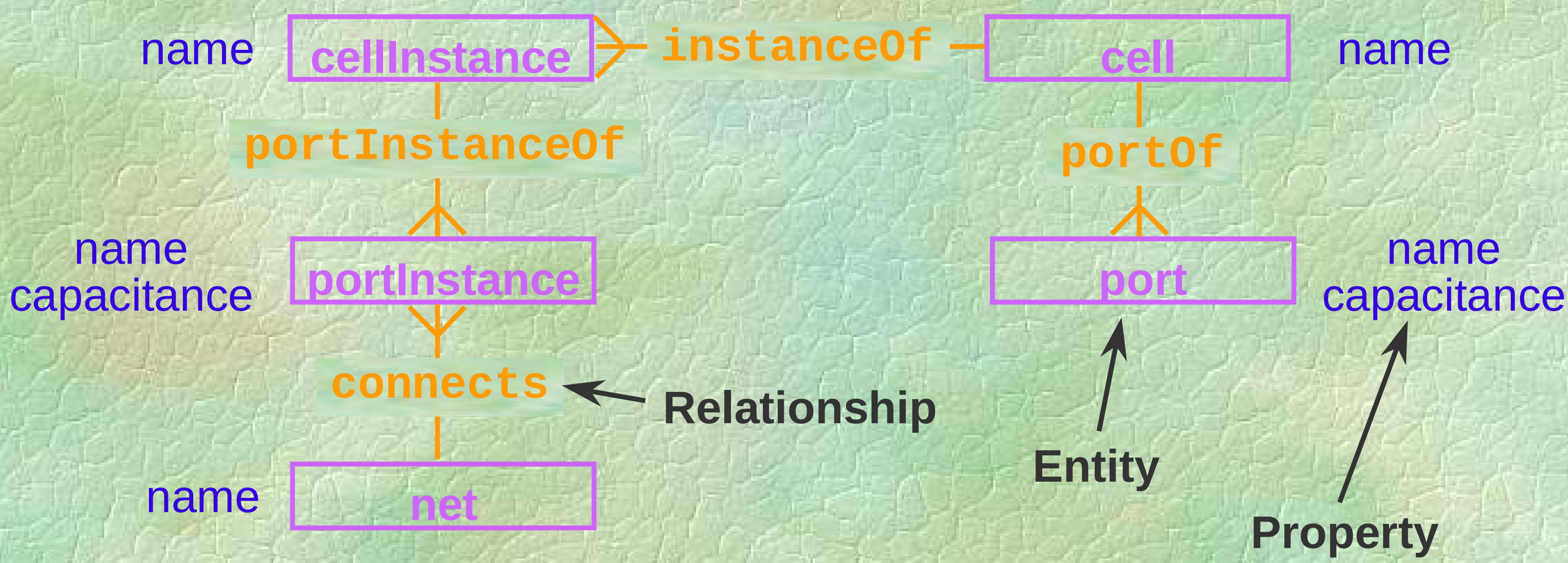
- Does not need to be a single component approach to integration
  - Any successful approach must reach critical mass
  - Critical mass requires sufficient variety and quality
  - Second sources often required as well
- Can also be used to offer a modular approach to products (stereos, Sun Solaris)
  - Efficient approach to “special-cases”

