Software Architecture

Lecture 1

Learning Objectives

- Define software architecture and its elements
- Distinguish between accidental and essential difficulties in software and enumerate them
- Compare and contrast software architecture to building architecture and understand the role of the architect
- Identify essential elements of the architecture of the WWW and the UNIX pipes and filters paradigm
- Understand software product lines and the role of reuse in software design

What is Software Architecture?

- It's all about software design
  - Architecture is software design, but not all design is software architecture
    - part of the design process

- [Gorton] Architecture focuses on 'issues that will be difficult/impossible to change once the system is built'
  - E.g., quality attributes like security, performance
  - E.g., non-functional requirements like cost, deployment hardware

Definitions – Taylor et al.

- *[Software architecture is] the set of principal design decisions governing a system*

Definitions - ANSI/IEEE Std 1471-2000

- *"Architecture is the **fundamental organization** of a system, embodied in its **components**, their **relationships** to **each other** and the **environment**, and the principles governing its **design and evolution**."*

Definitions - SEI

- *"The software architecture of a program or computing system is the **structure or structures** of the system, which comprise **software elements**, the **externally visible properties** of those elements, and the **relationships** among them."*

Main point:
Architecture Defines Structure

- Decomposition of system into components/modules/ subsystems

- Architecture defines:
  - Component interfaces
    - What a component can do
  - Component communications and dependencies
    - How components communicate
  - Component responsibilities
    - Precisely what a component will do

Learning Objectives

- Define software architecture and its elements
- Distinguish between accidental and essential difficulties in software and enumerate them
- Compare and contrast software architecture to building architecture and understand the role of the architect
- Identify essential elements of the architecture of the WWW and the UNIX pipes and filters paradigm
- Understand software product lines and the role of reuse in software design

Software Engineering Difficulties

- Software engineers deal with unique set of problems
  - Young field with tremendous expectations
  - Building of vastly complex, but intangible systems
  - Often software is not useful on its own
    - (e.g., unlike a car)
    - thus it must conform to changes in other engineering areas
- Some problems can be eliminated
  - “accidental difficulties”
- Other problems can be lessened, but not eliminated
  - “essential difficulties”

Accidental Difficulties

- Solutions exist
  - Possibly waiting to be discovered

- Past productivity increases were the result of overcoming
  - Inadequate programming constructs & abstractions
    - Remedied by high-level programming languages
    - Increased productivity by factor of five
    - Complexity was never inherent in program at all

Accidental Difficulties (cont'd)

- Past productivity increases were the result of overcoming (cont'd)
  - Inadequate tools:
    - E.g., Viewing results of programming decisions took long time
      - Remedied by time–sharing
      - Turnaround time approaching limit of human perception
  - Difficulty of using heterogeneous programs
    - Addressed by integrated software development environments
    - Support task that was conceptually always possible

Essential Difficulties

- Only partial solutions exist for them, if any
- Cannot be abstracted away

  - Complexity
  - Conformity
  - Changeability
  - Intangibility

Complexity

- No two software parts are alike
  - If they are, they are abstracted away into one
- <u>Complexity grows super-linearly with size</u>
  - E.g., it is impossible to enumerate all states of program
    - Except perhaps “toy” programs

Conformity

- Software is required to conform to its
  - Operating environment
  - Hardware
- Often “last kid on block”
- Perceived as most conformable

Changeability

- Software is viewed as infinitely malleable
- Change originates with
  - New applications, users, machines, standards, laws
  - Hardware problems

Intangibility

- Software is not embedded in space
  - Often no constraining physical laws
- No obvious representation
  - E.g., familiar geometric shapes

Promising Attacks On Complexity

- Buy vs. Build
- Requirements refinement & rapid prototyping
  - Hardest part is deciding what to build (or buy?)
  - Must show product to customer to get complete spec.
  - Need for iterative feedback

Promising Attacks On Complexity (cont’d)

- Incremental/Evolutionary/Spiral Development
  - Grow systems, don’t build them
  - Good for morale
  - Easy backtracking
  - Early prototypes
- Great designers
  - Good design can be taught; great design cannot
  - Nurture great designers

Primacy of Design

- Software engineers collect requirements, code, test, integrate, configure, etc.
- An architecture-centric approach to software engineering places an emphasis on design
  - Design pervades the engineering activity from the very beginning
- But how do we go about the task of architectural design?

