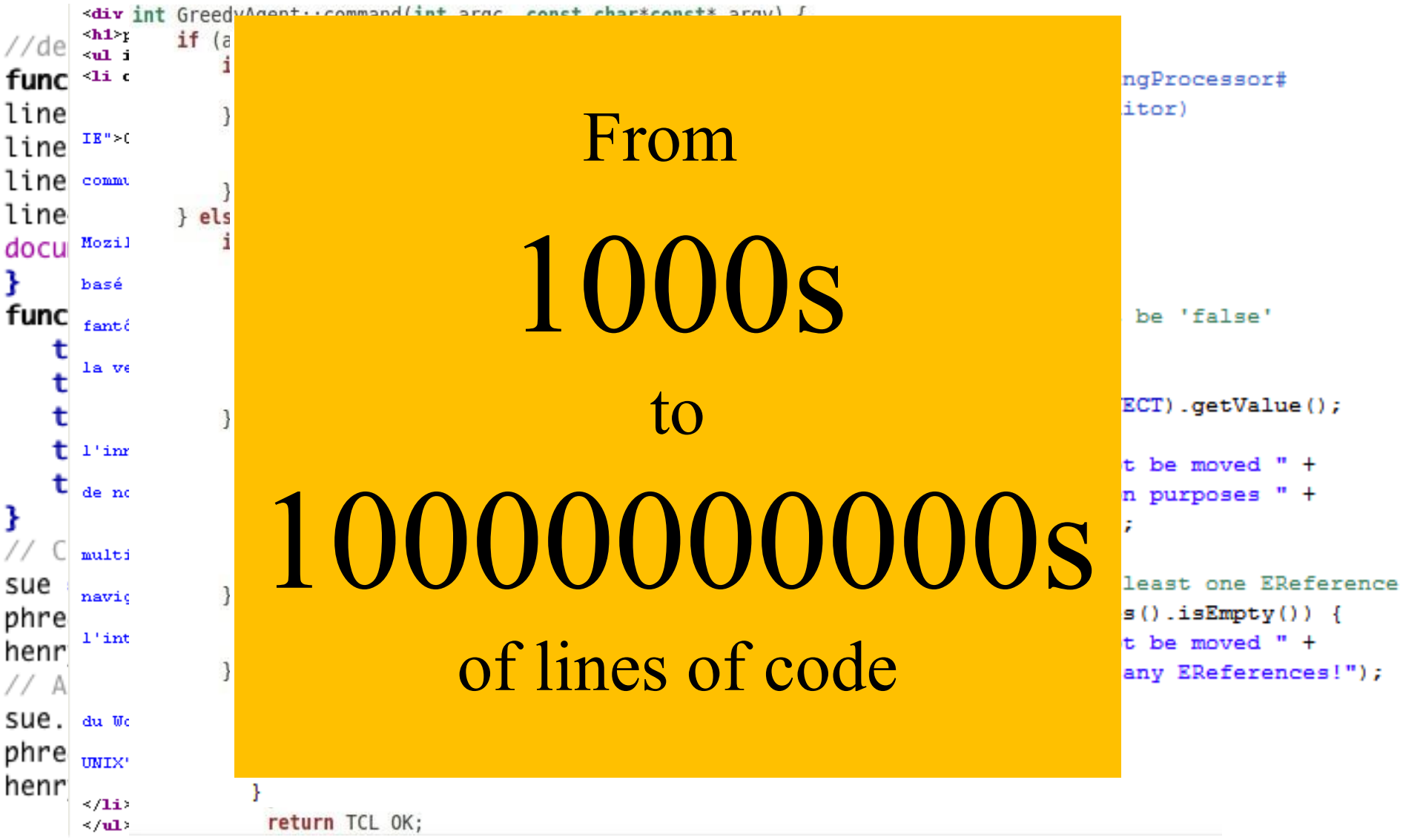
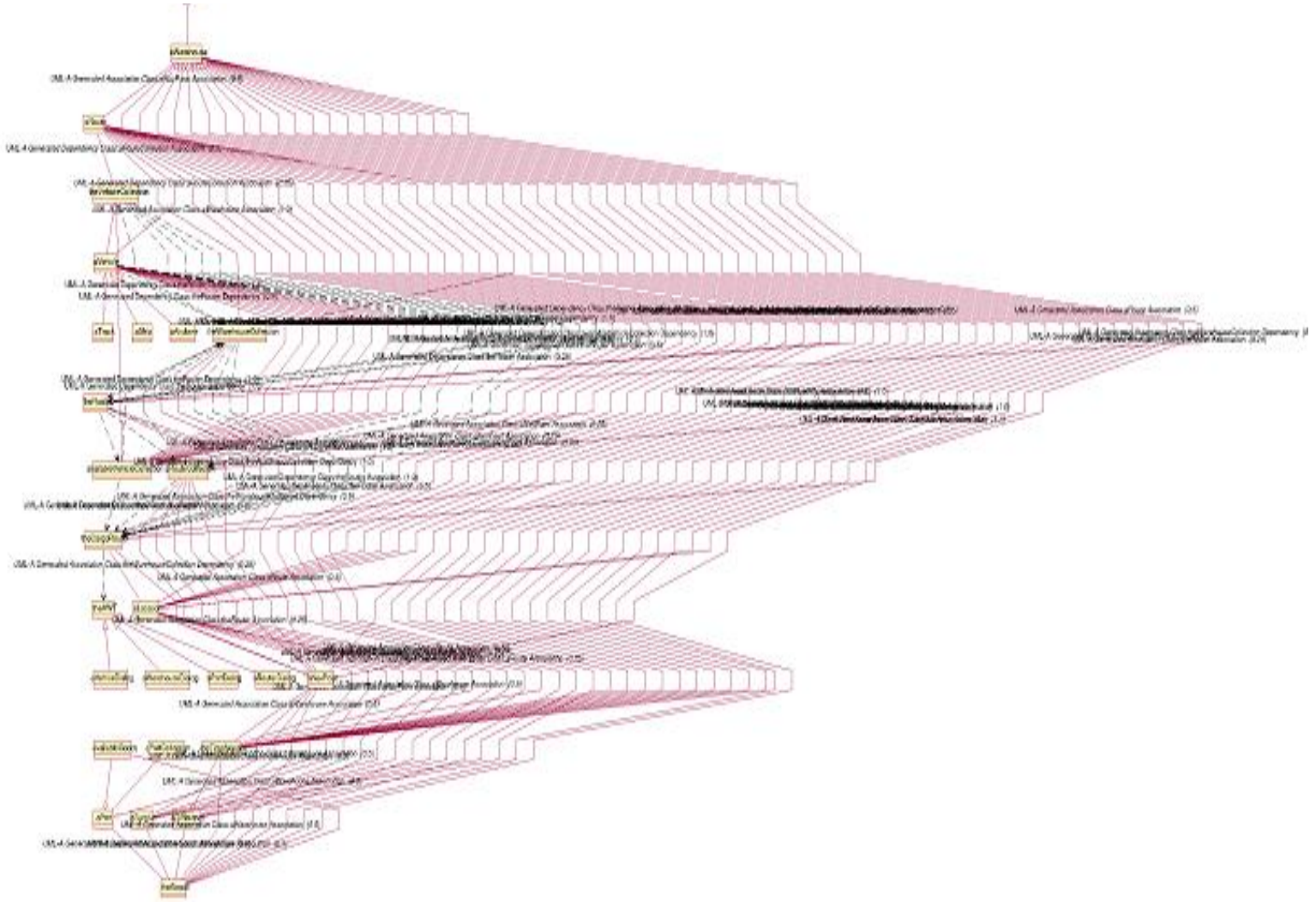
Software Architecture

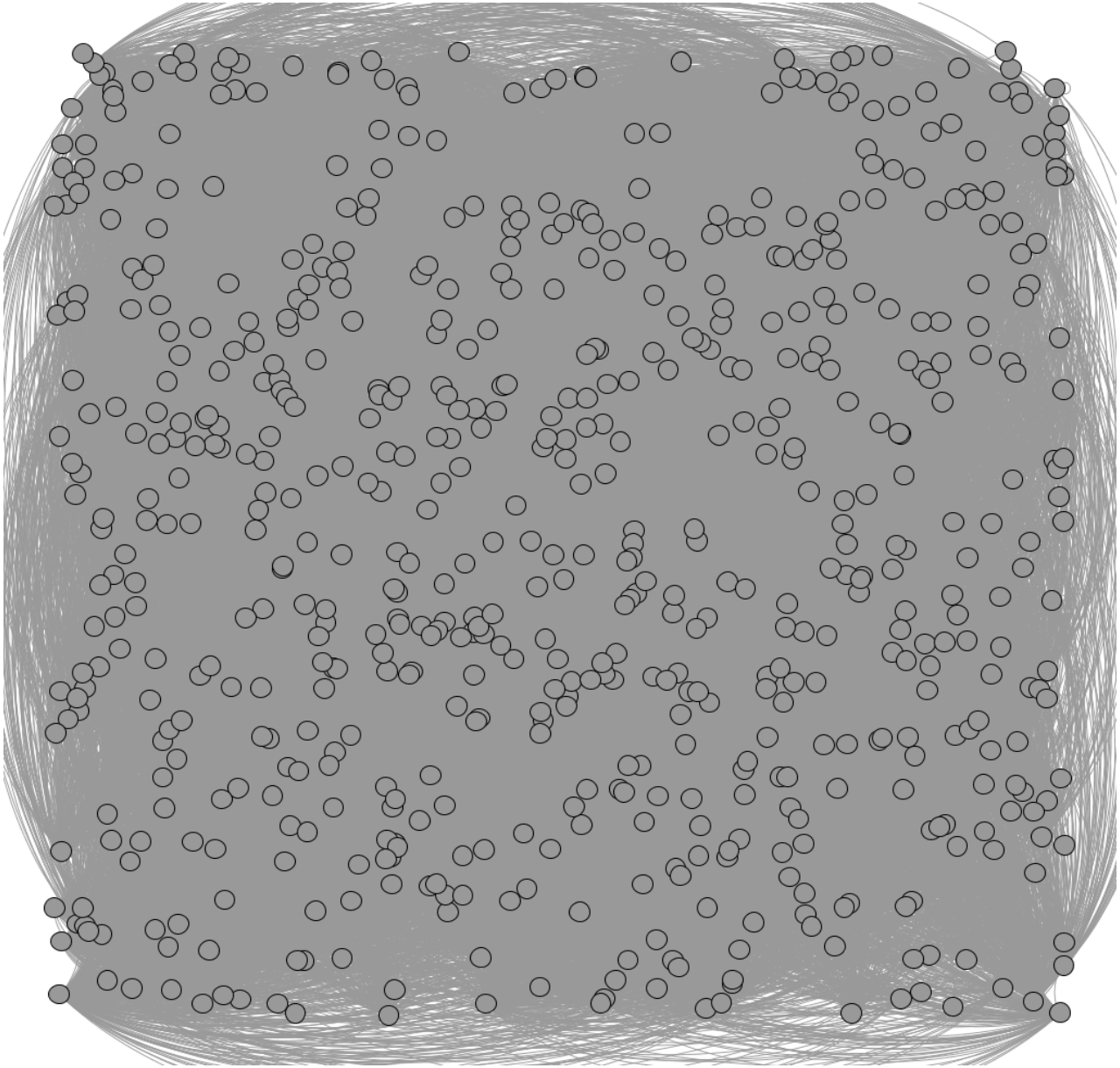
Software Is Incredibly Complex



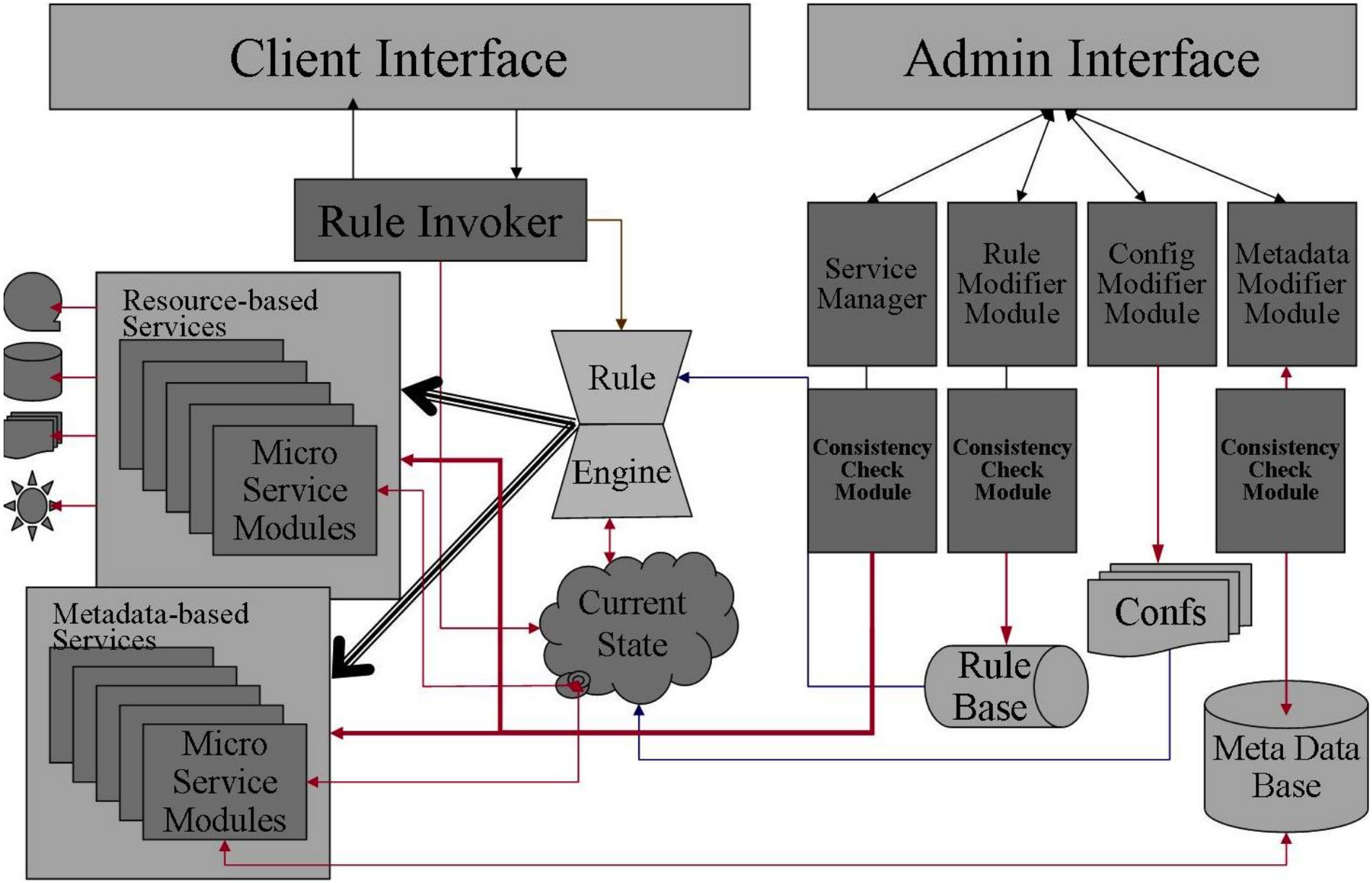
This Is What You See
(if you are lucky)



Or, More Likely



But What You Would Like to See



What is Software Architecture?