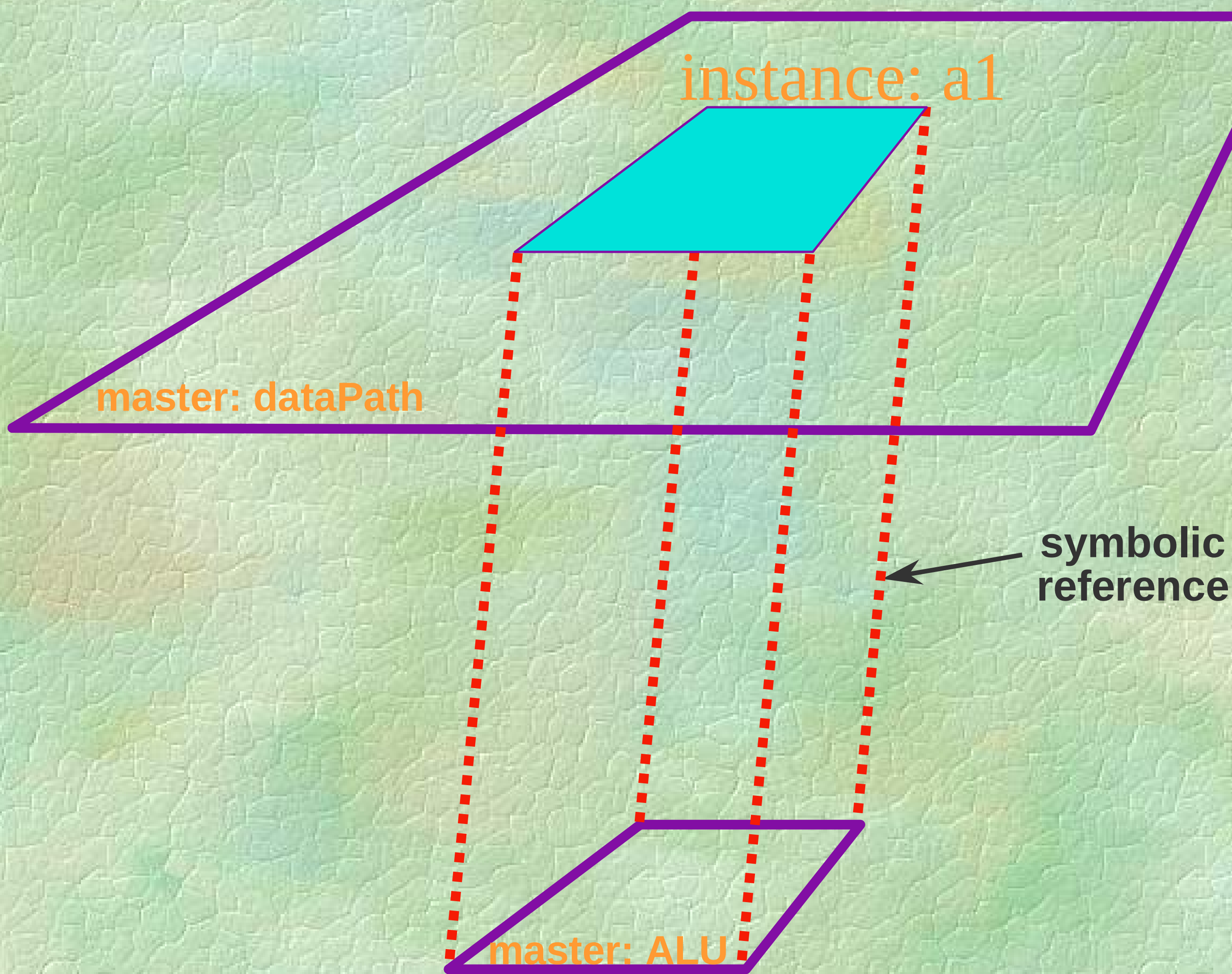


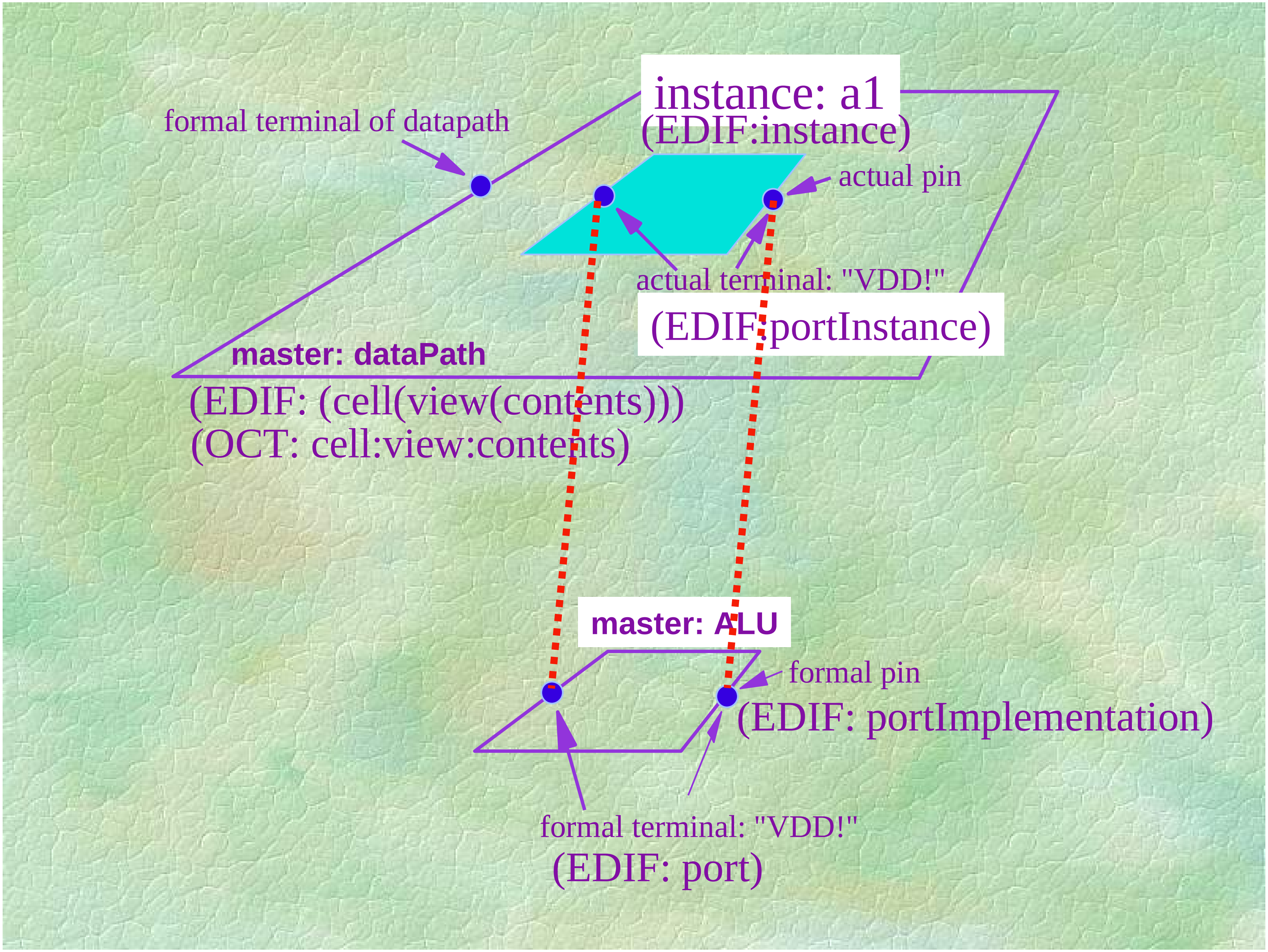
Important Distinctions

- Between components (abstractions) and their instances
- Like class and object, blueprint and product
- Has been a major issue since the introduction of entity-relationship diagrams

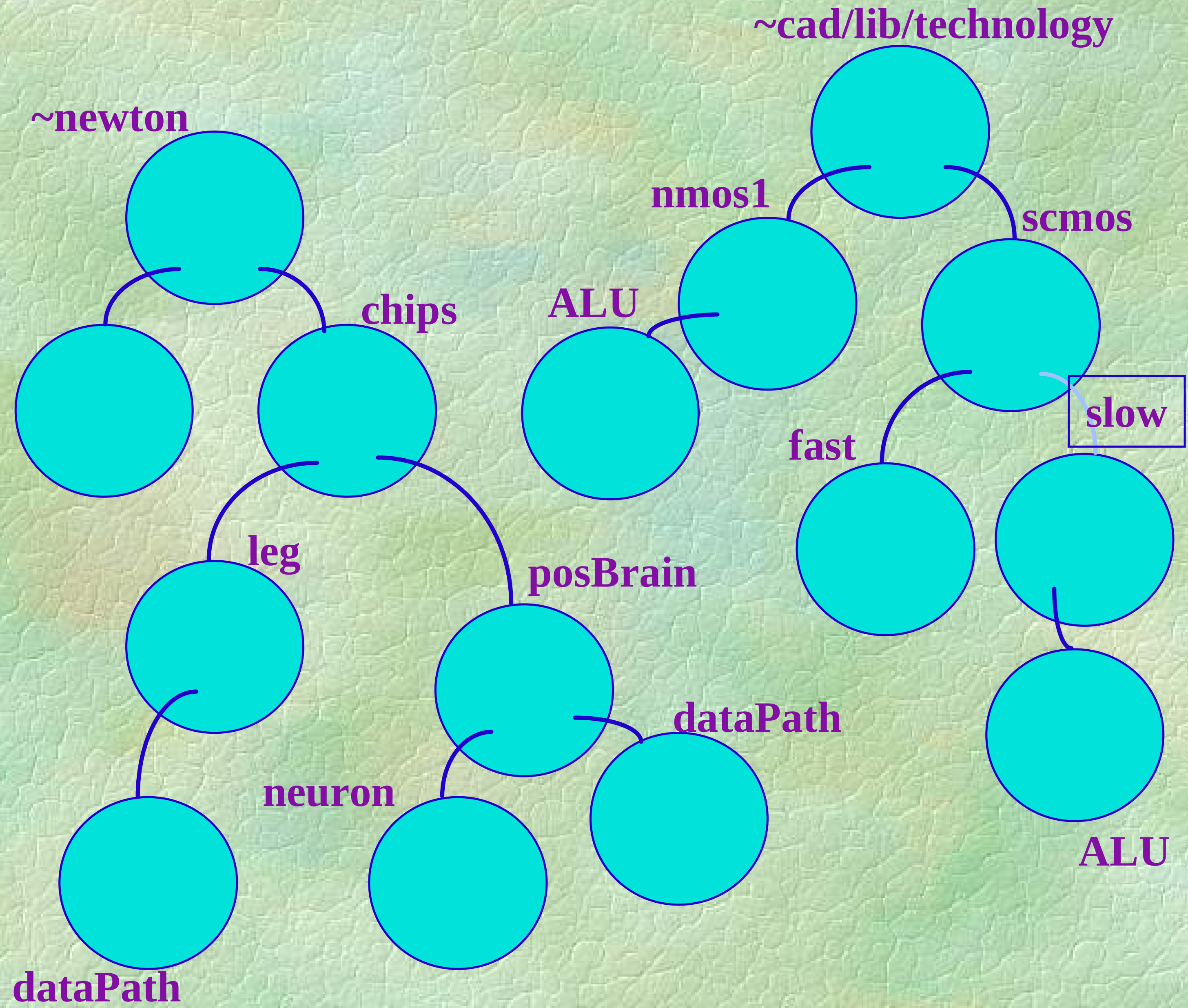
Entity-Relationship Approach to Data Representation