Analogy: Architecture of Buildings

- We all live in them
- (We think) We know how they are built
    - Requirements
    - Design (blueprints)
    - Construction
    - Use
- This is similar (though not identical) to how we build software

Some Obvious Parallels

- Satisfaction of customers’ needs
- Specialization of labor
- Multiple perspectives of the final product
- Intermediate points where plans and progress are reviewed

Deeper Parallels

- Architecture is different from, but linked with the product/structure
- Properties of structures are induced by the design of the architecture
- The architect has a distinctive role

Deeper Parallels (cont'd)

- Process is not as important as architecture
    - Design and resulting qualities are at the forefront
    - Process is a means, not an end
- Architecture has matured over time into a discipline
    - Architectural styles as sets of constraints
    - Styles also as wide range of solutions, techniques and palettes of compatible materials, colors, and sizes

More about the Architect

- A distinctive role in a project
- Very broad training
- Amasses and leverages extensive experience
- A keen sense of aesthetics
- Deep understanding of the domain
  - Properties of structures, materials, and environments
  - Needs of customers

More about the Architect (cont’d)

- Even first-rate programming skills are insufficient for the creation of complex software applications
  - But are they even necessary?

Limitations of the Analogy…

- We know a lot about buildings, much less about software
- The nature of software is different from that of building architecture
- Software is much more malleable than physical materials
- The two "construction industries" are very different
- Software deployment has no counterpart in building architecture
- Software is a machine; a building is not

…But Still Very Real Power of Architecture

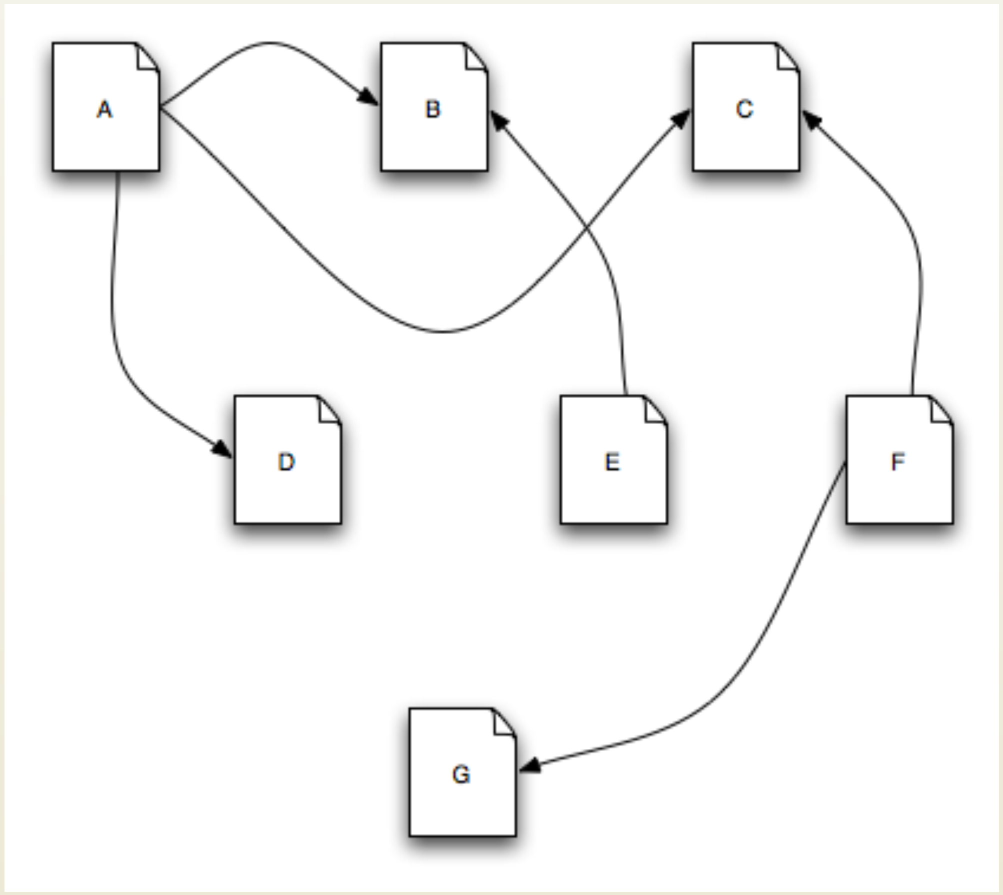
- Giving preeminence to architecture offers the potential for
  - Intellectual control
  - Conceptual integrity
  - Effective basis for knowledge reuse
  - Realizing experience, designs, and code
  - Effective project communication
  - Management of a set of variant systems
- Limited-term focus on architecture will not yield significant benefits!