- **Definition** (we will come back to it later)
  - A software system's architecture is the set of *principal design decisions* about the system
- Software architecture is the blueprint for a software system's construction and evolution
- Design decisions encompass every facet of the system under development
  - Structure
  - Behavior
  - Interaction
  - Non-functional properties

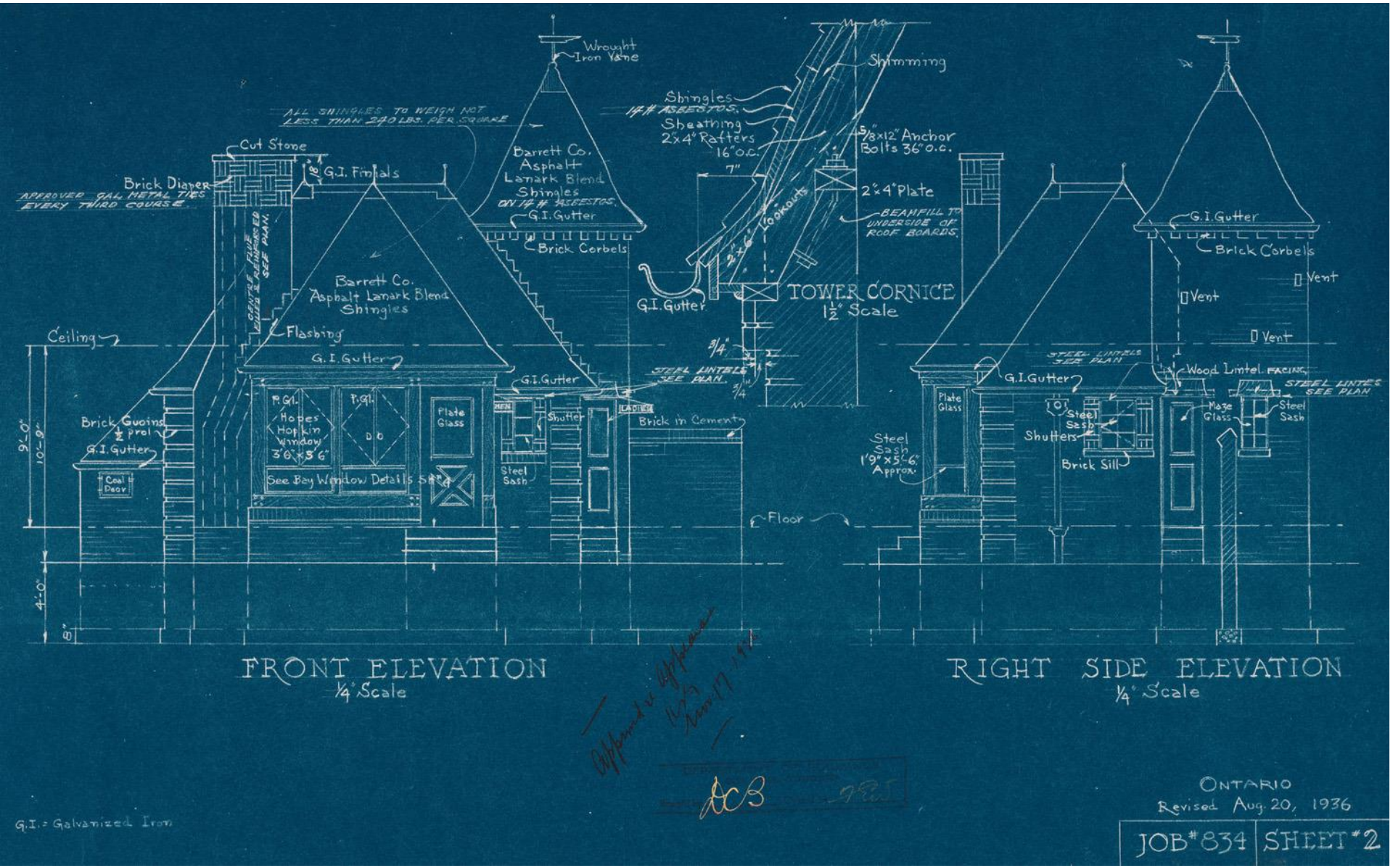
The Origins

- Software Engineers have always employed software architectures
  - Very often without realizing it!
- Address issues identified by researchers and practitioners
  - Essential software engineering difficulties
  - Unique characteristics of programming-in-the-large
  - Need for software reuse
- Many ideas originated in other (non-computing) domains

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

Quick Aside: Why “Blueprint”?

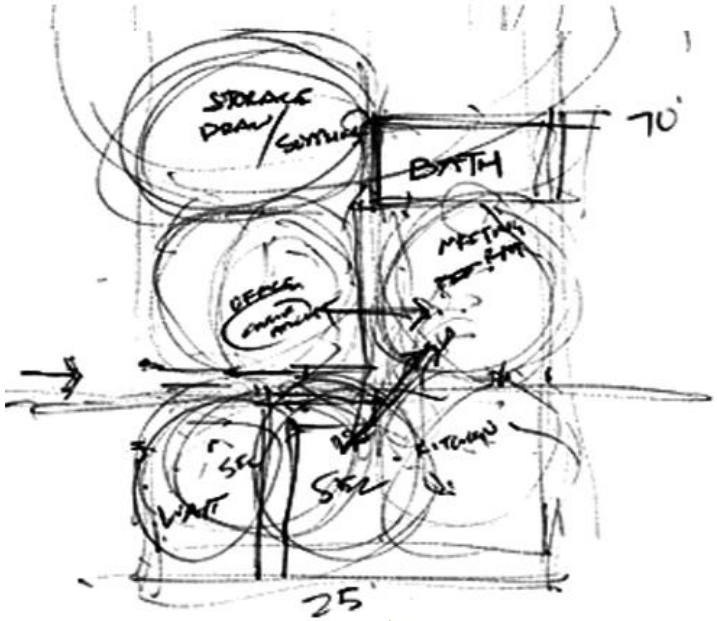


Some Obvious Parallels

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

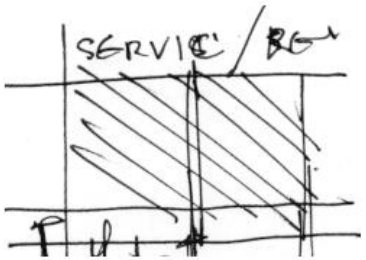
Deeper Parallels

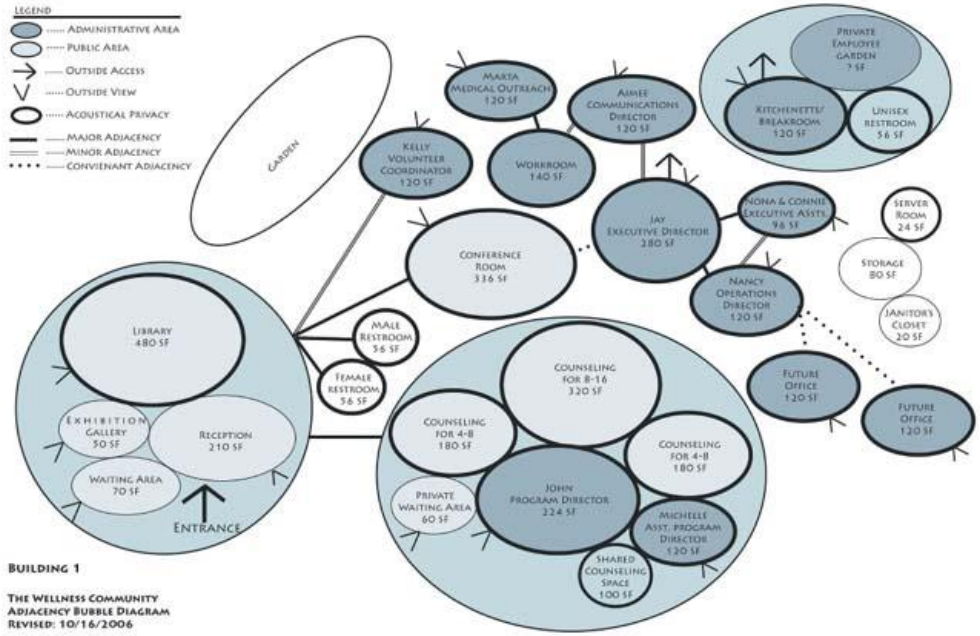
- Architecture is different from, but linked with the product/structure
- Properties of structures are induced by the design of the architecture
- The architect has a distinctive role and character
- Architecture has matured over time into a discipline
  - Wide range of solutions, techniques, and palettes of different “materials”, “colors”, and “sizes”

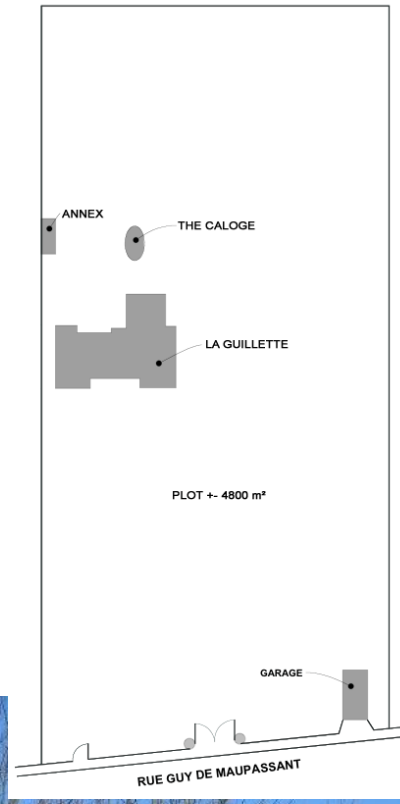

STORAGE
DRAW/SHELVING
BATH
70'
25'

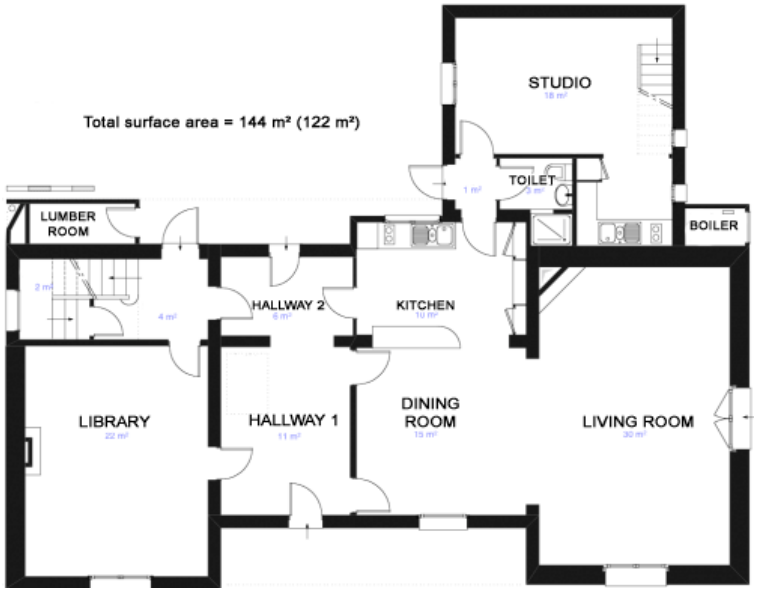

Public


SUPPORT

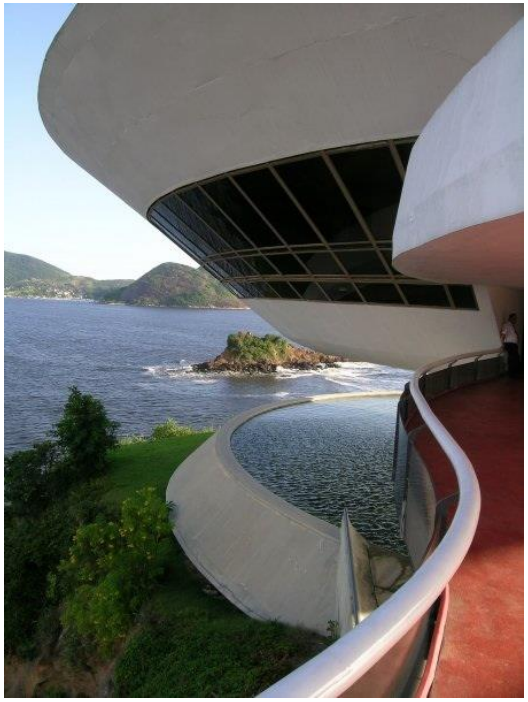

SERVICE


LEGEND
ADMINISTRATIVE AREA
PUBLIC AREA
OUTSIDE ACCESS
OUTSIDE VIEW
ACOUSTICAL PRIVACY
MAJOR ADJACENCY
MINOR ADJACENCY
CONVIENANT ADJACENCY
GARDEN
MARTA MEDICAL OUTREACH 120 SF
AIMEE COMMUNICATIONS DIRECTOR 120 SF
PRIVATE EMPLOYEE GARDEN 7 SF
KITCHENETTE/ BREAKROOM 120 SF
UNISEX RESTROOM 56 SF
KELLY VOLUNTEER COORDINATOR 120 SF
WORKROOM 140 SF
JAY EXECUTIVE DIRECTOR 280 SF
FIONA & CONNIE EXECUTIVE ASSTS 98 SF
SERVER ROOM 24 SF
STORAGE 80 SF
JANITOR'S CLOSET 20 SF
CONFERENCE ROOM 336 SF
NANCY OPERATIONS DIRECTOR 120 SF
LIBRARY 480 SF
MALE RESTROOM 56 SF
FEMALE RESTROOM 56 SF
EXHIBITION GALLERY 50 SF
RECEPTION 210 SF
WAITING AREA 70 SF
ENTRANCE
COUNSELING FOR 8-16 320 SF
COUNSELING FOR 4-8 180 SF
COUNSELING FOR 4-8 180 SF
FUTURE OFFICE 120 SF
FUTURE OFFICE 120 SF
PRIVATE WAITING AREA 60 SF
JOHN PROGRAM DIRECTOR 224 SF
MICHELLE ASST. PROGRAM DIRECTOR 120 SF
SHARED COUNSELING SPACE 100 SF
BUILDING 1
THE WELLNESS COMMUNITY
ADJACENCY BUBBLE DIAGRAM
REVISED: 10/16/2006


ANNEX
THE CALOGE
LA GUILLETTE
PLOT +- 4800 m²
GARAGE
RUE GUY DE MAUPASSANT


Total surface area = 144 m² (122 m²)
STUDIO
TOILET
LUMBER ROOM
BOILER
HALLWAY 2
KITCHEN
LIBRARY
HALLWAY 1
DINING ROOM
LIVING ROOM


Total surface area : 140 m² (117 m²)
BEDROOM
BEDROOM
TOILET
OPEN SPACE
LANDING
BEDROOM
BEDROOM
BATHROOM
BEDROOM
BEDROOM
BALCONY



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 changeable 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

Changeability of Buildings

It is doable

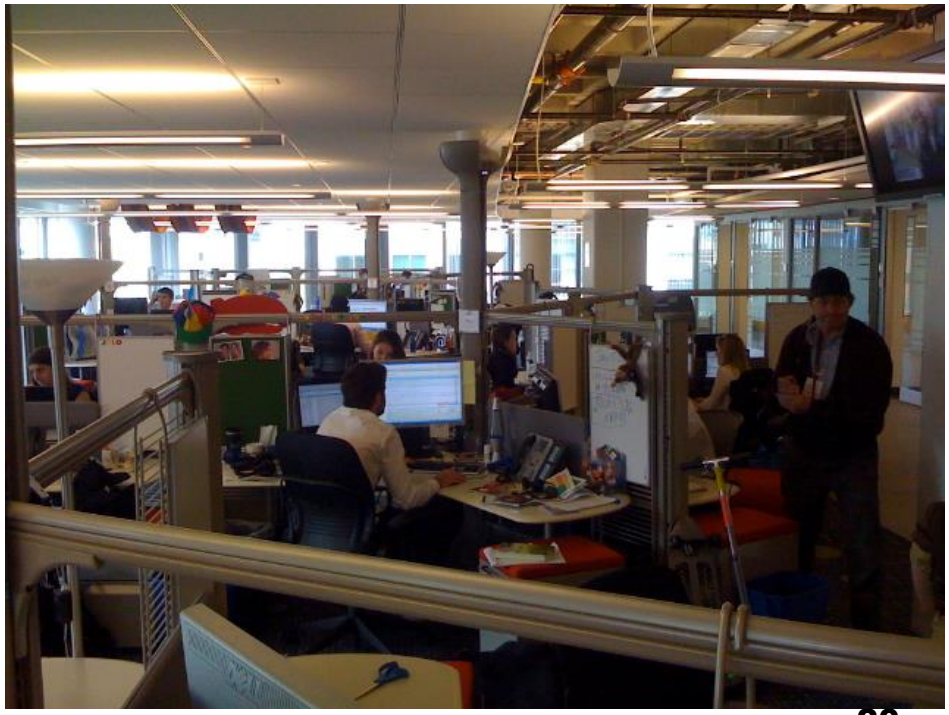
But it’s not easy or cheap!