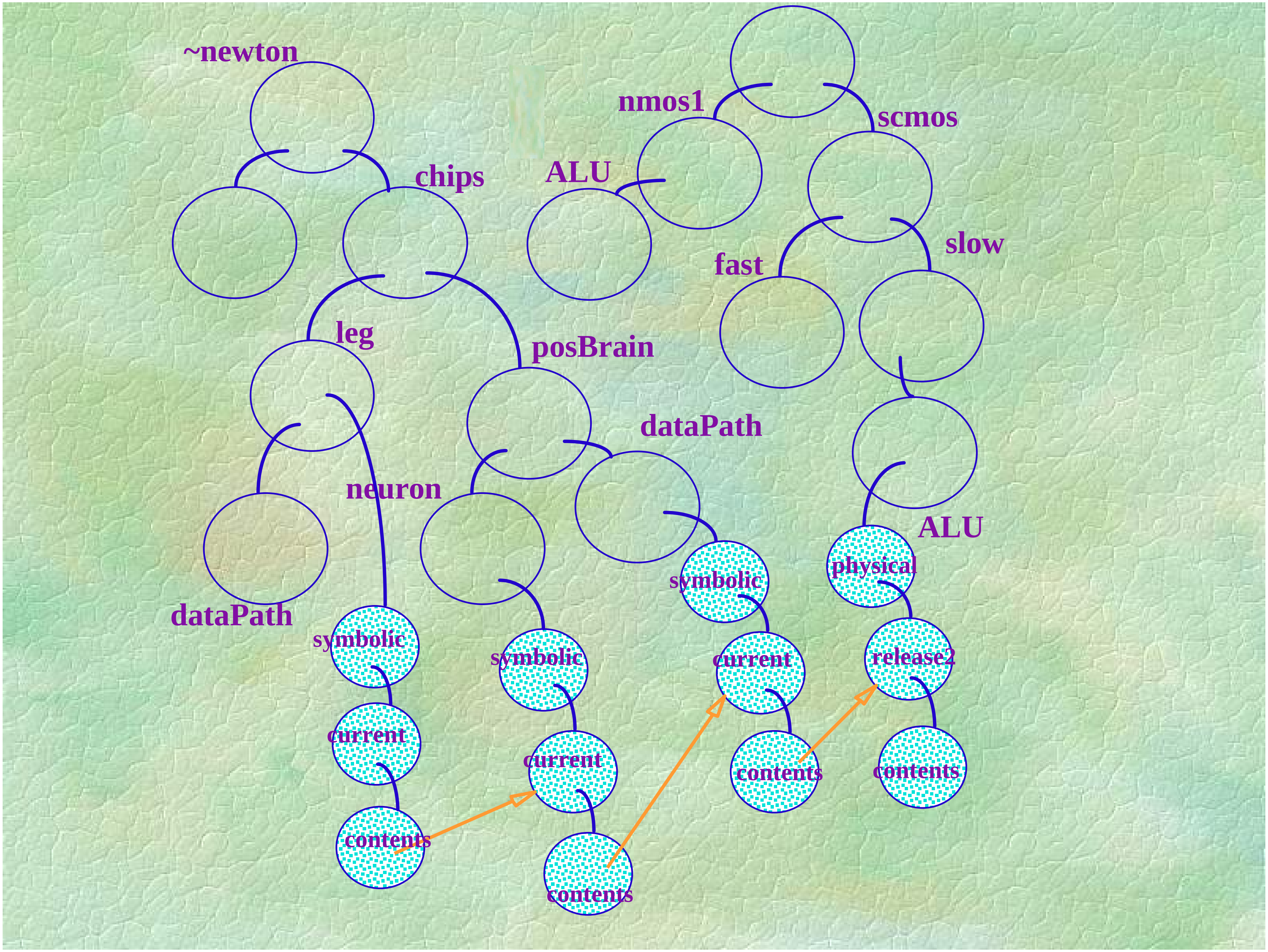


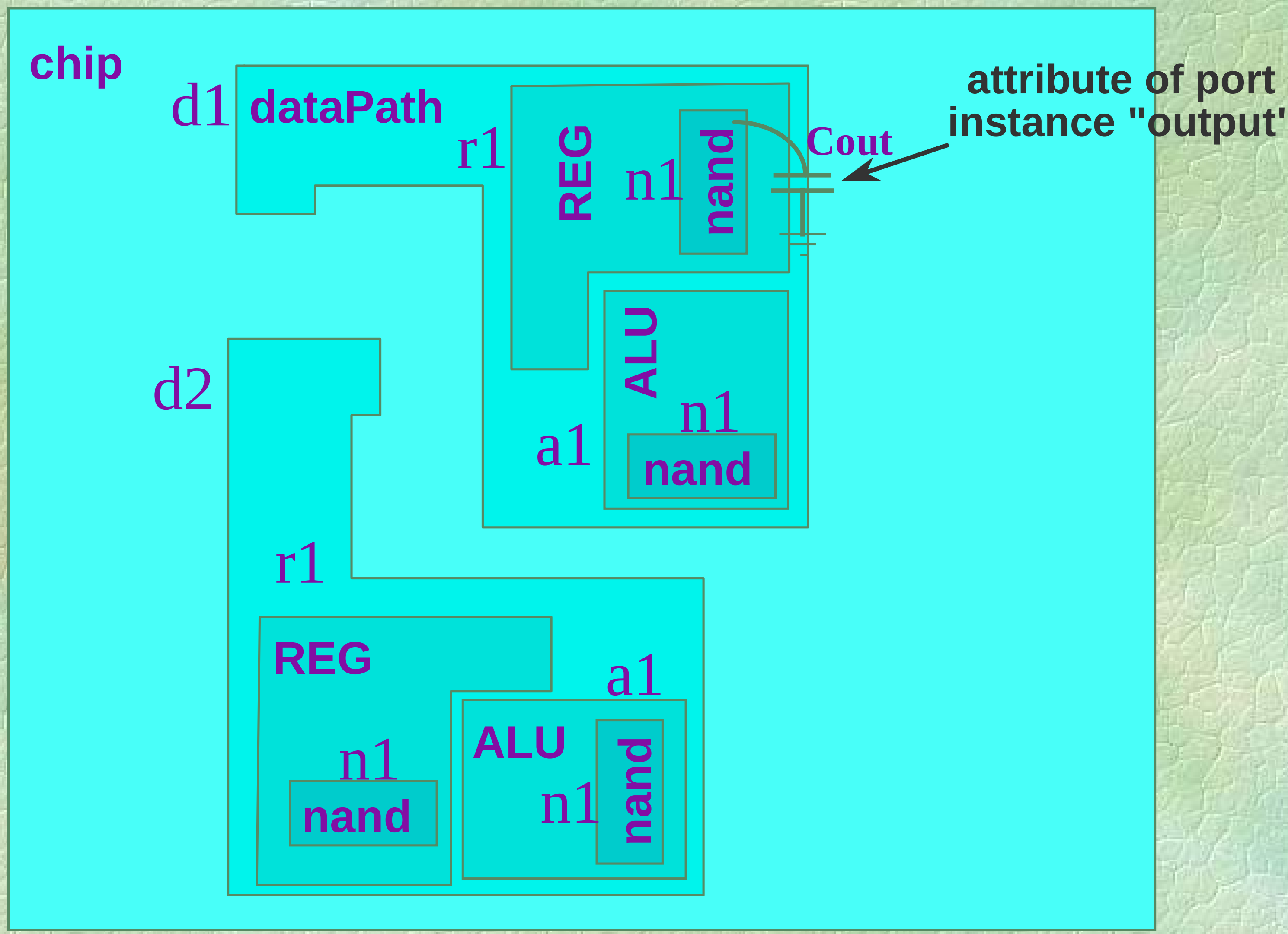

instance: a1
master: dataPath
symbolic
reference
master: ALU

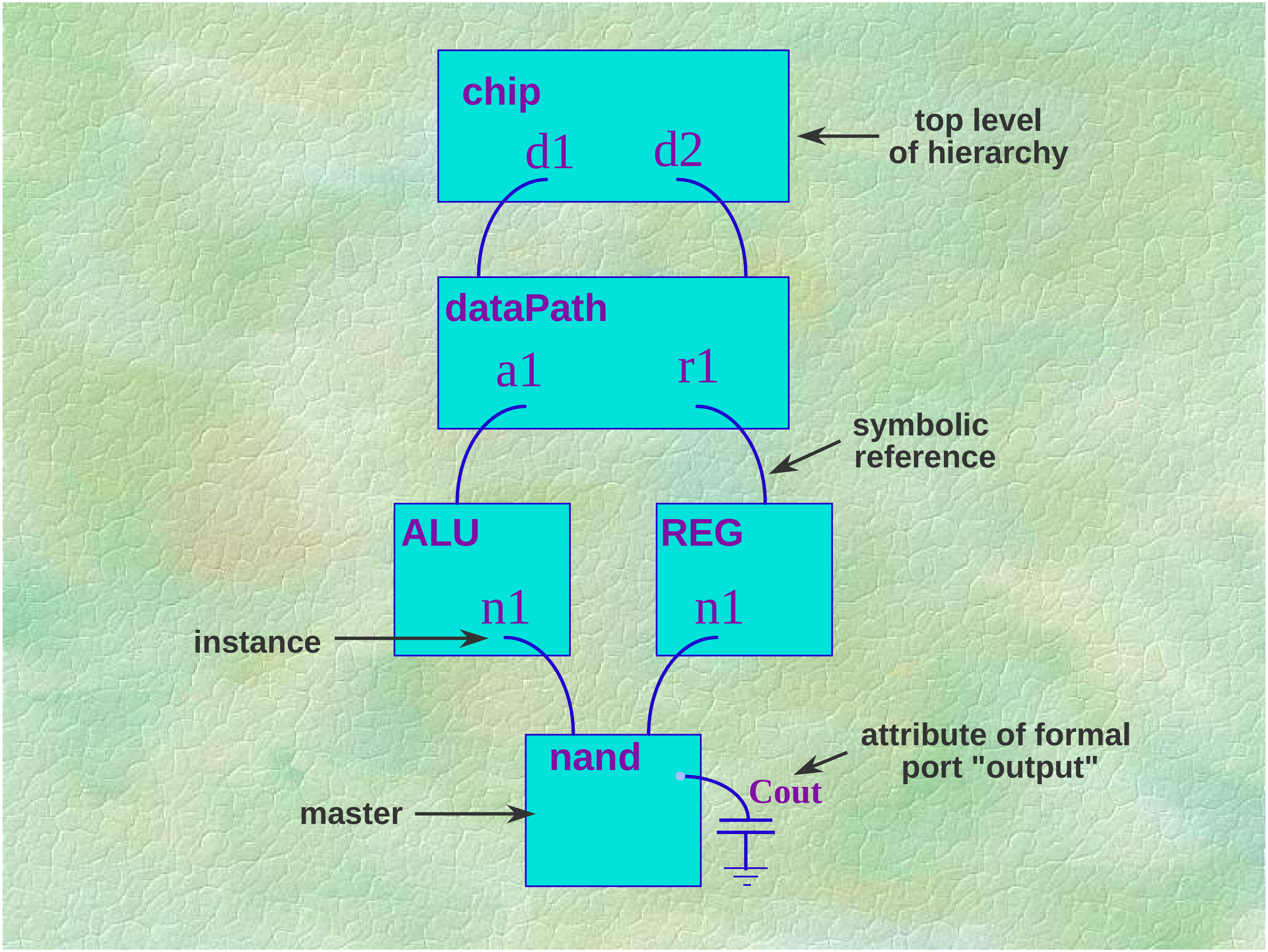


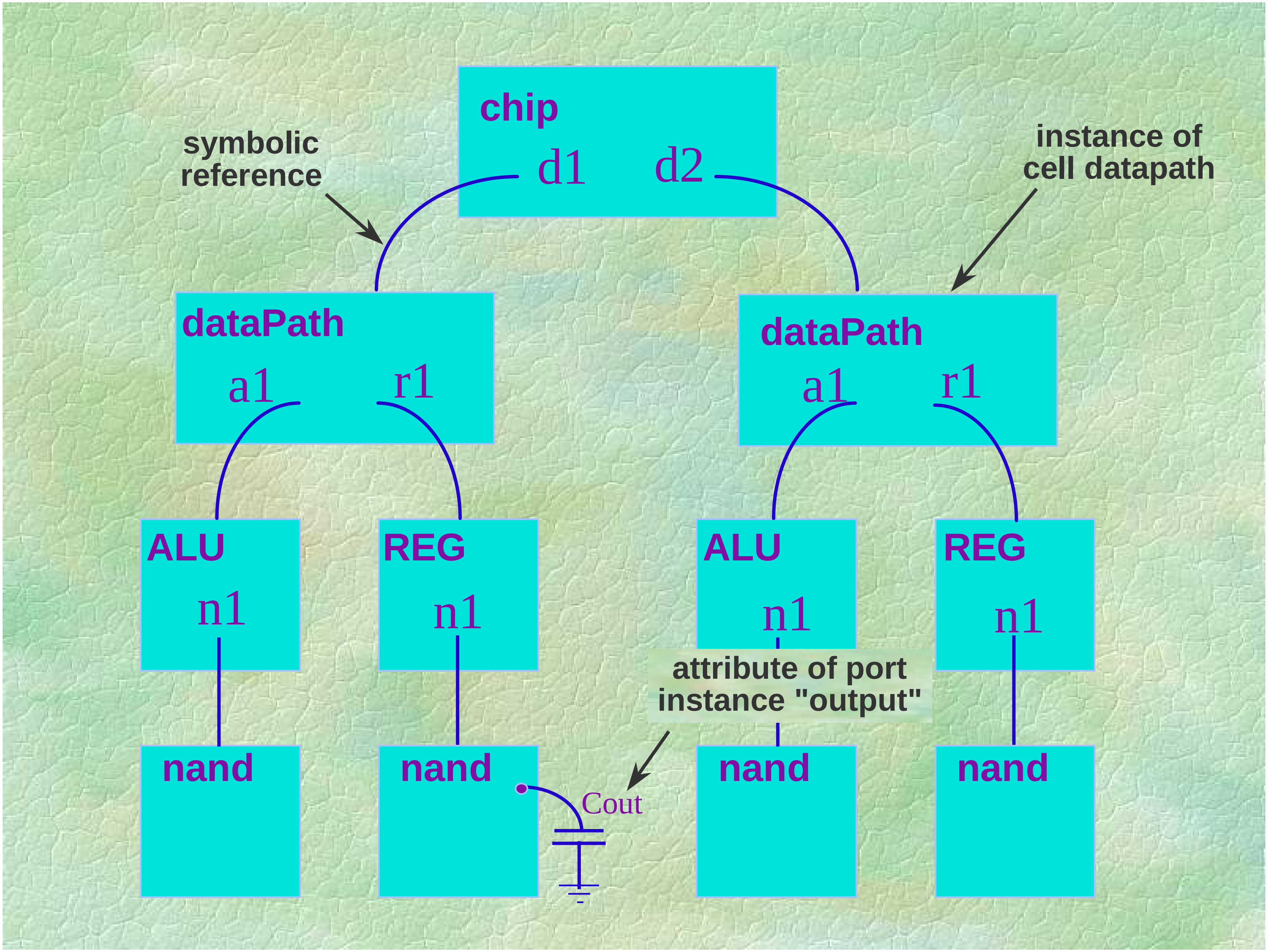
instance: a1
(EDIF:instance)
formal terminal of datapath
actual pin
actual terminal: "VDD!"
(EDIF:portInstance)
master: dataPath
(EDIF: (cell(view(contents)))
(OCT: cell:view:contents)
master: ALU
formal pin
(EDIF: portImplementation)
formal terminal: "VDD!"
(EDIF: port)