Learning Objectives

- Define software architecture and its elements
- Distinguish between accidental and essential difficulties in software and enumerate them
- Compare and contrast software architecture to building architecture and understand the role of the architect
- Identify essential elements of the architecture of the WWW and the UNIX pipes and filters paradigm
- Understand software product lines and the role of reuse in software design

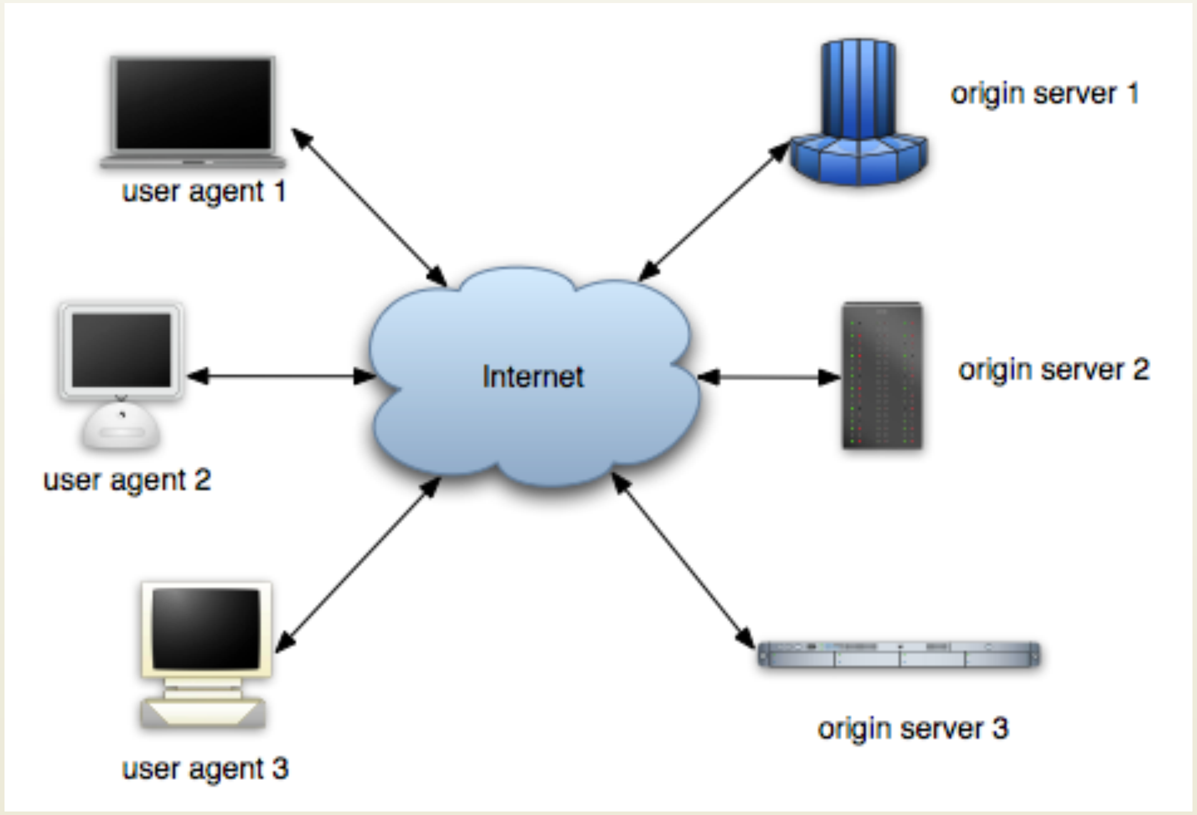
Architecture in Action: WWW

- This is the Web



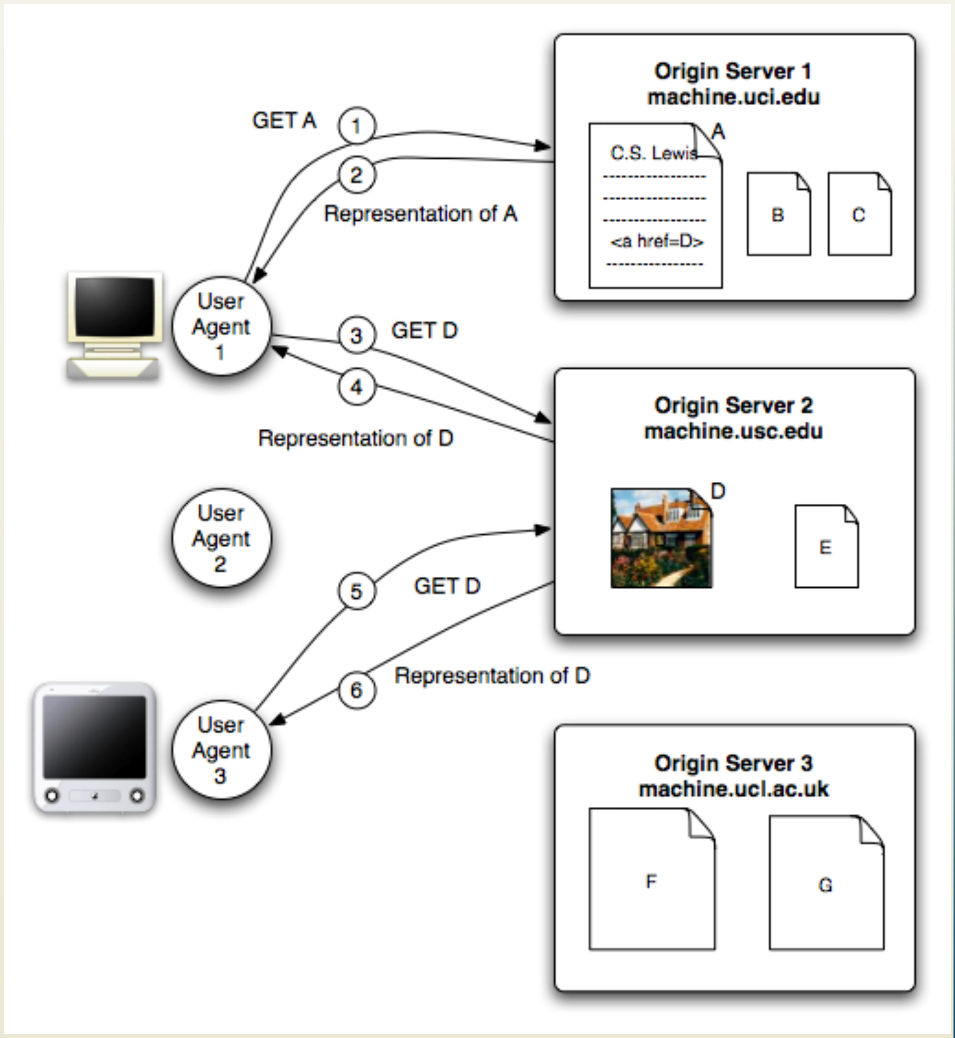
Architecture in Action: WWW

- So is this



Architecture in Action: WWW

- And this



WWW in a (Big) Nutshell

- The Web is a collection of resources, each of which has a unique name known as a “uniform resource locator” (URL)
- Each resource denotes, informally, some information.
- URI’s can be used to determine the identity of a machine on the Internet, known as an origin server, where the value of the resource may be ascertained.