The Two Construction Industries

vs.

Or, More Realistically

vs.

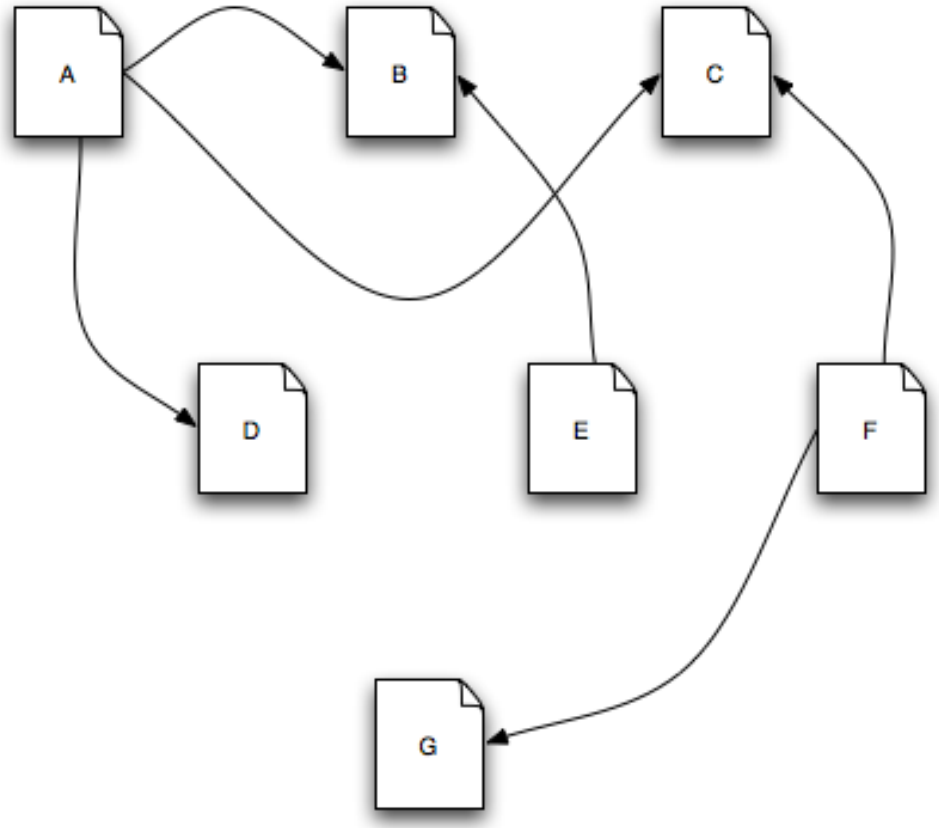


Or, If You Get Unlucky

vs.