~cad/lib/technology
~newton
nmos1
scmos
chips
ALU
slow
fast
leg
posBrain
dataPath
neuron
ALU
dataPath


~newton
chips
leg
posBrain
dataPath
neuron
dataPath
symbolic
current
contents
symbolic
current
contents
nmos1
ALU
scmos
fast
slow
ALU
symbolic
current
contents
physical
release2
contents


chip
d1
dataPath
r1
REG
n1
nand
Cout
attribute of port
instance "output"
ALU
n1
a1
nand
d2
r1
REG
a1
n1
nand
ALU
n1
nand


chip
d1
d2
top level
of hierarchy
dataPath
a1
r1
symbolic
reference
ALU
n1
REG
n1
instance
nand
attribute of formal
port "output"
Cout
master


chip
d1
d2
symbolic
reference
instance of
cell datapath
dataPath
a1
r1
dataPath
a1
r1
ALU
n1
REG
n1
ALU
n1
REG
n1
attribute of port
instance "output"
nand
nand
Cout
nand
nand

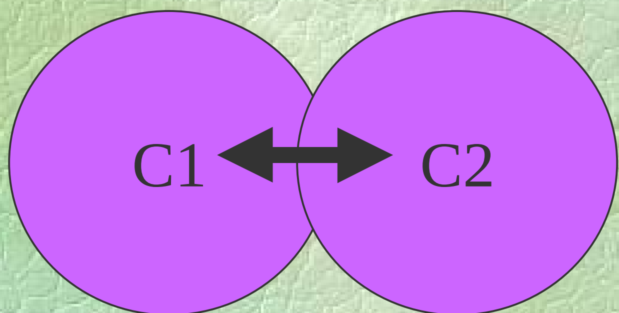
Objects versus Components

"Object orientation has failed but component software is succeeding"
(Udell, 1994)

- Definition of objects is purely technical
  - Encapsulation of state and behavior, polymorphism, inheritance
  - Does not include notions of independence or late composition (although they can be added…)
- Object markets did not happen
  - Like the FPGA market-- vendors give the tools away to sell a companion product (e.g. MFC)
- In OO, construction and assembly share a common base
  - Development is very technical, assembly is very technical
  - In CO, construction is technical, but assembly must be open to a wider user base
- Objects are rarely shaped to support "plug-and-play"
- Typically a component has to have sufficiently many uses, and therefore clients, for it to be viable

Component-Based Design

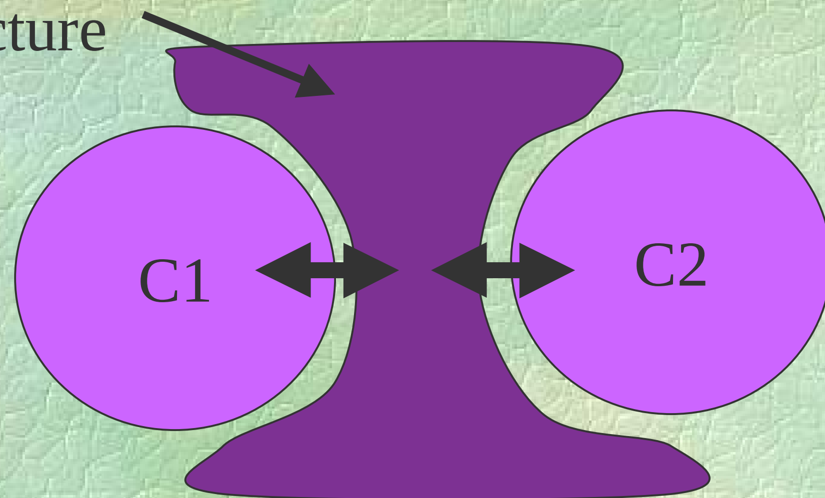
Today



"pass a pointer"
implemented on a stack

The component infrastructure

Tomorrow



Reliable, robust,
adaptable, and 'efficient':
"the operating system"

Component-Based Design

- In software languages:
  - Assume we are all on the same “team”
  - Optimize for efficiency, follow-up with debugging to fix problems (fragile interfaces)
  - Doesn’t scale well! (e.g. the Web)
- In communication protocols (e.g. TCP/IP)
  - Assume the guy at the other end is brain-dead
  - Assume whatever can go wrong will (links break, etc.)
  - Results in an “architecture” (e.g. packet-based) that is robust under the assumptions

Component-Based Design

- What is the “TCP/IP of component assembly”?
  - In the early days of TCP/IP we needed an IMP to implement the protocol, today it runs in s/w on a laptop
  - Must be reliable, robust, adaptable (“learn”, self-optimizing, self-balancing, negotiate for resources…)
  - Self-verifying (what does that mean?)
  - Self-testing
  - “Queriable”
- In many ways, it’s the “OS” of a component-oriented world
- Components might be collections of transistor, chunks of software (objects), applications, operating systems, NOW clusters, etc.

Next-Generation Operating Environments

- Advances in hardware and networking will enable an entirely new kind of operating system, which will raise the level of abstraction significantly for users and developers.
- Such systems will enforce extreme location transparency
  - Any code fragment runs anywhere
  - Any data object might live anywhere
  - System manages locality, replication, and migration of computation and data
- Self-configuring, self-monitoring, self-tuning, scaleable and secure

Adapted from Microsoft "Millenium" White Paper
http://www.research.microsoft.com

Next-Generation Operating Environments

- Seamless Distribution: System decides where computation should execute or data should reside, moving them there dynamically
- Worldwide Scalability:Logically there should only be one system, although at any one time it might be partitioned into many pieces.
- Fault-Tolerance: Transparently handle failures or removal of machines, network links, etc.

Next-Generation Operating Environments

- Self-Tuning: System should be able to reason about its computations and resources, allocating, replicating, and migrating computation and data to optimize performance, resource usage, and fault tolerance.
- Self-Configuring: New machines, network links, and resources should be automatically assimilated.
- Security: Allow non-hierarchical trust domains.
- Resource Controls: Both providers and consumers may explicitly manage the use of resources belonging to different trust domains.

Next-Generation Operating Environments

- No Storage Hierarchy: Once information is created, it should be accessible until it is no longer needed or referenced.
- Introspection: The system should posses some aspects of introspection and reflection.
  - Pervasively self-monitoring
  - Reason about its own configuration and performance
  - Suggest improvements
- Just-in-Time Binding: Sort of like the Internet today, but extended to all object interactions. “Binding-by-Search”
- Tools Emphasis Shifting: From code-efficiency to rapid application development with wizards automatically generating scaffolding or framework code.