- Communication is initiated by clients, known as user agents, who make requests to servers.
  - Web browsers are common instances of user agents.

WWW in a (Big) Nutshell (cont'd)

- Resources can be manipulated through their representations.
  - HTML is a very common representation language used on the Web.
- All communication between user agents and origin servers must be performed by a simple, generic protocol (HTTP), which offers the command methods GET, POST, etc.
- Communication between user agents and origin servers is fully self-contained.
  - (So-called "stateless interactions")

WWW's Architecture

- Architecture of the Web is wholly separate from the code
- There is no one piece of code that implements the architecture.
- There are multiple pieces of code that implement the various components of the architecture.
  - E.g., different Web browsers

WWW’s Architecture (cont’d)

- Stylistic constraints of the Web’s architectural style are not apparent in the code
  - The effects of the constraints are evident in the Web

- One of the world’s most successful applications is only understood adequately from an architectural vantage point.

Architecture in Action (2): @ small scale

- Unix command line

```
ls invoices | grep -e august | sort
```

- Application architecture can be understood based on very few rules
- Applications can be composed by non-programmers
    - Akin to Lego blocks
- A simple architectural concept that can be comprehended and applied by a broad audience

Learning Objectives

- Define software architecture and its elements
- Distinguish between accidental and essential difficulties in software and enumerate them
- Compare and contrast software architecture to building architecture and understand the role of the architect
- Identify essential elements of the architecture of the WWW and the UNIX pipes and filters paradigm
- Understand software product lines and the role of reuse in software design