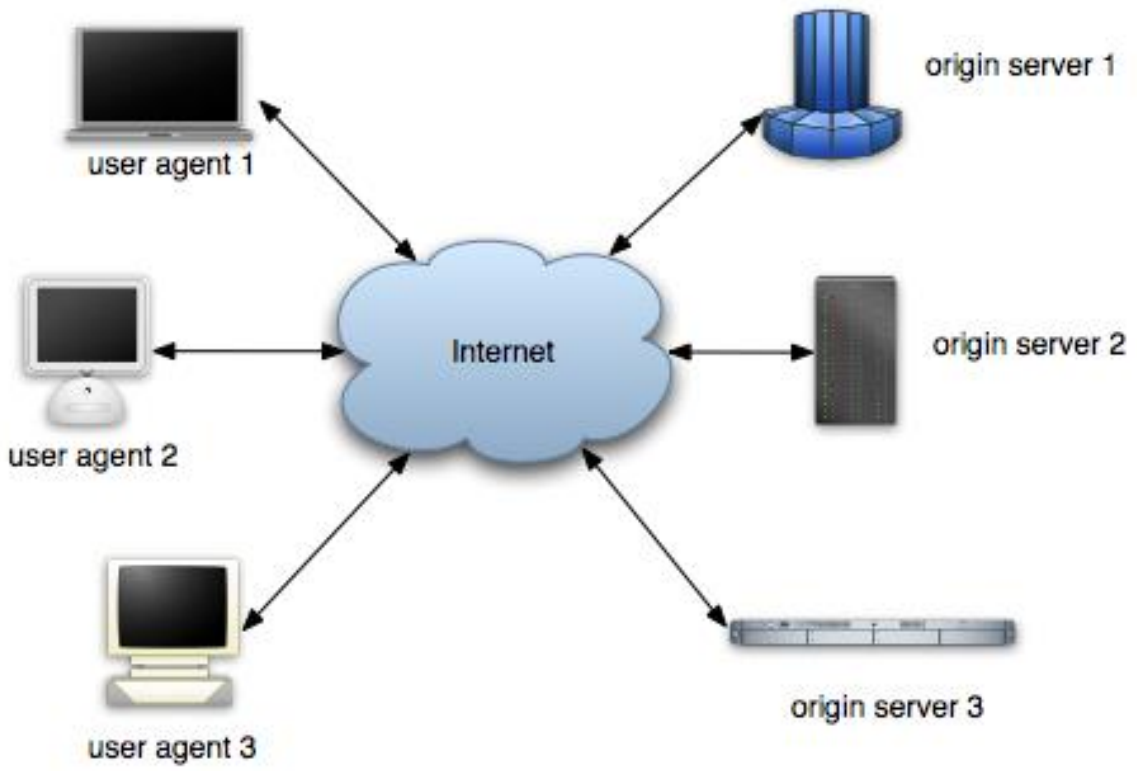
Architecture in Action: WWW

- This is the Web



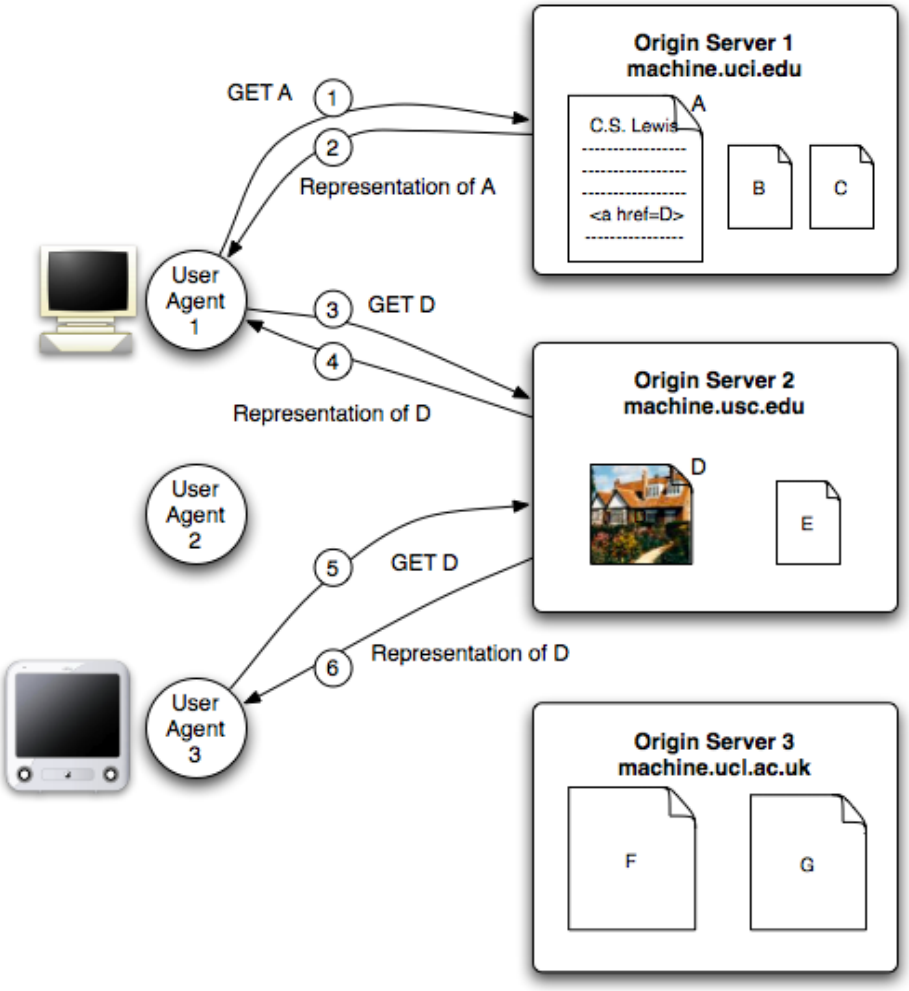
Architecture in Action: WWW

- So is this



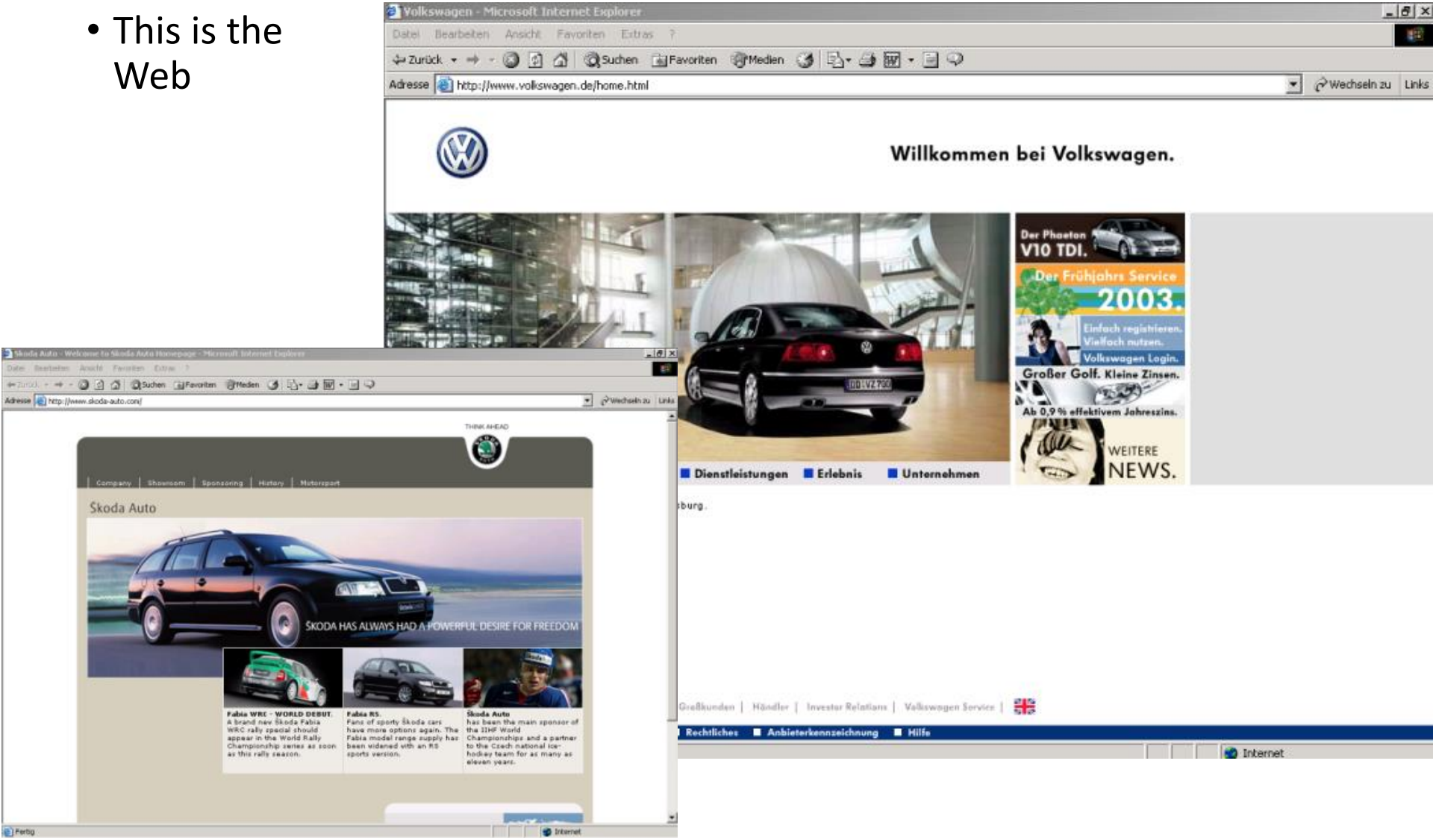
Architecture in Action: WWW

- And this



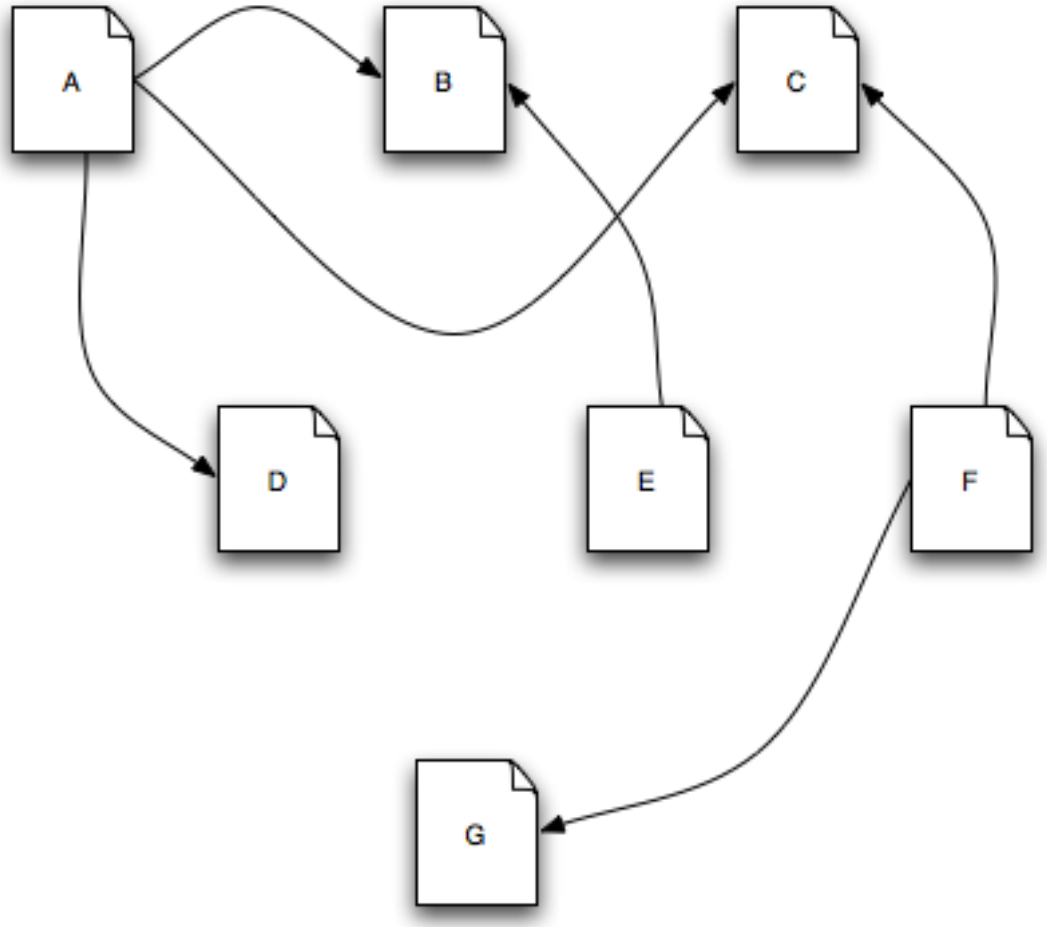
Architecture in Action: WWW

- This is the Web



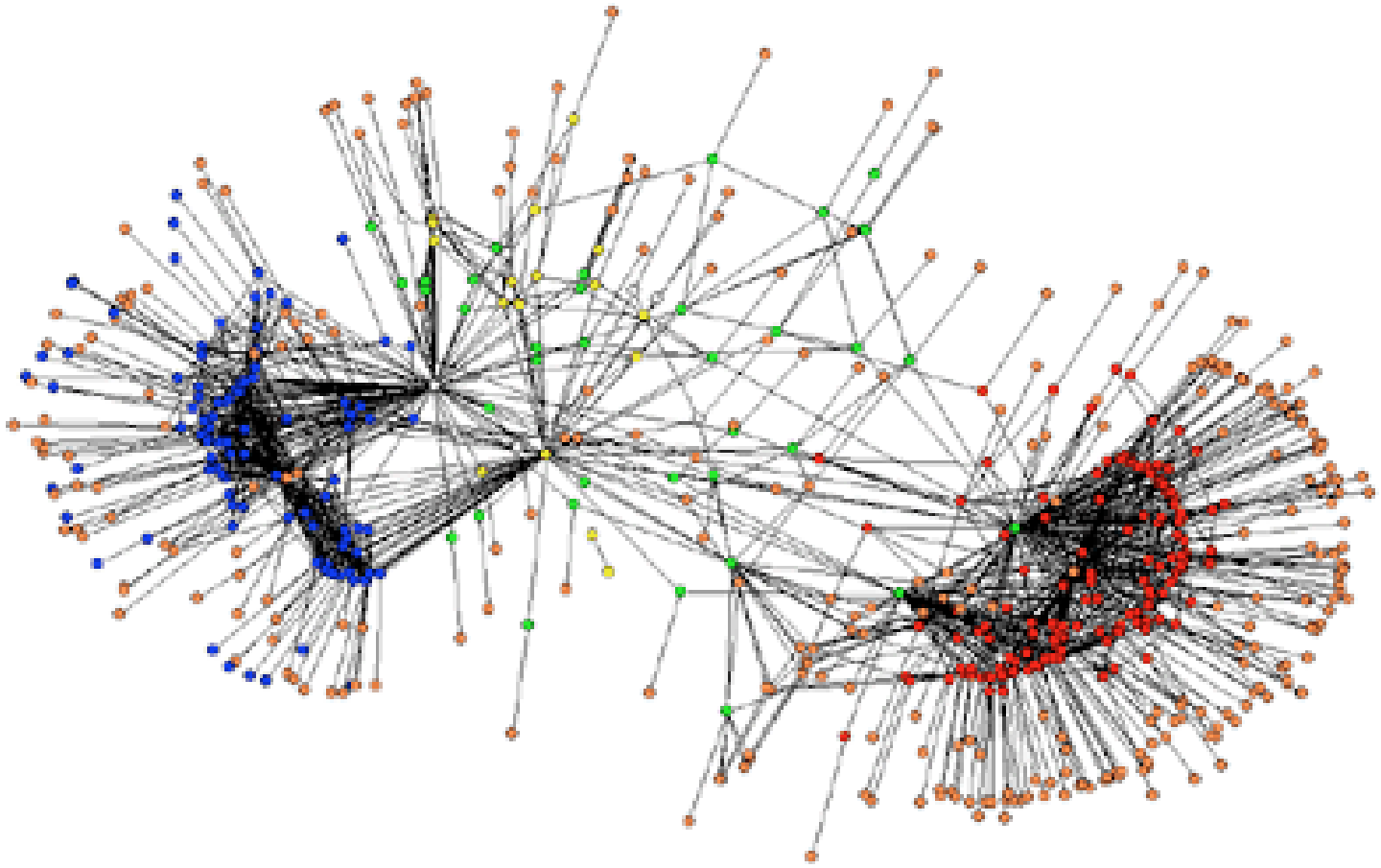
Architecture in Action: WWW

- So is this



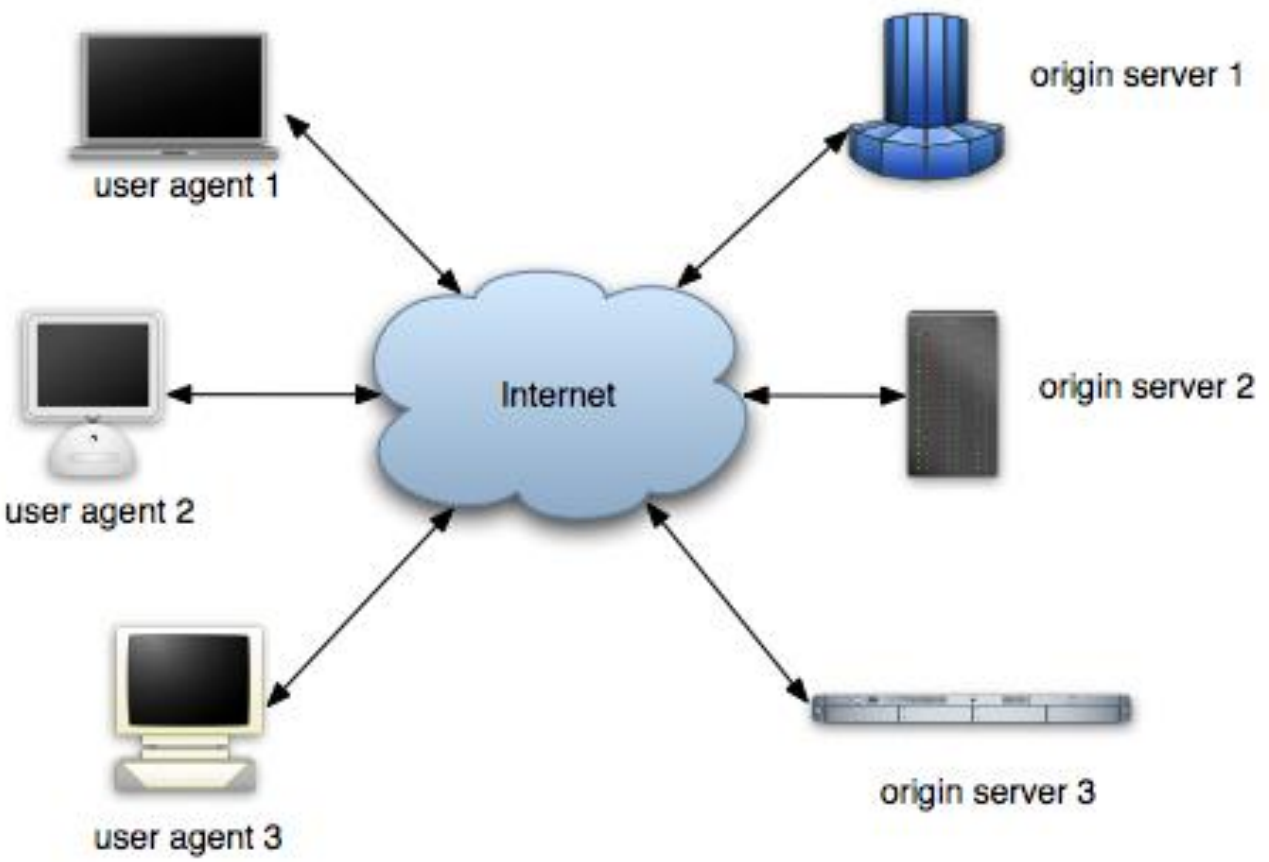
Architecture in Action: WWW

- So is this



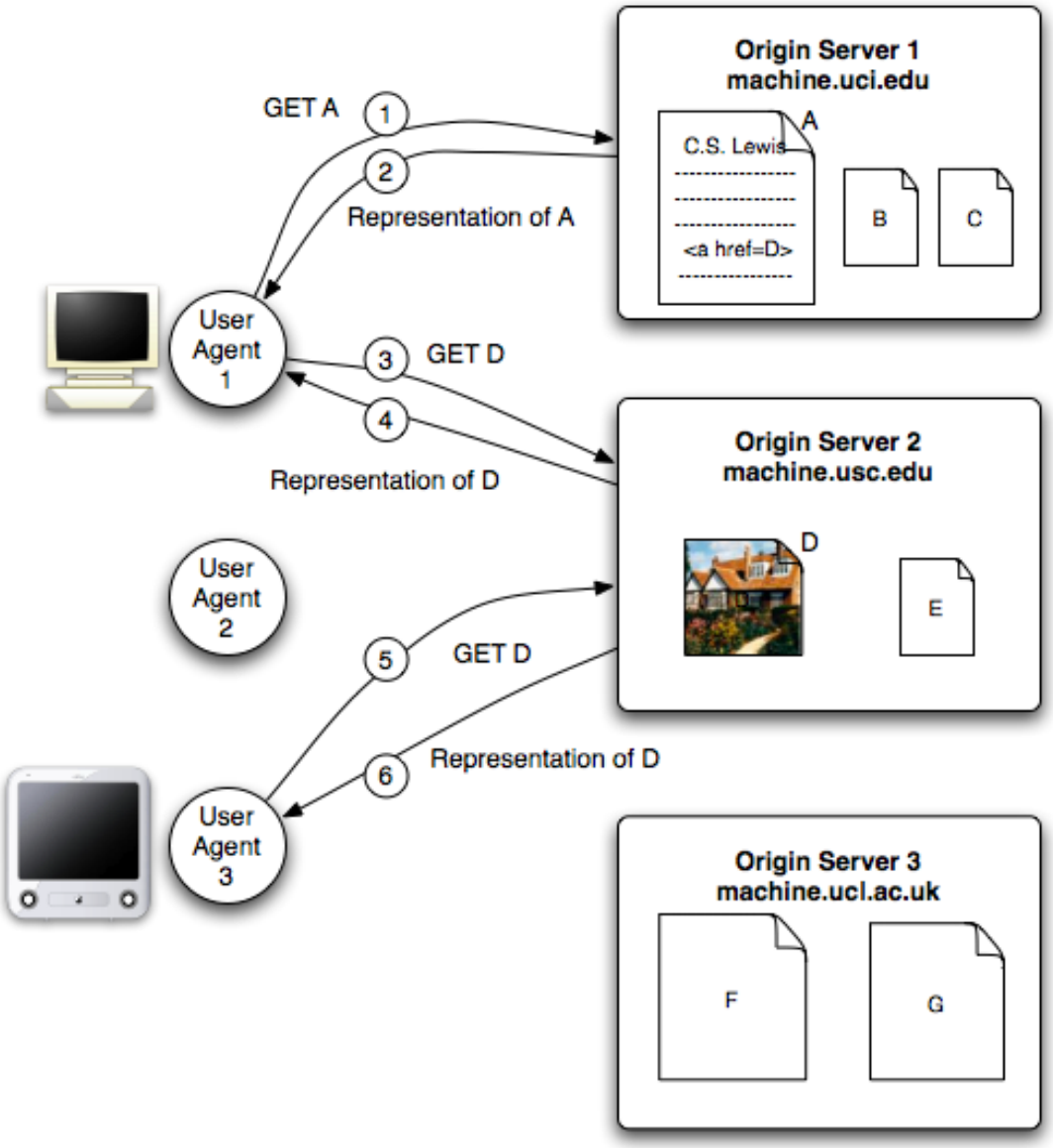
Architecture in Action: WWW

- And 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, or “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 of 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.
- All communication between user agents and origin servers must be fully self-contained. (So-called "stateless interactions")

WWW's Architecture

- Architecture of the Web is wholly separate from the code
- There is no single 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

Architecture in Action: Product Line

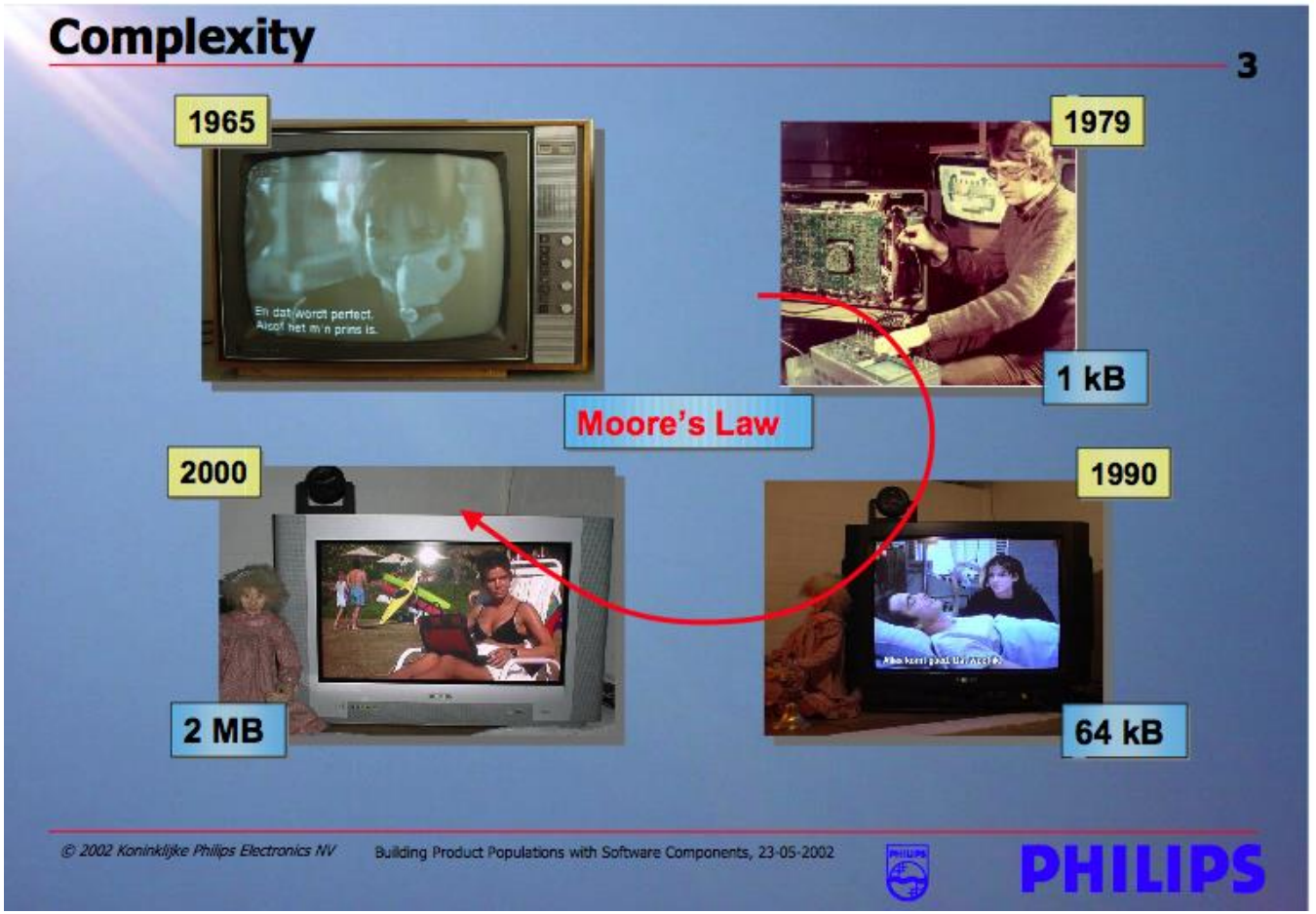
- Motivating example
  - A consumer is interested in a 35-inch HDTV with a built-in DVD player for the North American market.