"WebOS"

- The goal is to provide a common set of OS services to wide area applications, including mechanisms for:
  - Resource discovery
  - A global namespace
  - Remote process execution
  - Resource management
  - Authentication
  - Security
- Provide services needed to build applications that are:
  - Geographically distributed
  - Highly available
  - Incrementally scalable
  - Dynamically reconfiguring

Interfaces and Standards

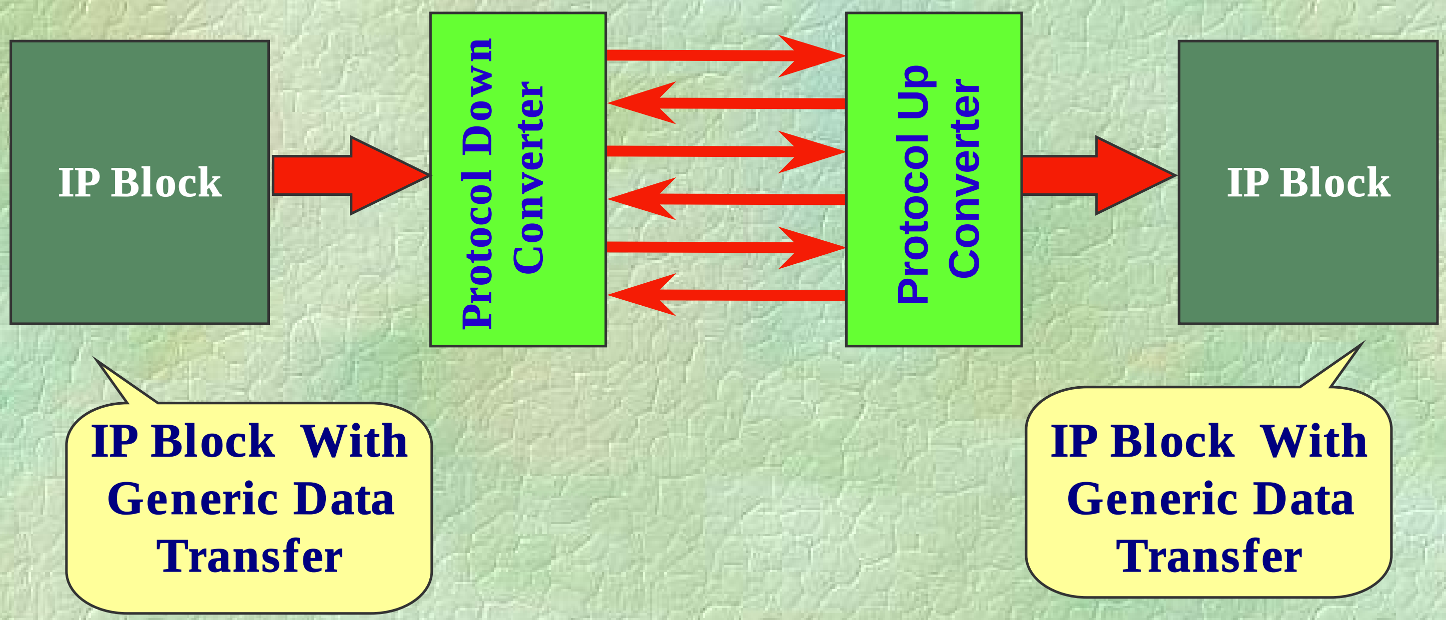
- “A component needs to hold a significant portion of a market specific to its domain”
  - Generally drives (quasi) standards
- A standard should specific *just* as much about interfacing of certain components as is needed to allow sufficiently many clients and vendors to work together (including acceptable deviations and “tolerances”)
- Wiring standards are not enough
  - People can find ways around wiring as needed: adaptors

Some Potential Key Technologies

- What software technology, or technologies, will play the central role in enabling such a distributed component architecture?
- Java and *JavaBeans*
- CORBA
- Microsoft COM (COM, DCOM, COM+)
- Jini

Communication Refinement

- Separate *Function* of blocks from inter-block *Communication*
- Substitute lower-level detail for communications behavior



Source: Prof. Alberto Sangiovanni

Communication Refinement

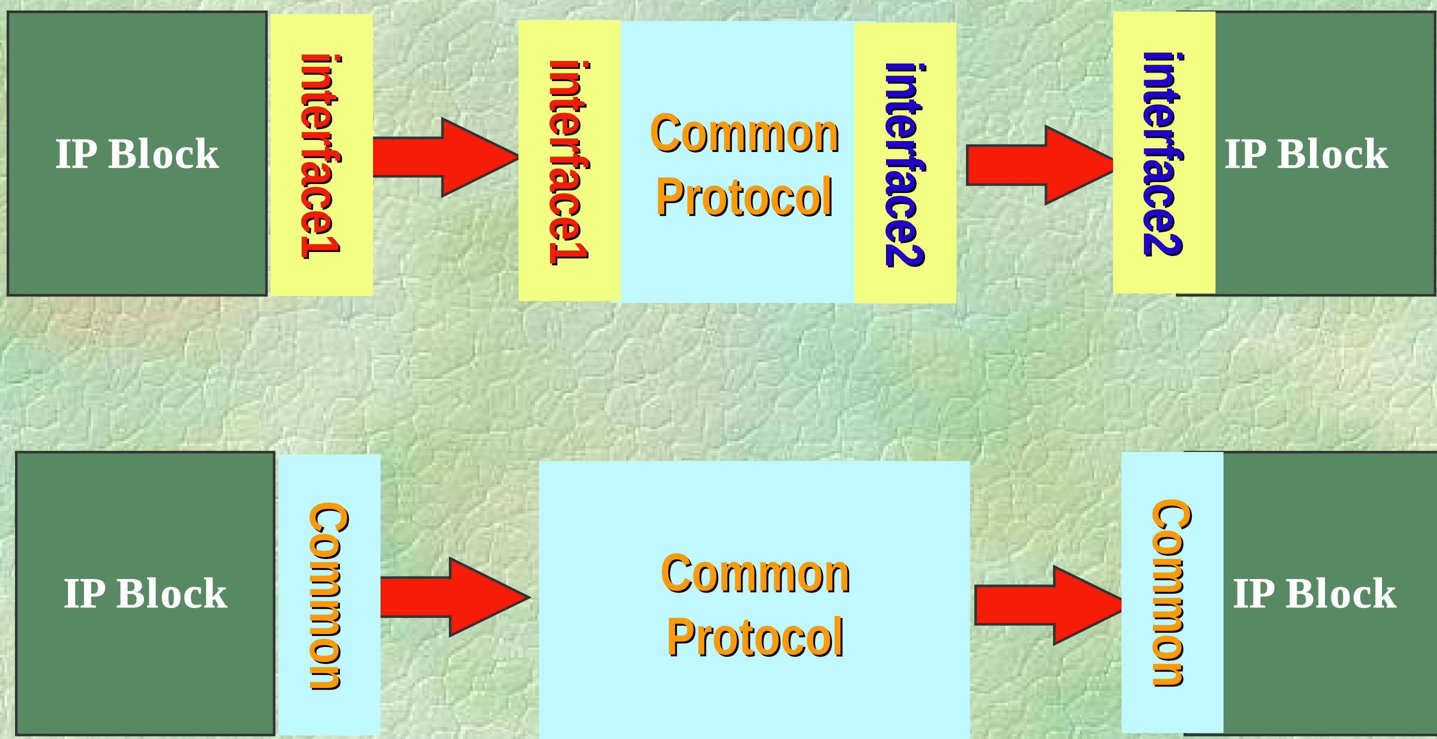
- Issue: Where do we cut? Where are the “standards”?
- Where is the communication burden placed?
- Applies to both hardware and software

Microsoft COM Analogy (Component Object Model)

- Binary and network (DCOM) standard that allows two objects to communicate, regardless of what machine they are running on.
- Can be used from C++, C, VB, Java, Delphi, …
- Supports three types of objects: In-process (DLL), local (EXE), and remote (DLL or EXE)

Communication Refinement

- Issue: Where do we cut? Where are the "standards"?
- Where is the communication burden placed?
- Applies to both hardware and software



Java/JavaBeans Analogy

- *JavaBeans* is a portable, platform-independent component model written in Java.
- It enables developers to write reusable components once and run them anywhere -- benefiting from the platform-independent power of Java.
- *JavaBeans* acts as a bridge between proprietary component models and provides a seamless means for developers to build components that run in ActiveX container applications.