Software Product Line

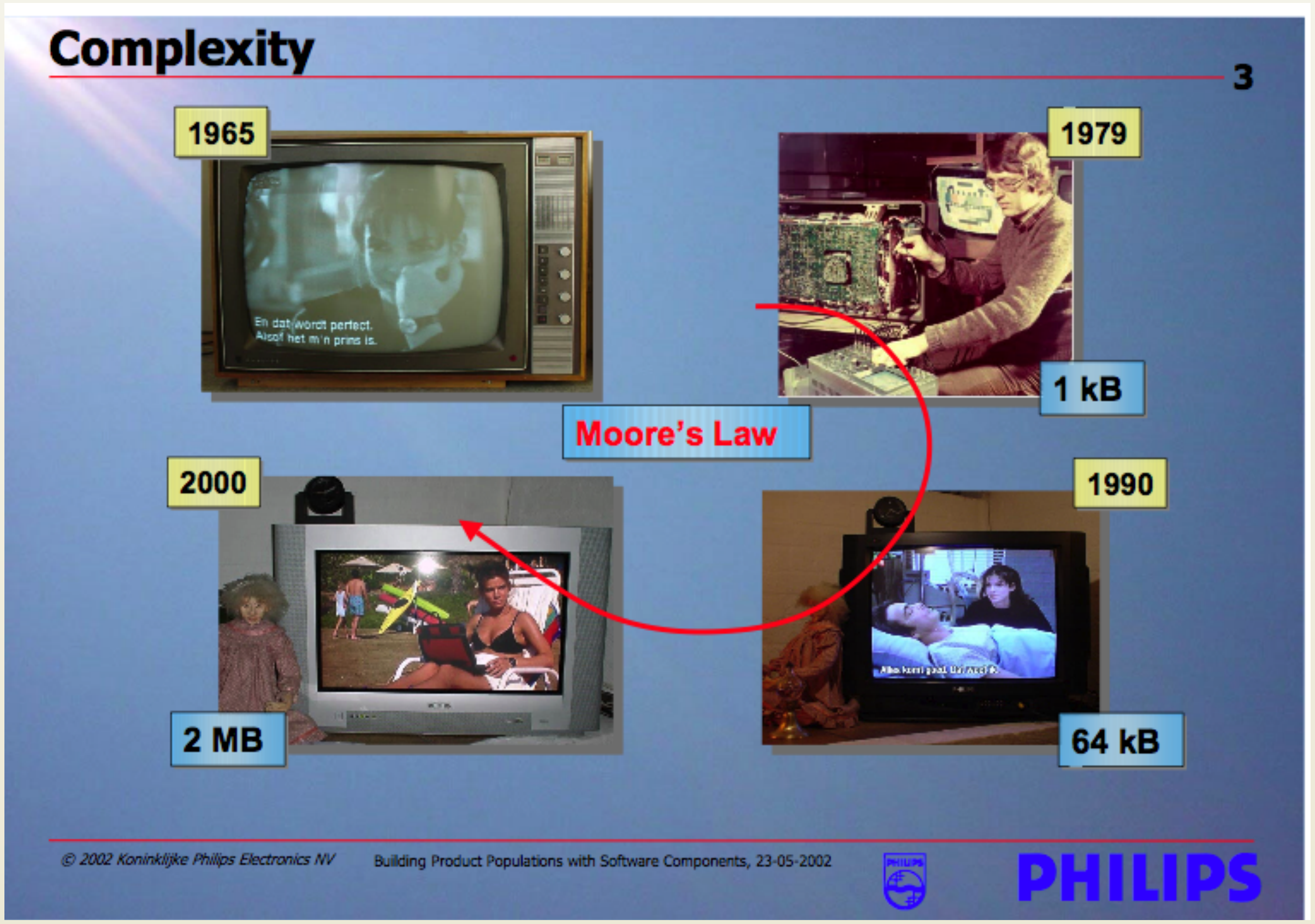
- Motivating example
  - A consumer is interested in a 35-inch HDTV with a built-in DVD player for the North American market.

    Such a device might contain upwards of a million lines of embedded software.

    This particular television/DVD player will be very similar to a 35-inch HDTV without the DVD player, and also to a 35-inch HDTV with a built-in DVD player for the European market, where the TV must be able to handle PAL or SECAM encoded broadcasts, rather than North America's NTSC format.

    These closely related televisions will similarly each have a million or more lines of code embedded within them.