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

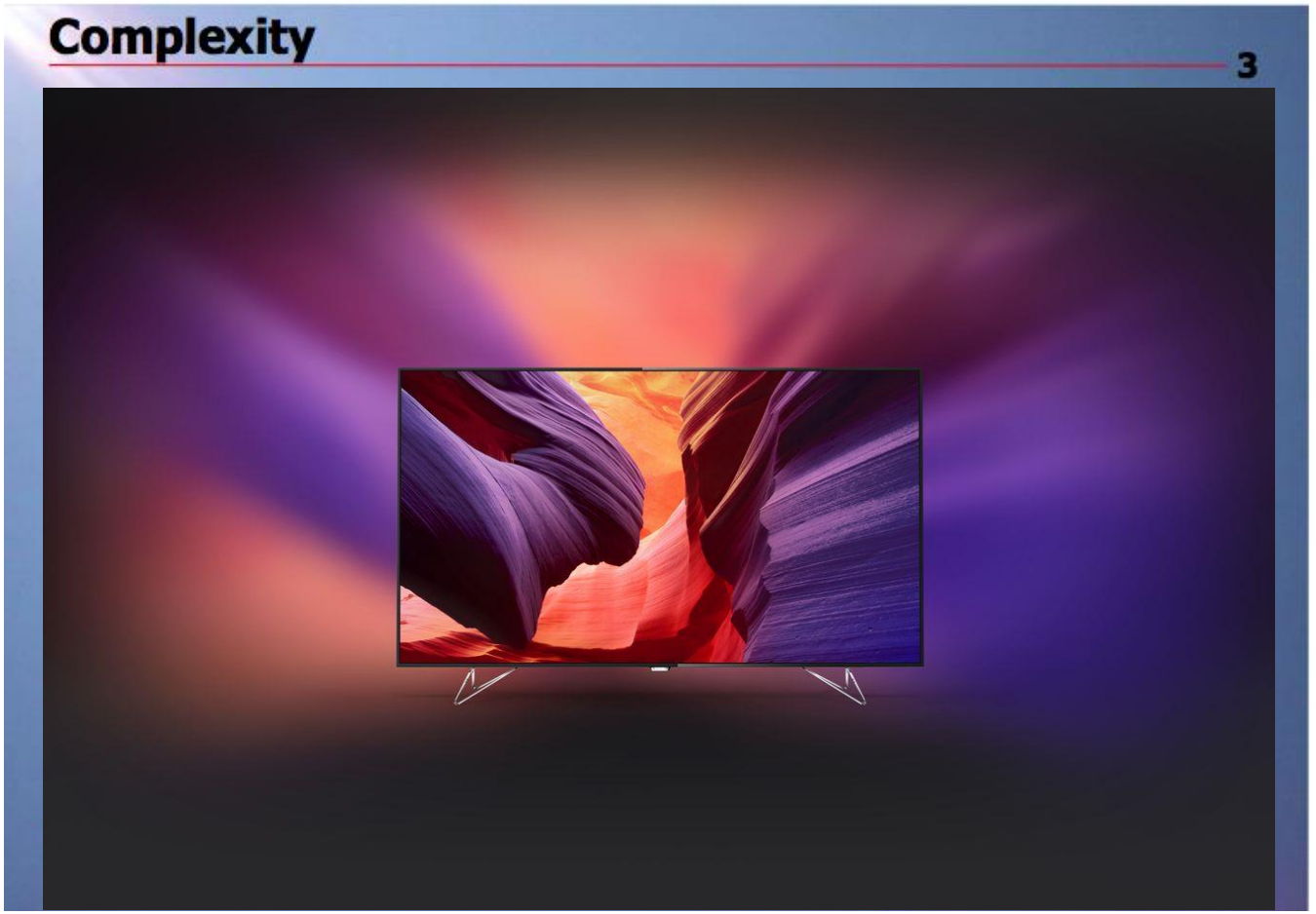
    This particular television/DVD player will be very similar to a 35-inch HDTV without the DVD player, and also to a 35-inch HDTV with a built-in DVD player for the European market, where the TV must be able to handle PAL or SECAM encoded broadcasts, rather than North America's NTSC format.

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

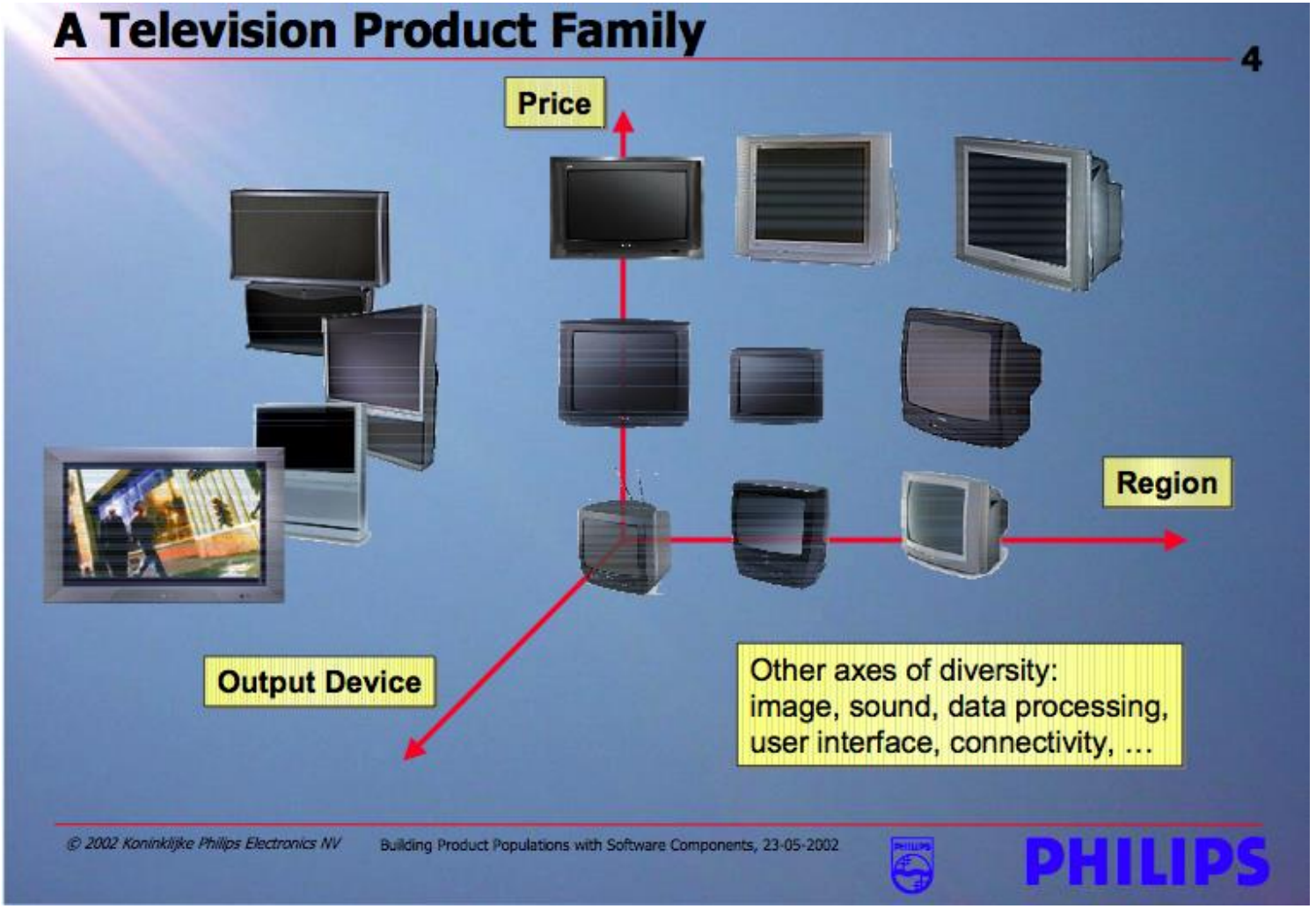
Growing Sophistication of Consumer Devices



Growing Sophistication of Consumer Devices



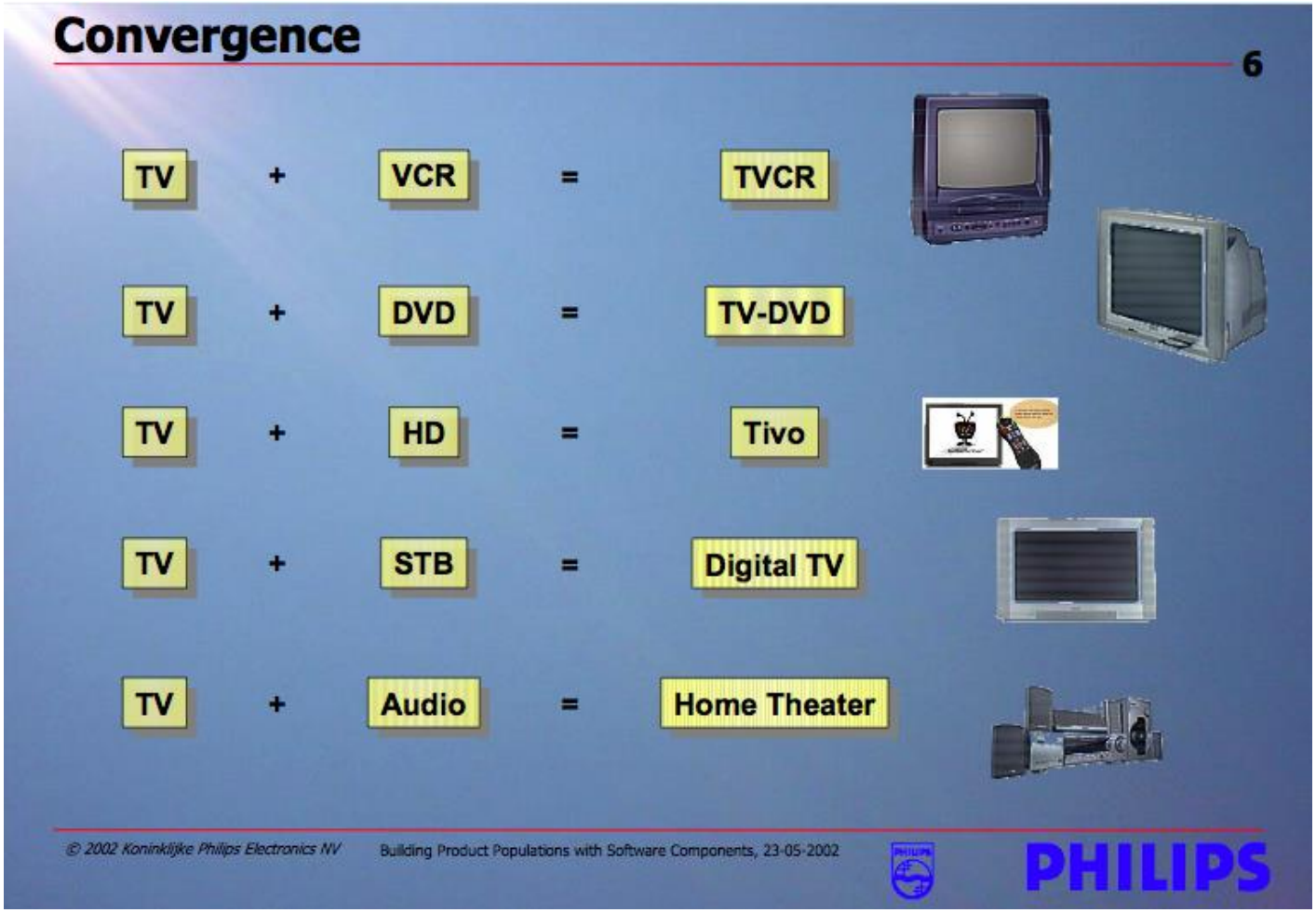
Families of Related Products



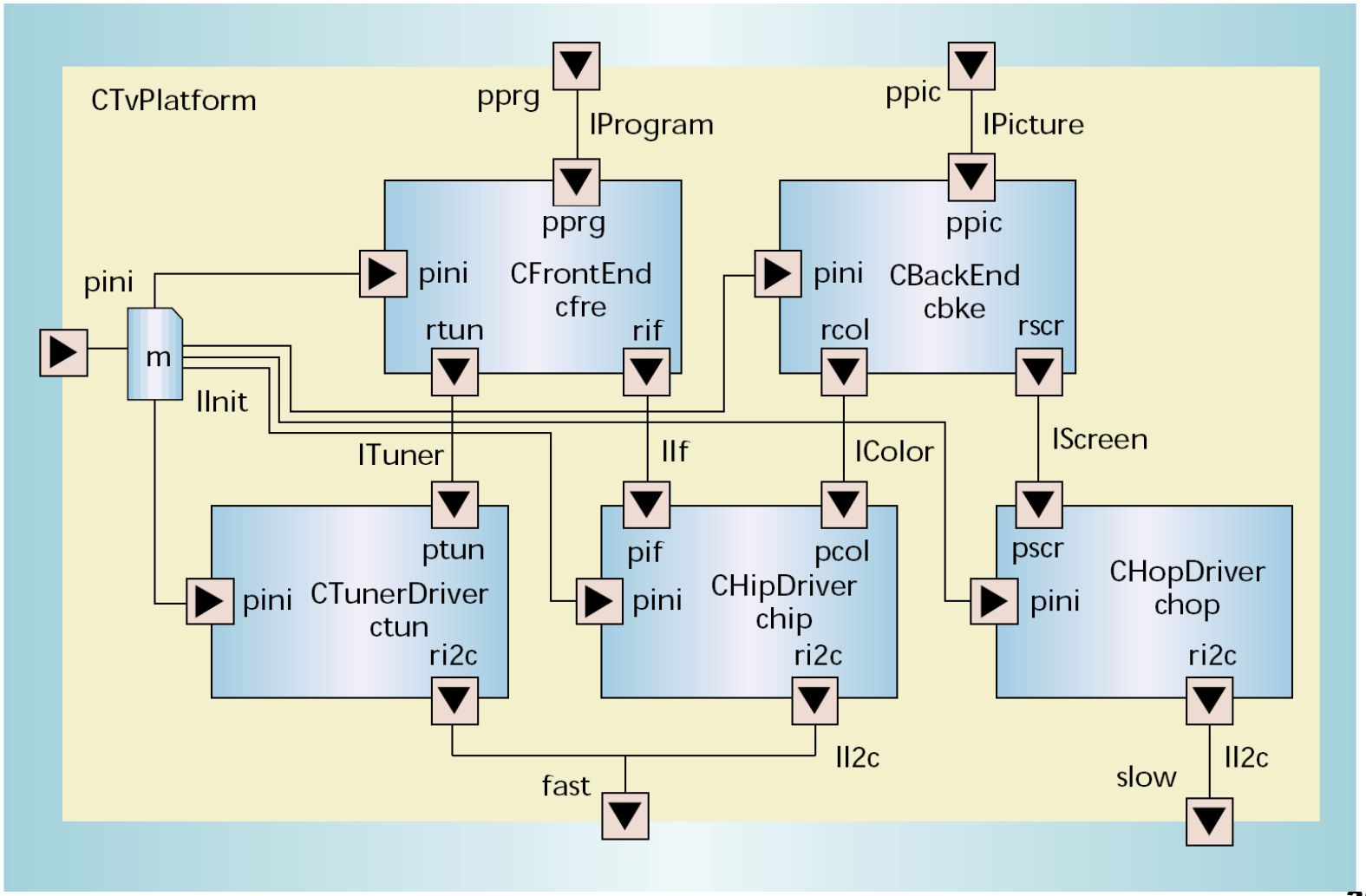
Build Product Lines!

- 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

And Product Populations



Philips' Solution – Architecture



What is Software Architecture?