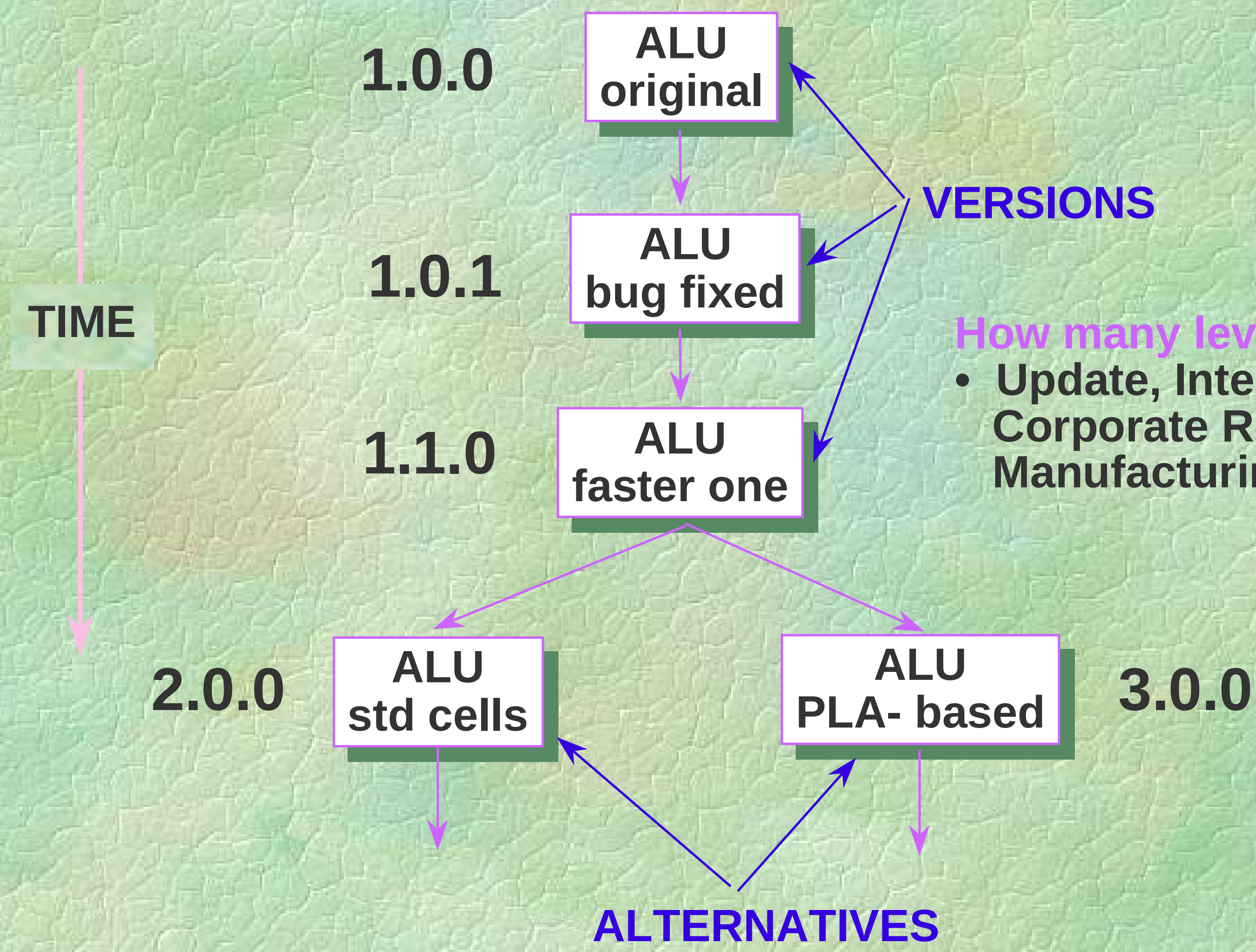
Attributes of *JavaBeans*

- Introspection: enables a builder tool to analyze how a Bean works
- Customization: enables a developer to use an app builder tool to customize the appearance and behavior of a Bean
- Events: enables Beans to communicate and connect together
- Properties: enable developers to customize and program with Beans
- Persistence: enables developers to customize Beans in an app builder, and then retrieve those Beans, with customized features intact, for future use

CORBA
(Common Object Request Broker Architecture)

- A standard for distributed objects being developed by the Object Management Group (OMG).
- CORBA provides the mechanisms by which objects transparently make requests and receive responses, as defined by OMG's ORB.
- The CORBA ORB is an application framework that provides interoperability between objects, built in (possibly) different languages, running on (possibly) different machines in heterogeneous distributed environments.

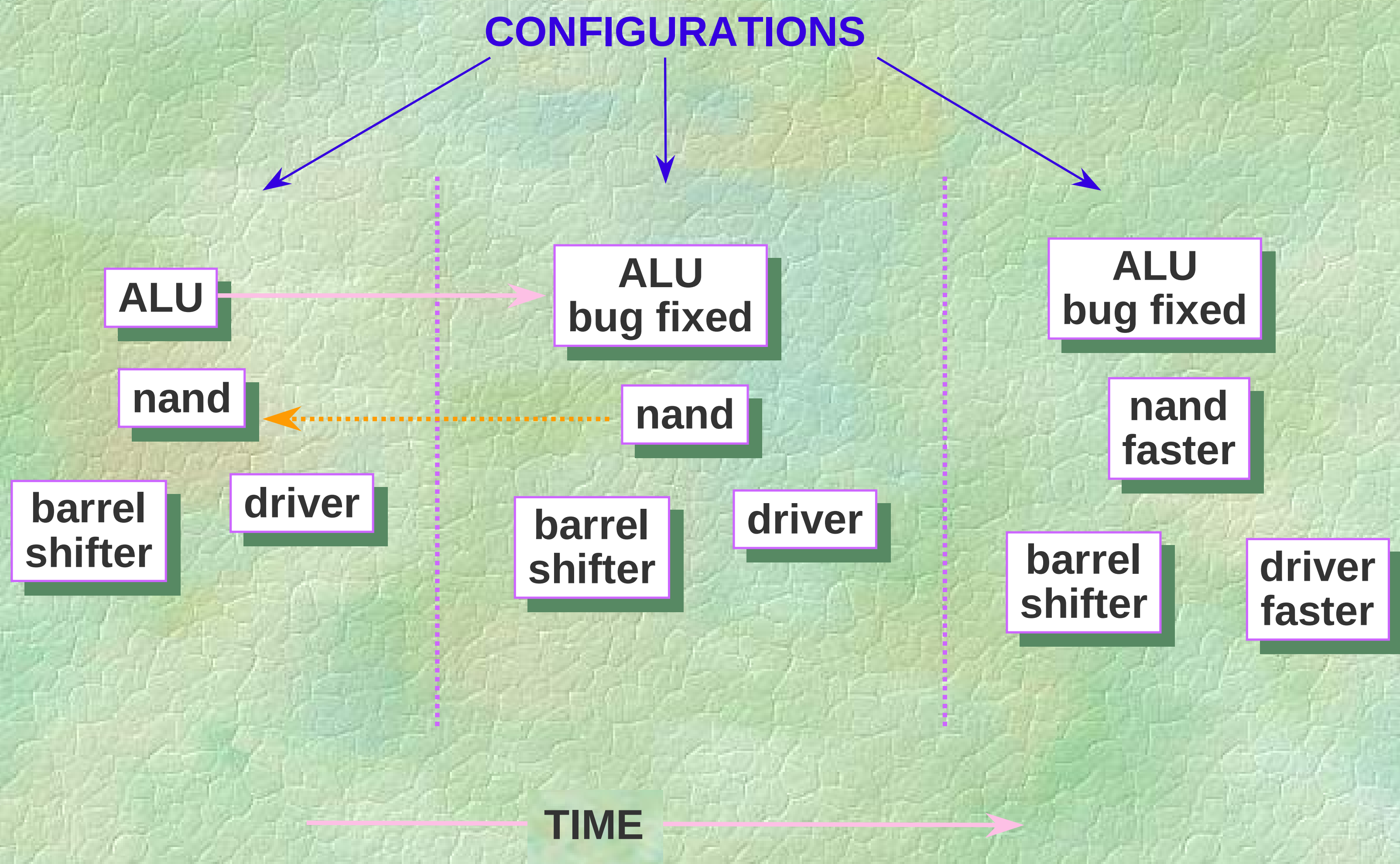
Versions, Alternatives, and Configurations



How many levels?