Growing Sophistication of Consumer Devices



Complexity

3

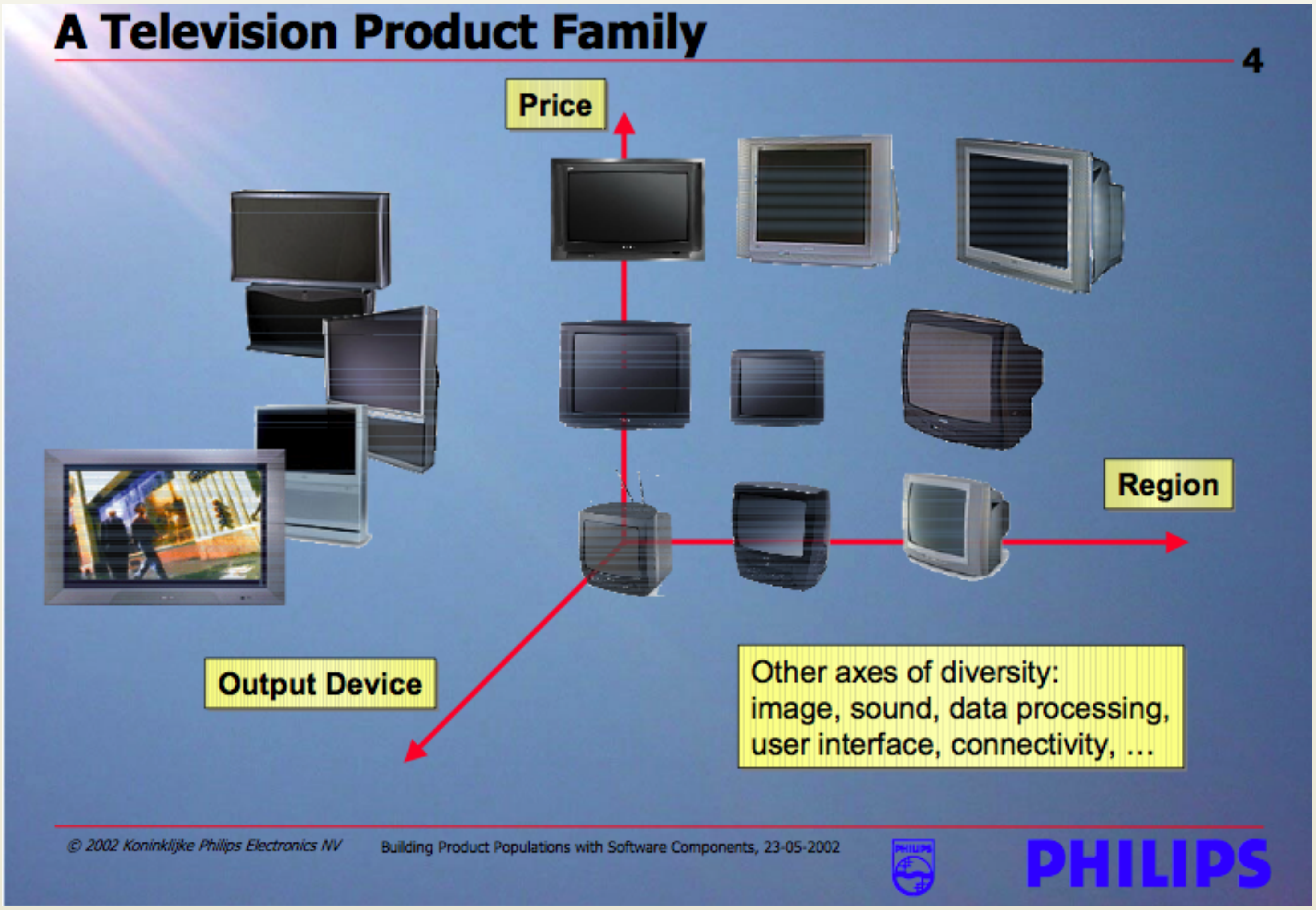
1965
En dat wordt perfect.
Alsof het m'n prins is.
1979
1 kB
Moore's Law
2000
1990
2 MB
64 kB

© 2002 Koninklijke Philips Electronics NV

Building Product Populations with Software Components, 23-05-2002

PHILIPS

Families of Related Products


A Television Product Family

4
Price
Region
Output Device

Other axes of diversity: image, sound, data processing, user interface, connectivity, …

© 2002 Koninklijke Philips Electronics NV
Building Product Populations with Software Components, 23-05-2002
PHILIPS