- **Definition:**
  - A software system's architecture is the set of *principal design decisions* about the system
- Software architecture is the blueprint for a software system's construction and evolution
- Design decisions encompass every facet of the system under development
  - Structure
  - Behavior
  - Interaction
  - Non-functional properties

What is “Principal”?

- “Principal” implies a degree of importance that grants a design decision “architectural status”
  - It implies that not all design decisions are architectural
  - That is, they do not necessarily impact a system’s architecture
- How one defines “principal” will depend on what the stakeholders define as the system goals

Other Definitions of Software Architecture

- Perry and Wolf
  - Software Architecture = { Elements, Form, Rationale }
    what *how* *why*
- Shaw and Garlan
  - Software architecture [is a level of design that] involves
    - the description of elements from which systems are built,
    - interactions among those elements,
    - patterns that guide their composition, and
    - constraints on these patterns.
- Kruchten
  - Software architecture deals with the design and implementation of the high-level structure of software.
  - Architecture deals with abstraction, decomposition, composition, style, and *aesthetics*.

Safari File Edit View History Bookmarks Window Help (100%) Wed Sep 19 1:10 PM

software architecture diagram - Google Search

http://www.google.com/search?client=safari&rls=en&q=software+architecture&oe=UTF-8&um=1&ie=UTF-8&hl=en&tbm=isch&source=og&sa=N&tab=wi&ei=IyVaU Reader

software architecture

NSF ESD Speedtest Google Grants NSF SaTC SoftArch Reading Group Army BAA TMD Book Project IP->Loc MS SEIF Gmaps Pedometer Yugo CrossFit GeoF1 NSF Core ICSE'11 Chairs TSE Math Worksheets Kailash

+Nenad Search Images Maps Play YouTube News Gmail Documents Calendar More

Google

software architecture

Nenad Medvidovic + Share

Search

About 2,180,000 results (0.34 seconds)

SafeSearch moderate

Web
Images
Maps
Videos
News
Shopping
More

Any time
Past 24 hours
Past week
Custom range...

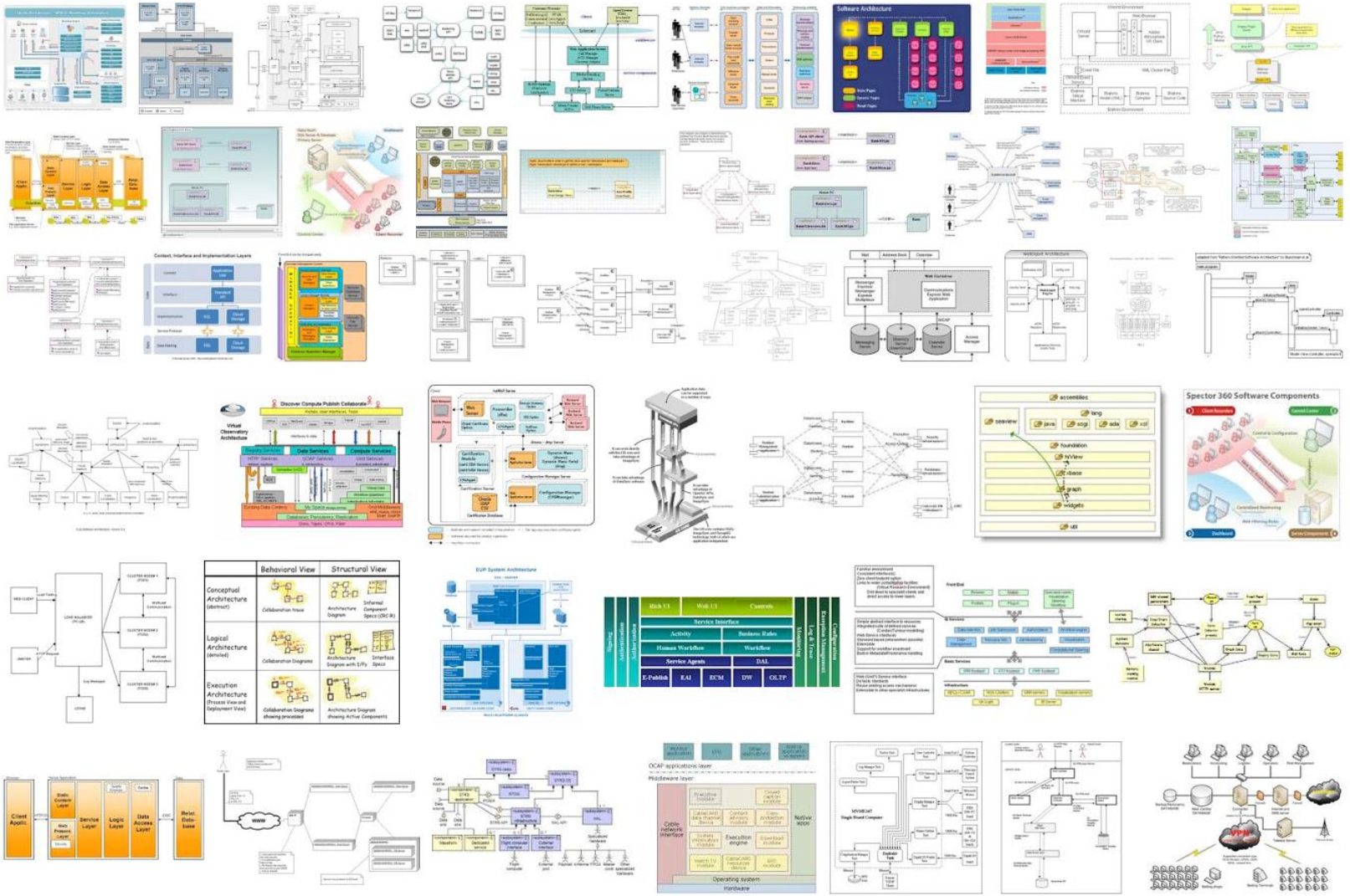
All results
By subject

Any size
Large
Medium
Icon
Larger than...
Exactly...

Any color
Full color
Black and white

Any type
Face
Photo
Clip art
Line drawing

Standard view
Show sizes



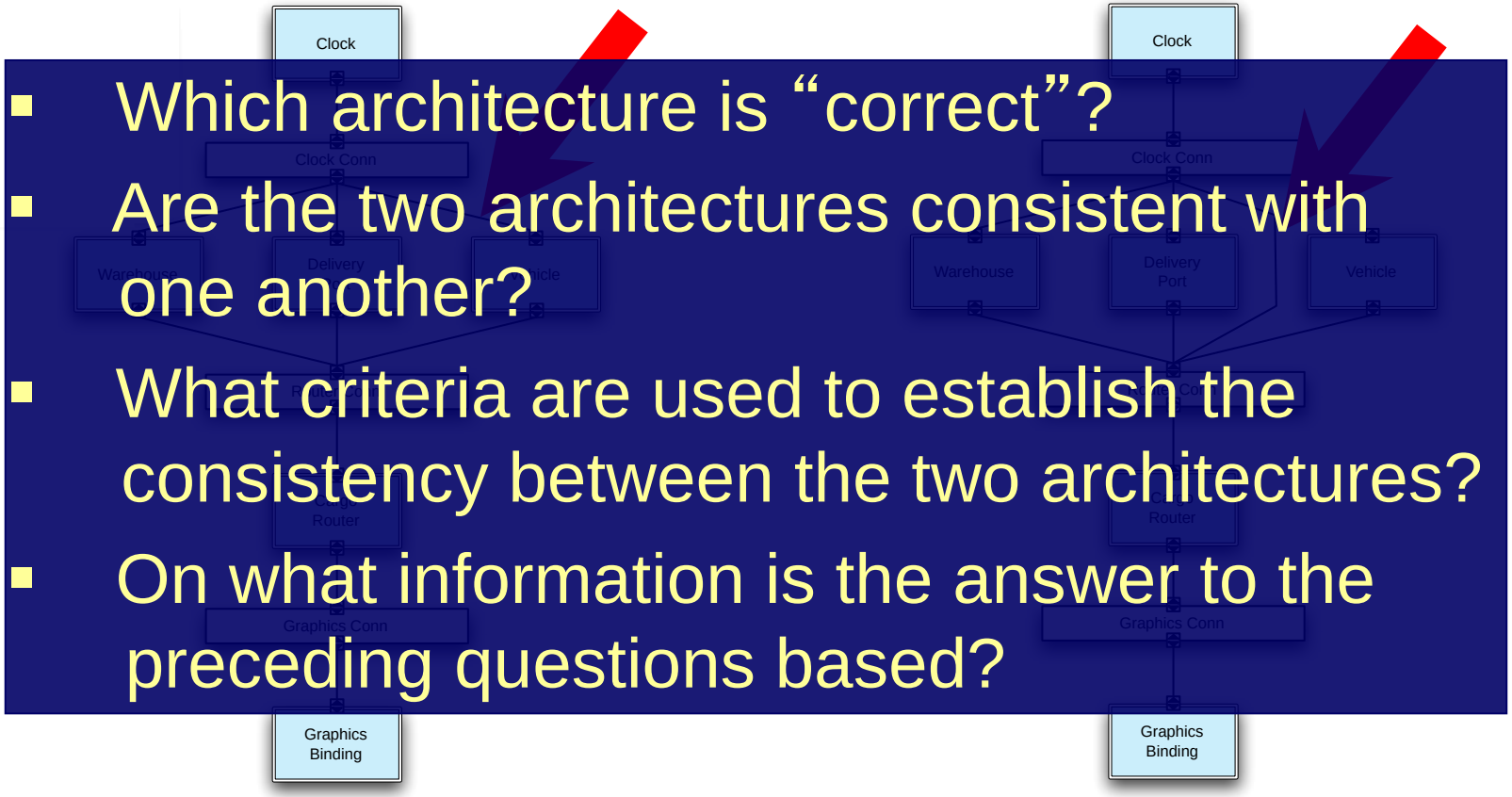
Temporal Aspect

- Design decisions are and unmade over a system's lifetime
  → Architecture has a temporal aspect
- At any given point in time the system has only one architecture
- A system's architecture will change over time

Prescriptive vs. Descriptive Architecture

- A system's *prescriptive architecture* captures the design decisions made prior to the system's construction
  - It is the *as-conceived* or *as-intended* architecture
- A system's *descriptive architecture* describes how the system has been built
  - It is the *as-implemented* or *as-realized* architecture

Two Architectures Side-by-Side



- Which architecture is “correct”?
- Are the two architectures consistent with one another?
- What criteria are used to establish the consistency between the two architectures?
- On what information is the answer to the preceding questions based?

Prescriptive Architecture

Descriptive Architecture

Architectural Evolution

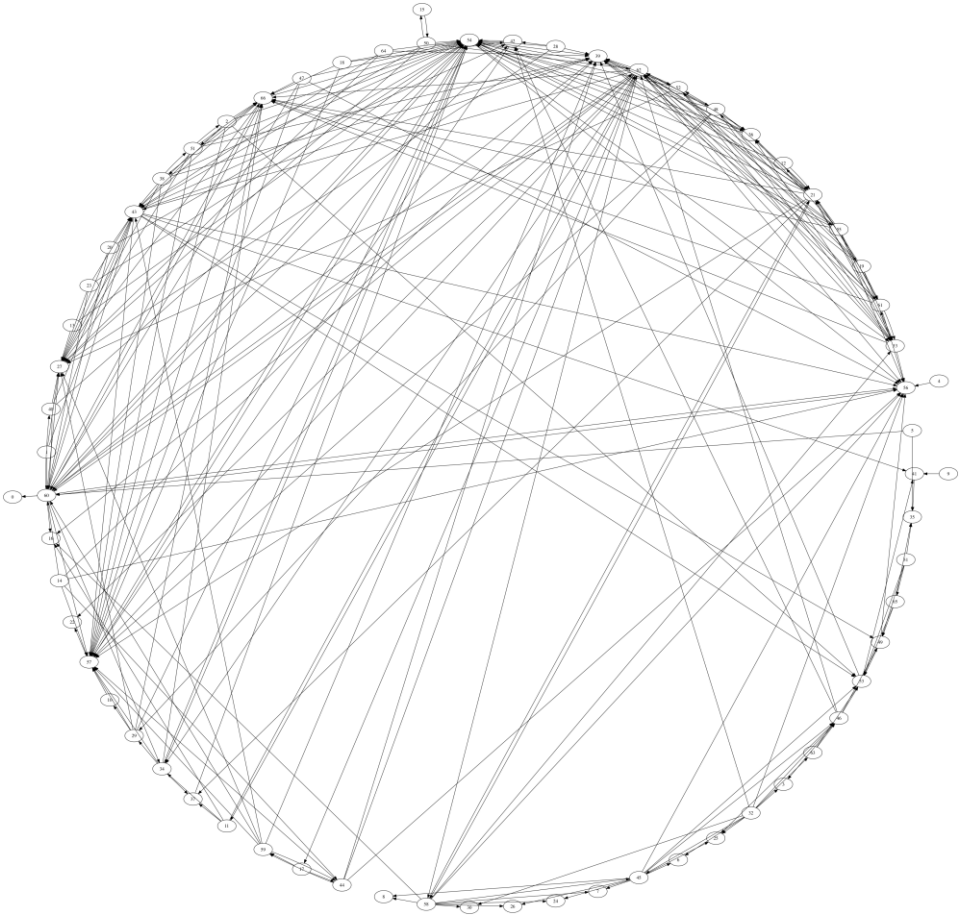
- When a system evolves, ideally its prescriptive architecture is modified first
- In practice, the system's implementation – i.e., the descriptive architecture – is often directly modified
- This happens because of
  - Developer sloppiness
  - Perception of short deadlines which prevent thinking through and documenting
  - Lack of documented prescriptive architecture
  - Need or desire for code optimizations
  - Inadequate techniques or tool support

Architectural Degradation / Decay

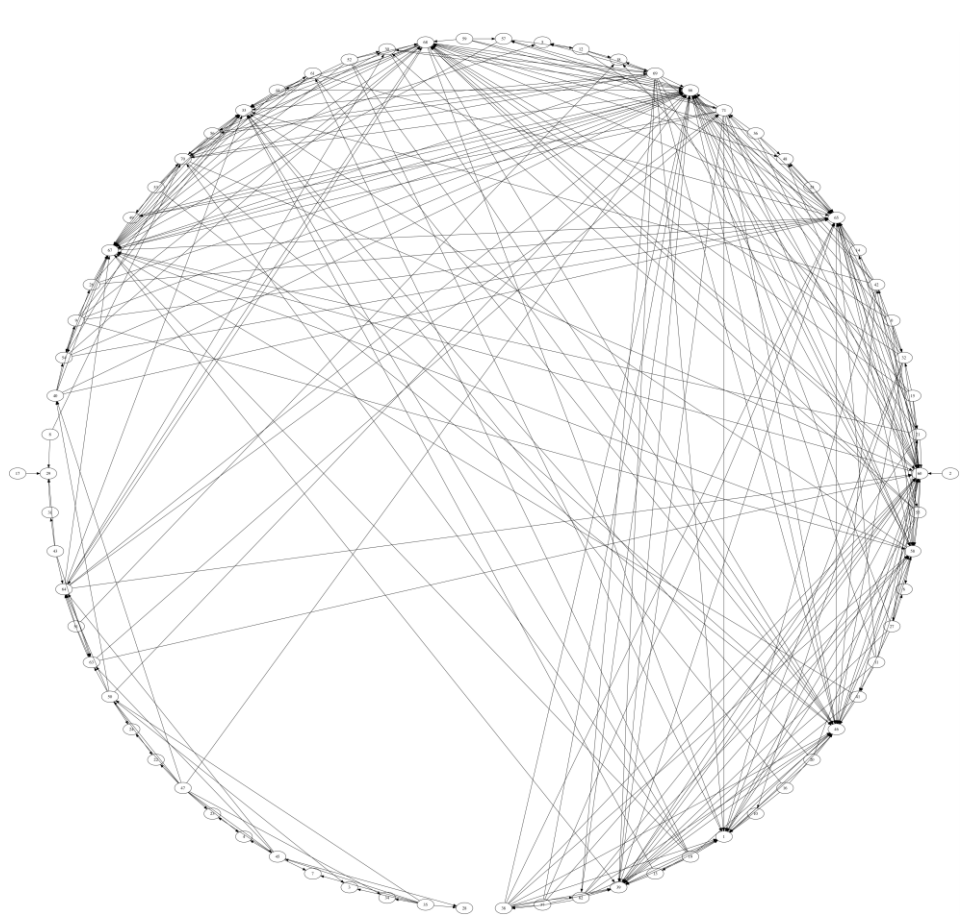
- Two related concepts
  - Architectural drift
  - Architectural erosion
- *Architectural drift* is introduction of principal design decisions into a system's descriptive architecture that
  - are not included in, encompassed by, or implied by the prescriptive architecture
  - but which do not violate any of the prescriptive architecture's design decisions
- *Architectural erosion* is the introduction of architectural design decisions into a system's descriptive architecture that violate its prescriptive architecture

At What Point Does Change Become Decay?