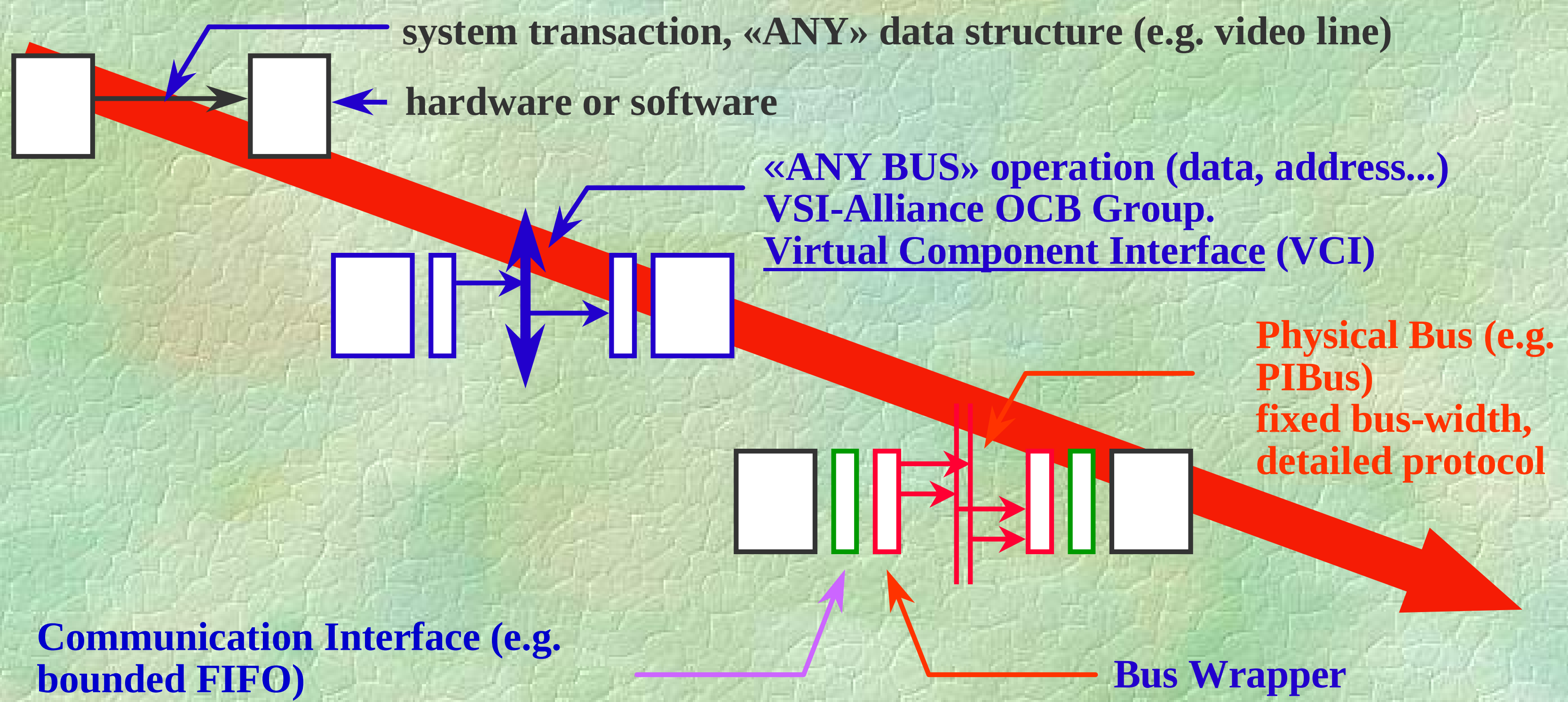
- Update, Internal Release, Corporate Release, Manufacturing Release, ...

Versions, Alternatives, and Configurations



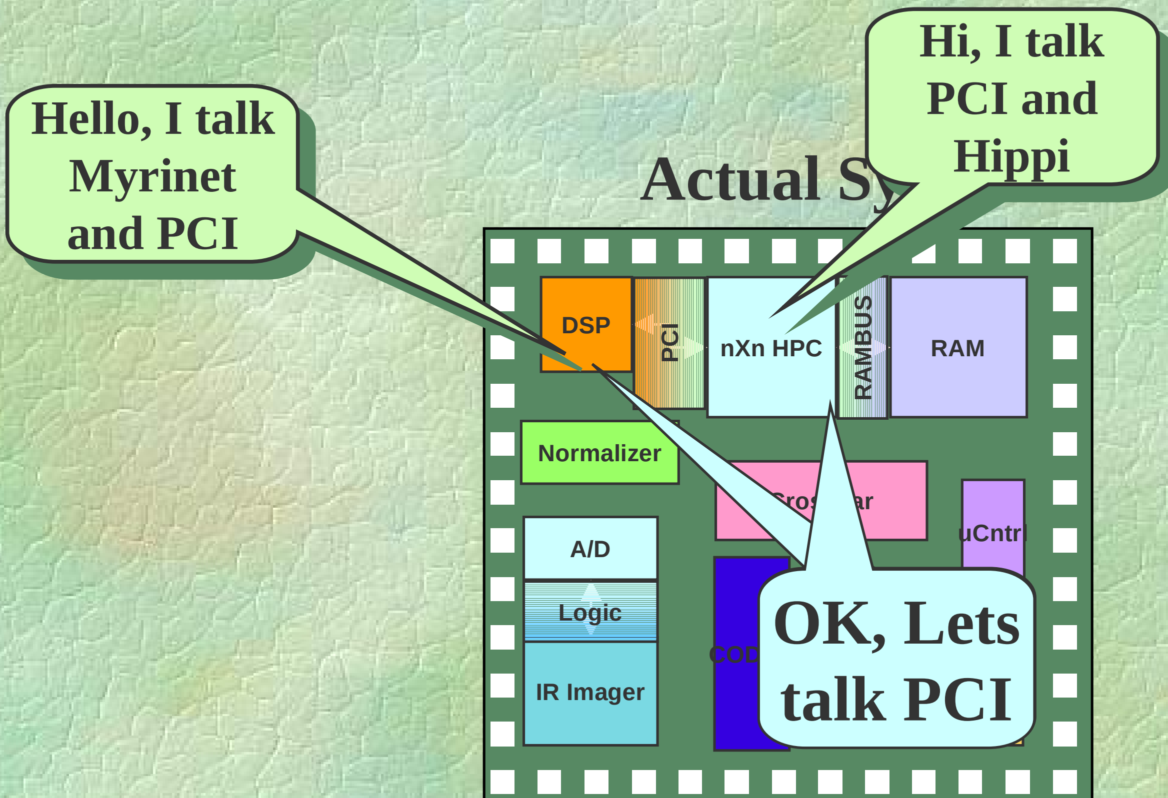
Communication Refinement

Standard interfaces constitute the backbone of an IP market: abstract form the concerns of hardware implementation (multi-target VC), abstract from the concerns of a particular bus (bus-independent VC)



Source: Prof. Alberto Sangiovanni

Automated Interface Synthesis



Source: DARPA ISAT *Silicon 2010* Study, 1997 (Randy Harr, Synopsys)