The Necessity and Benefit of PLs

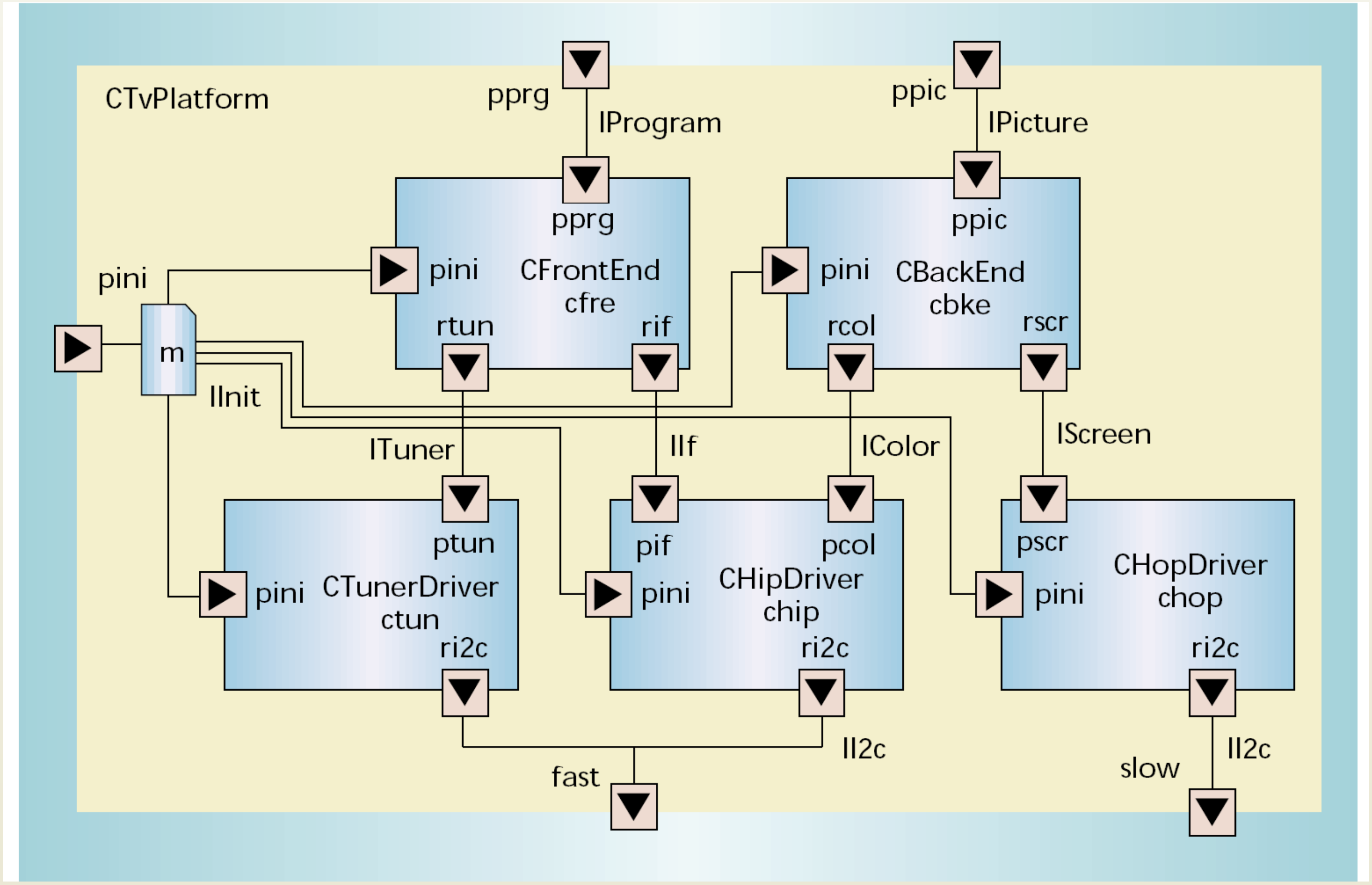
- Building each of these TVs from scratch would likely put Philips out of business
- Reusing structure, behaviors, and component implementations is increasingly important to successful business practice
  - It simplifies the software development task
  - It reduces the development time and cost
  - it improves the overall system reliability
- Recognizing and exploiting commonality and variability across products

Reuse as the Big Win

- Architecture: reuse of
  - Ideas
  - Knowledge
  - Patterns
  - Engineering guidance
  - Well-worn experience

- Product families: reuse of
  - Structure
  - Behaviors
  - Implementations
  - Test suites…

The Centerpiece – Architecture



Learning Objectives

- Define software architecture and its elements
- Distinguish between accidental and essential difficulties in software and enumerate them
- Compare and contrast software architecture to building architecture and understand the role of the architect
- Identify essential elements of the architecture of the WWW and the UNIX pipes and filters paradigm
- Understand software product lines and the role of reuse in software design