Apache Chukwa 0.3.0



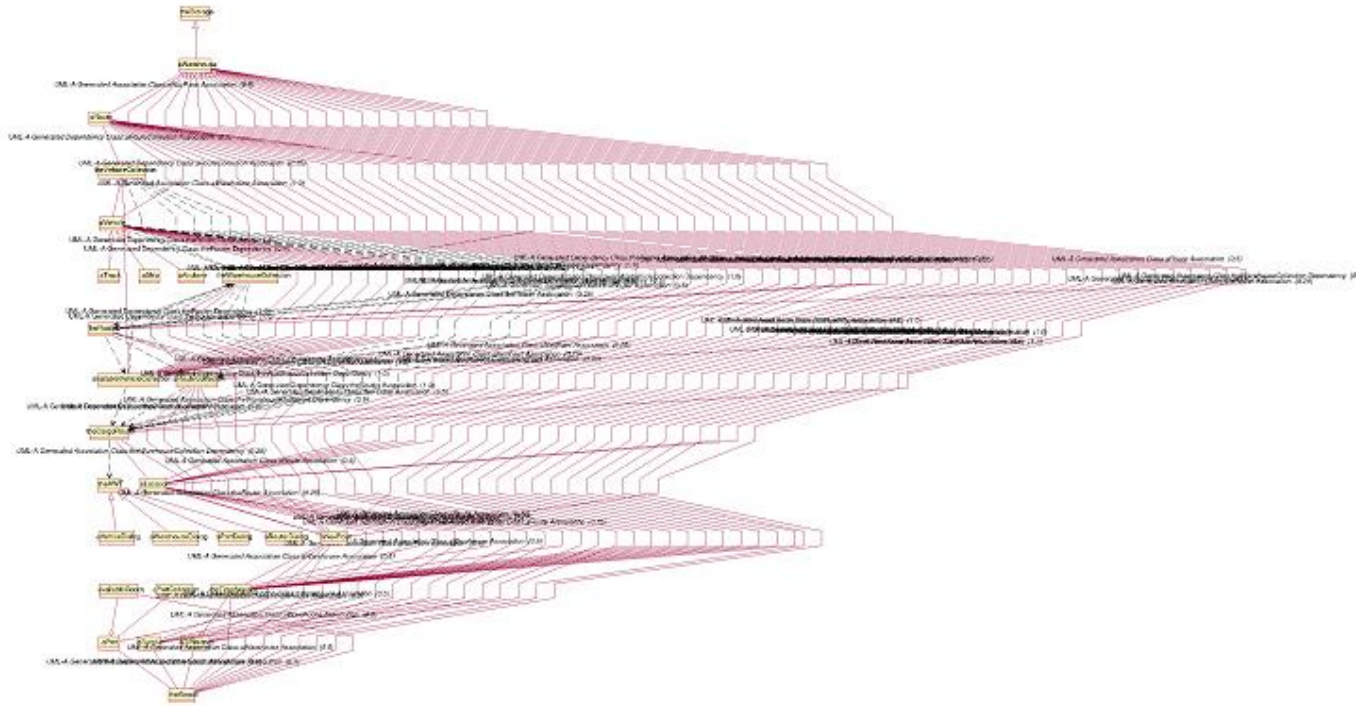
Apache Chukwa 0.4.0



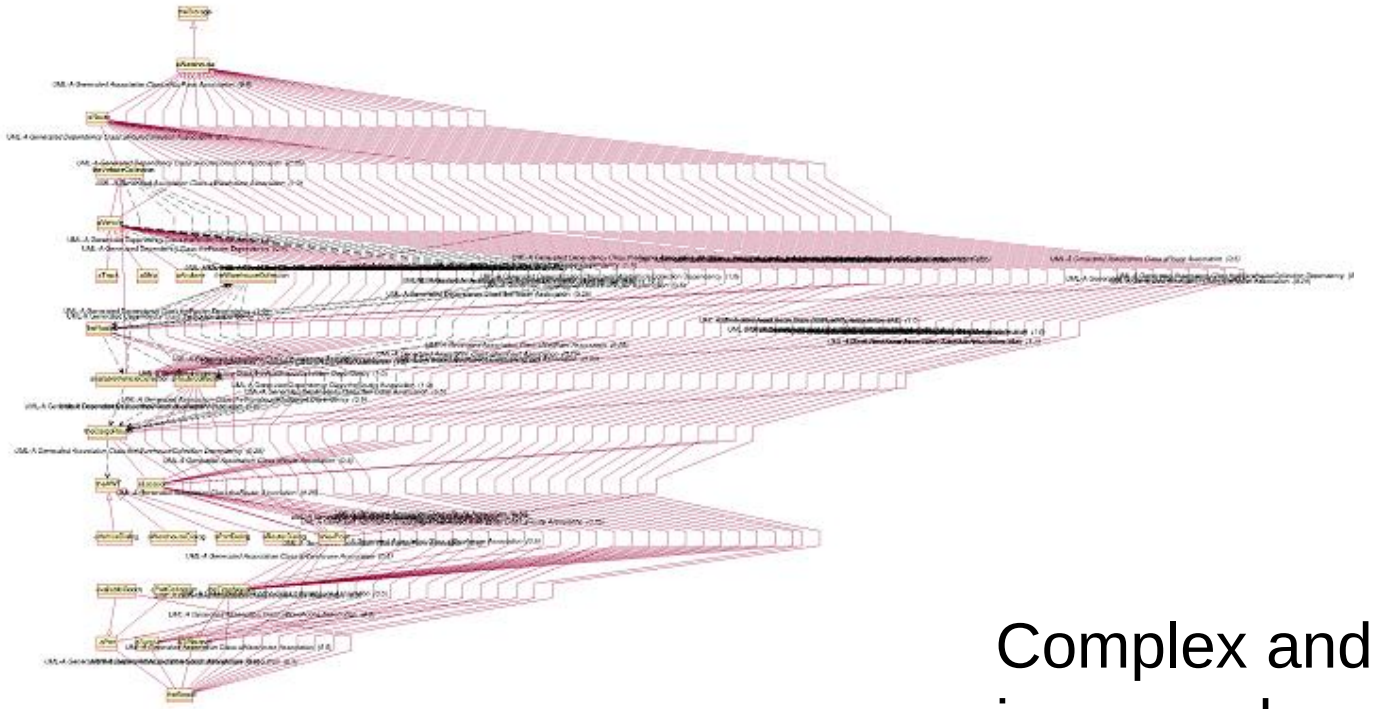
Architecture Recovery

- If architectural degradation is allowed to occur, one will be forced to *recover* the system's architecture sooner or later
- *Architectural recovery* is the process of determining a software system's architecture from its implementation-level artifacts
- Implementation-level artifacts can be
  - Source code
  - Executable files
  - Java .class files

Implementation-Level View of an Application



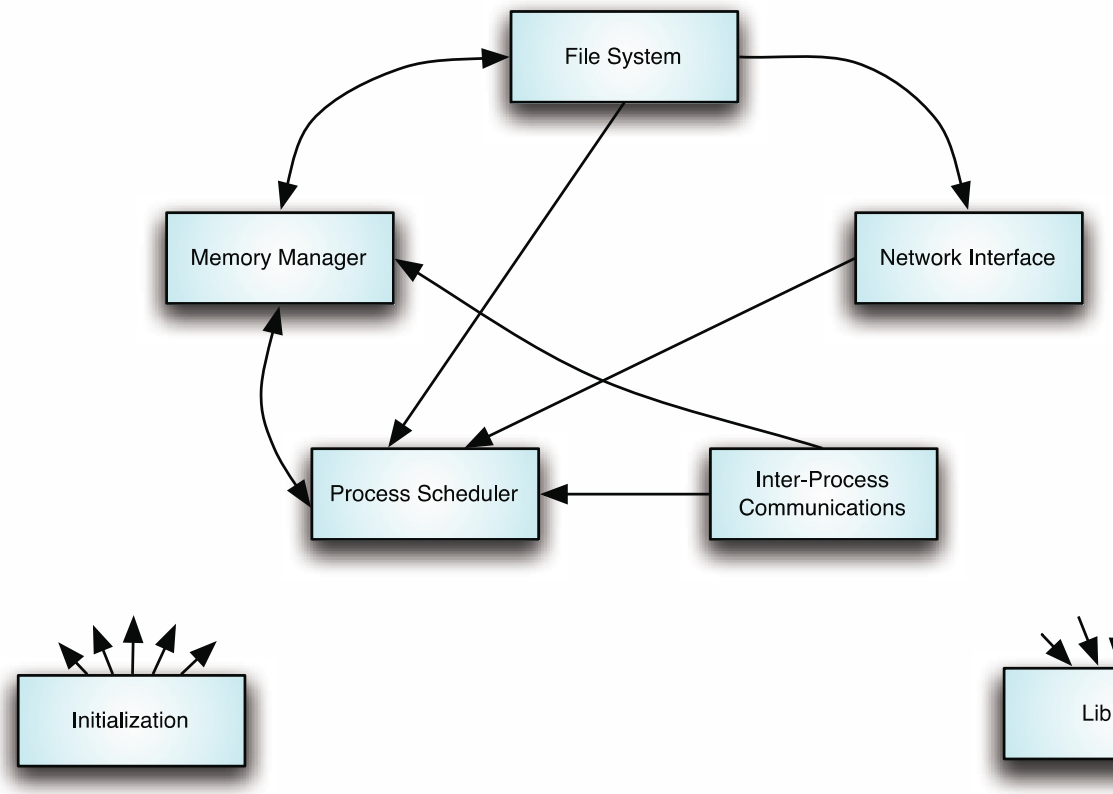
Implementation-Level View of an Application

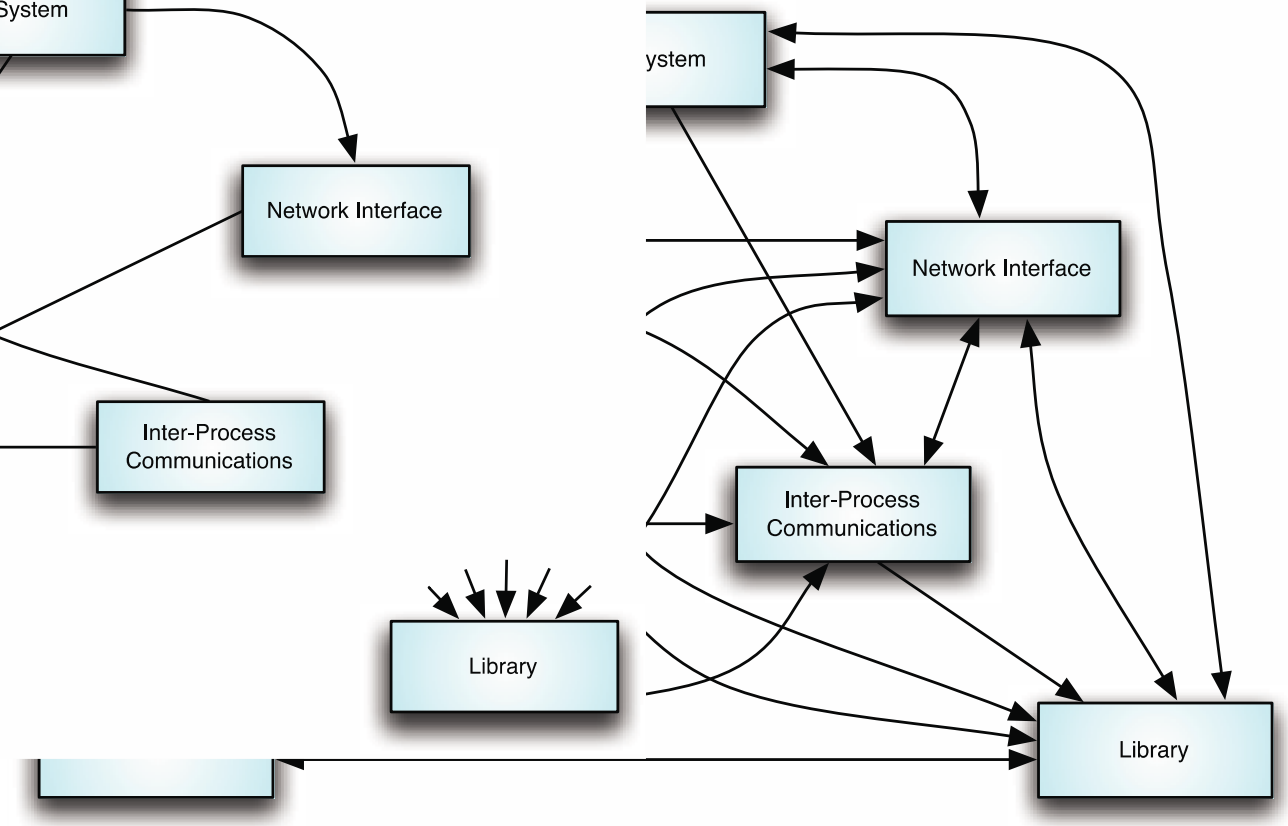


Complex and virtually incomprehensible!

What about "Real" Examples?

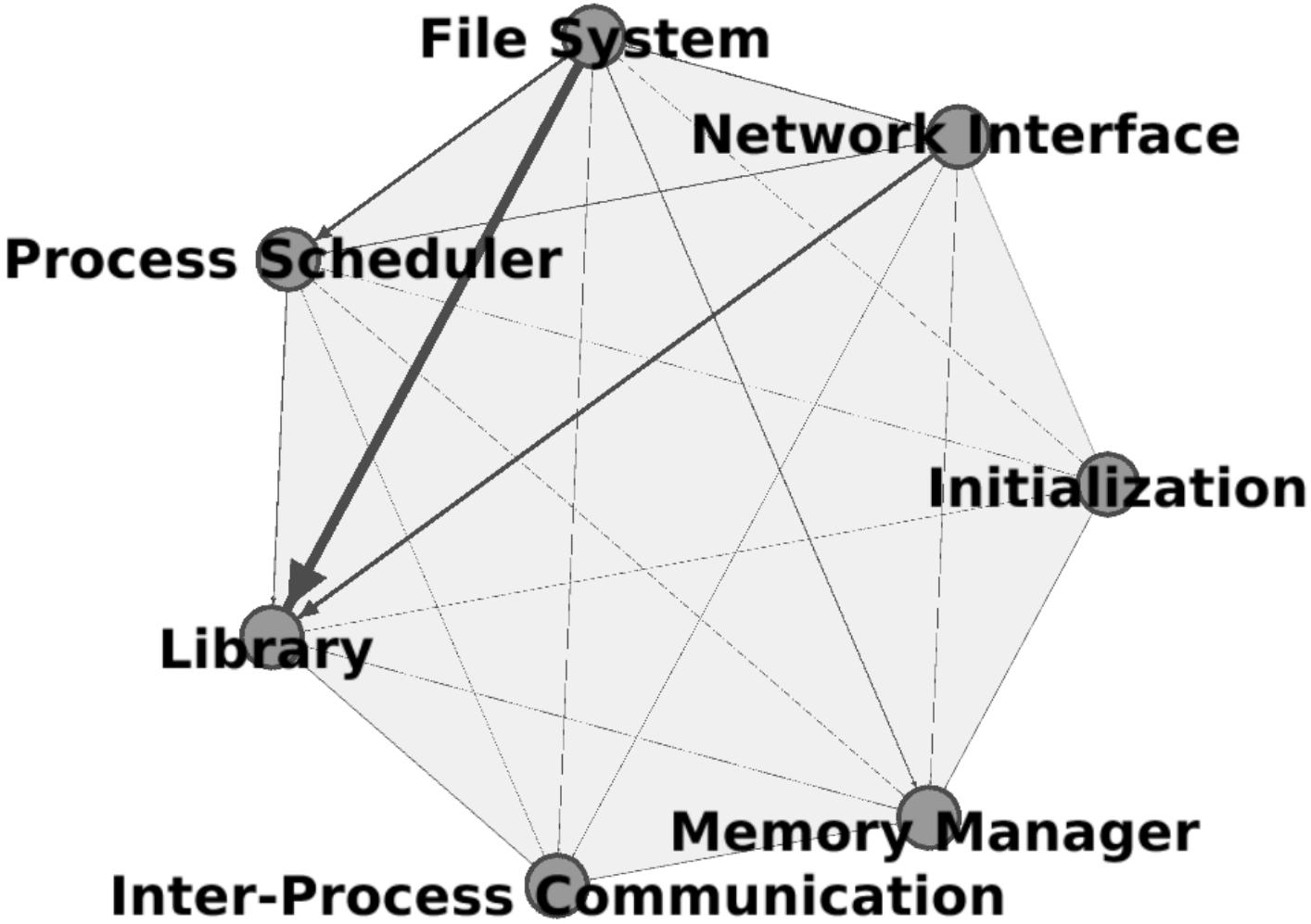
Linux – Prescriptive Architecture



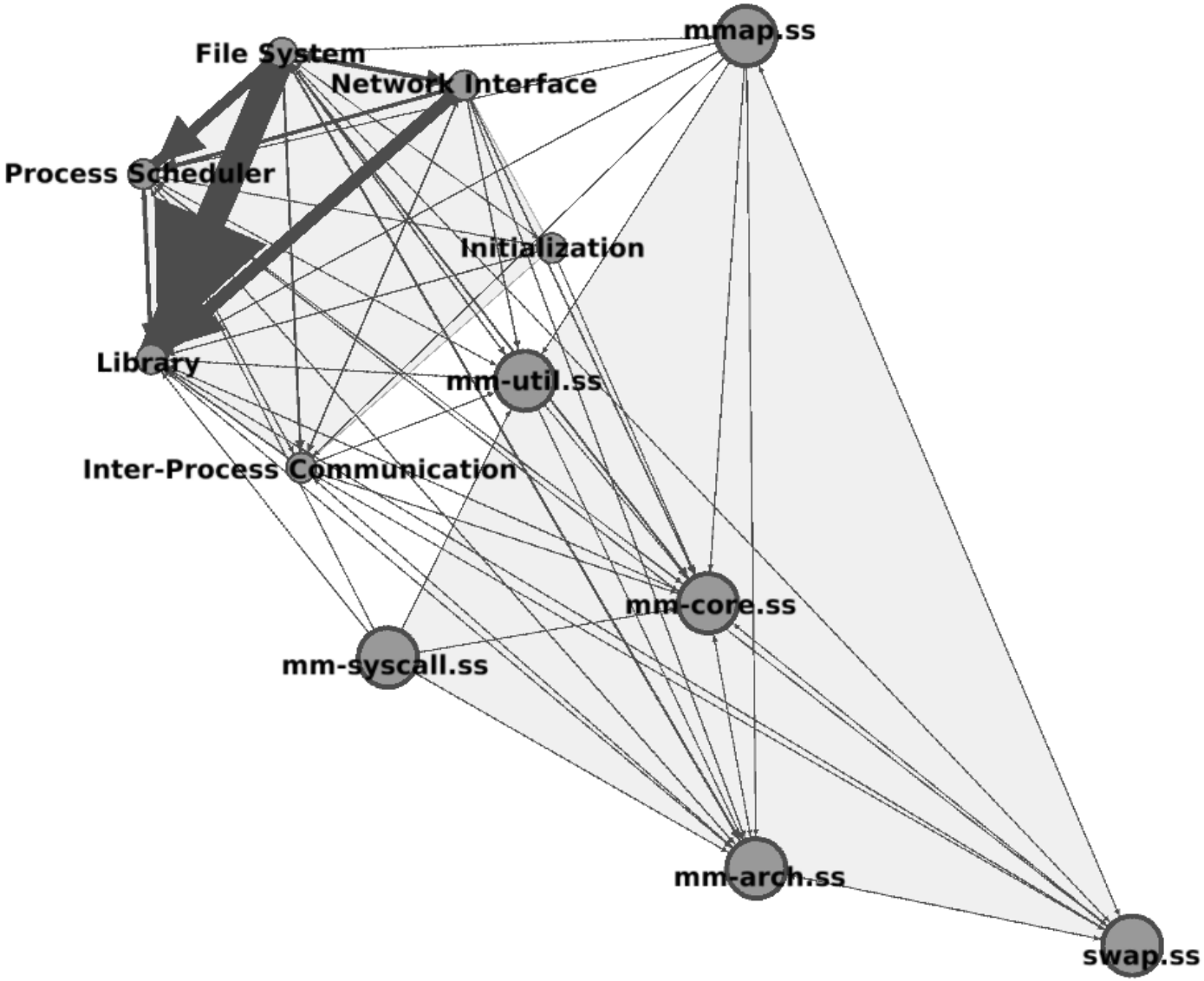


Linux – Descriptive Architecture

Top-Level Architecture – Another View

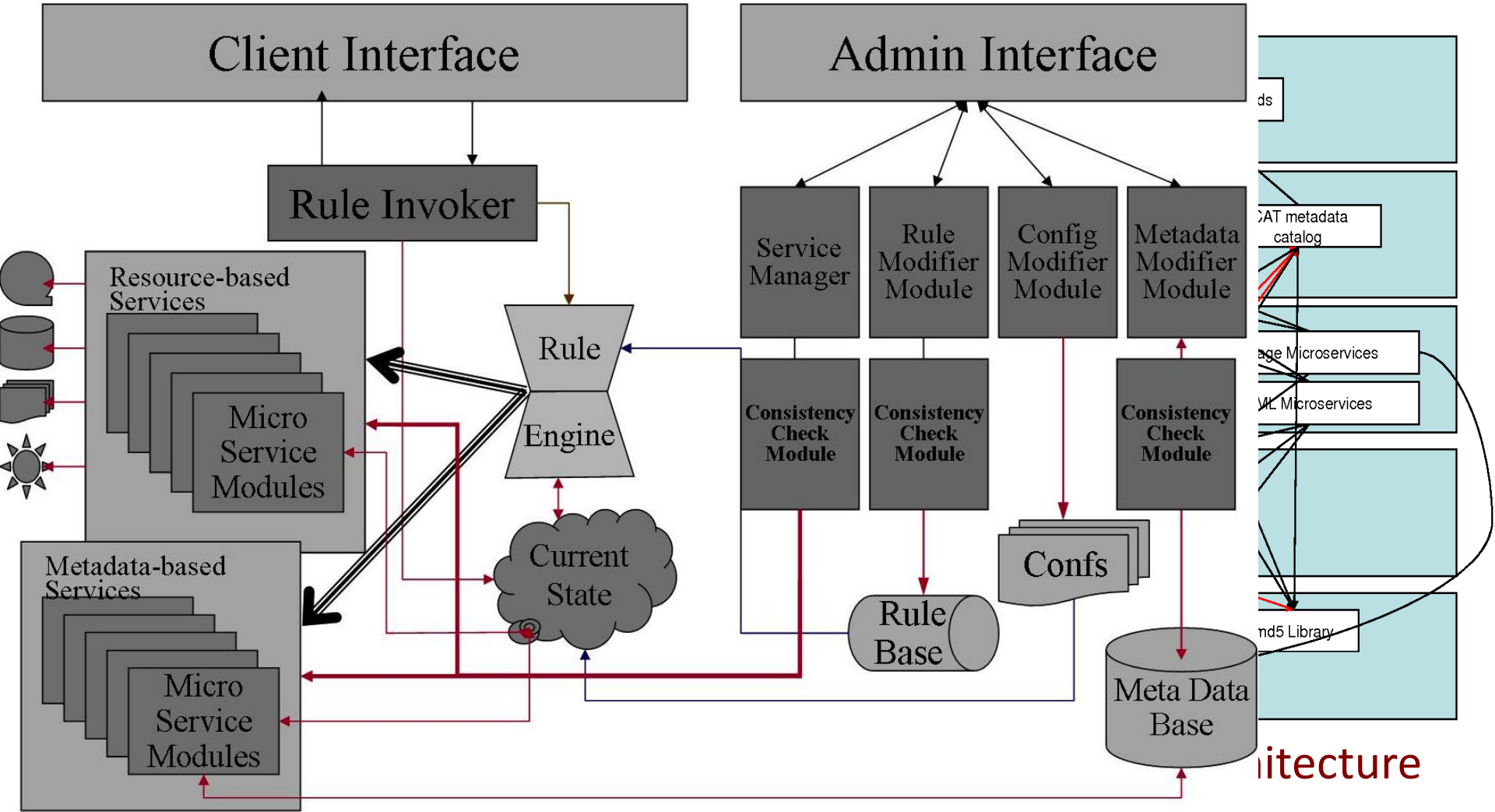


Memory Manager Subsystem



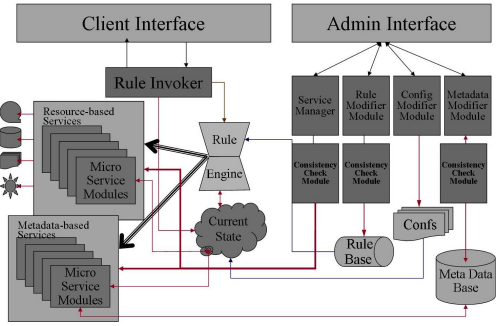
Another Example

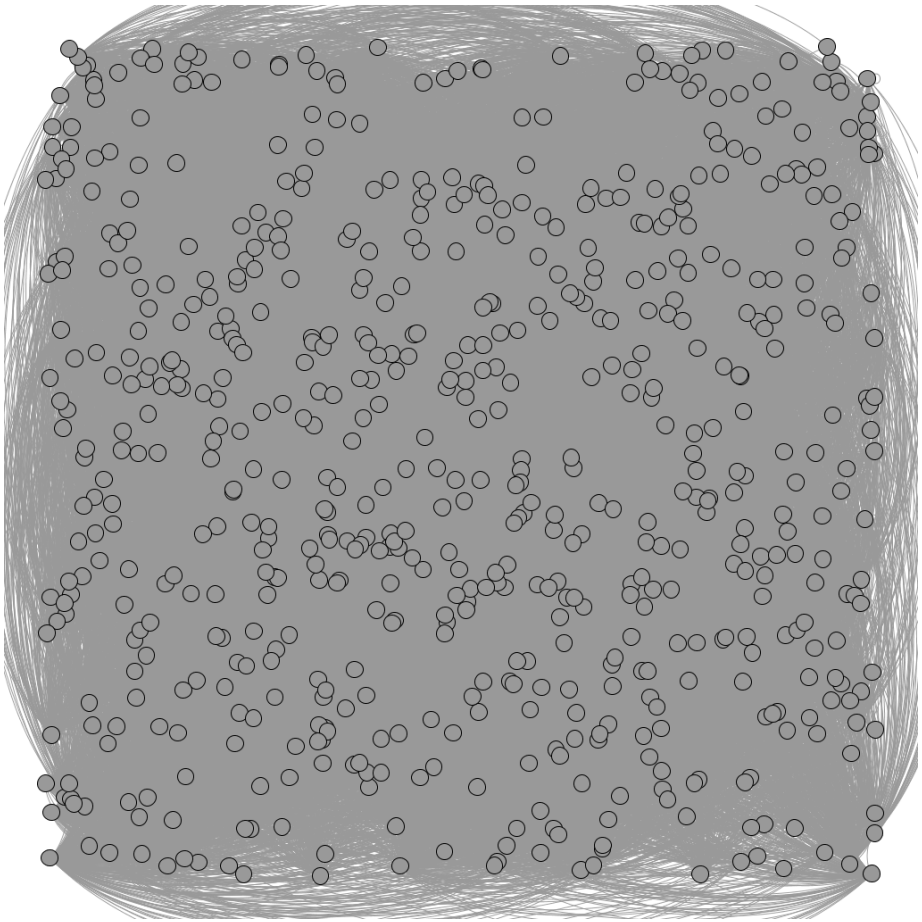
iRODS – Prescriptive Architecture



itecture

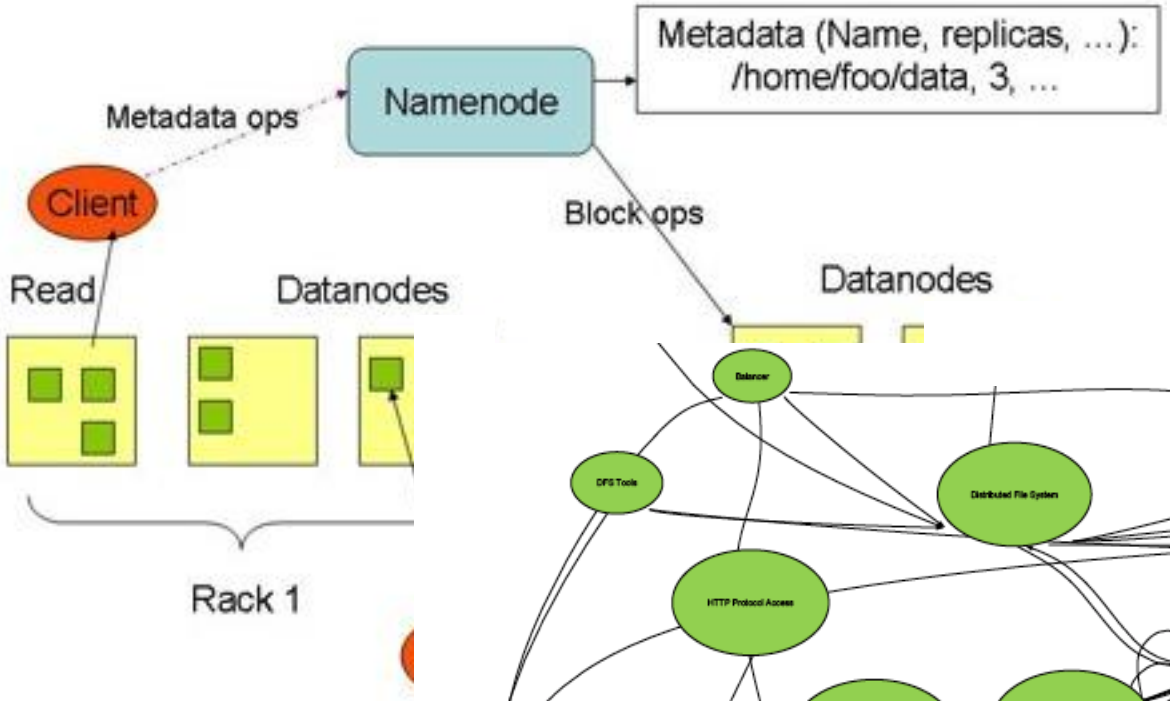
But What You Really See

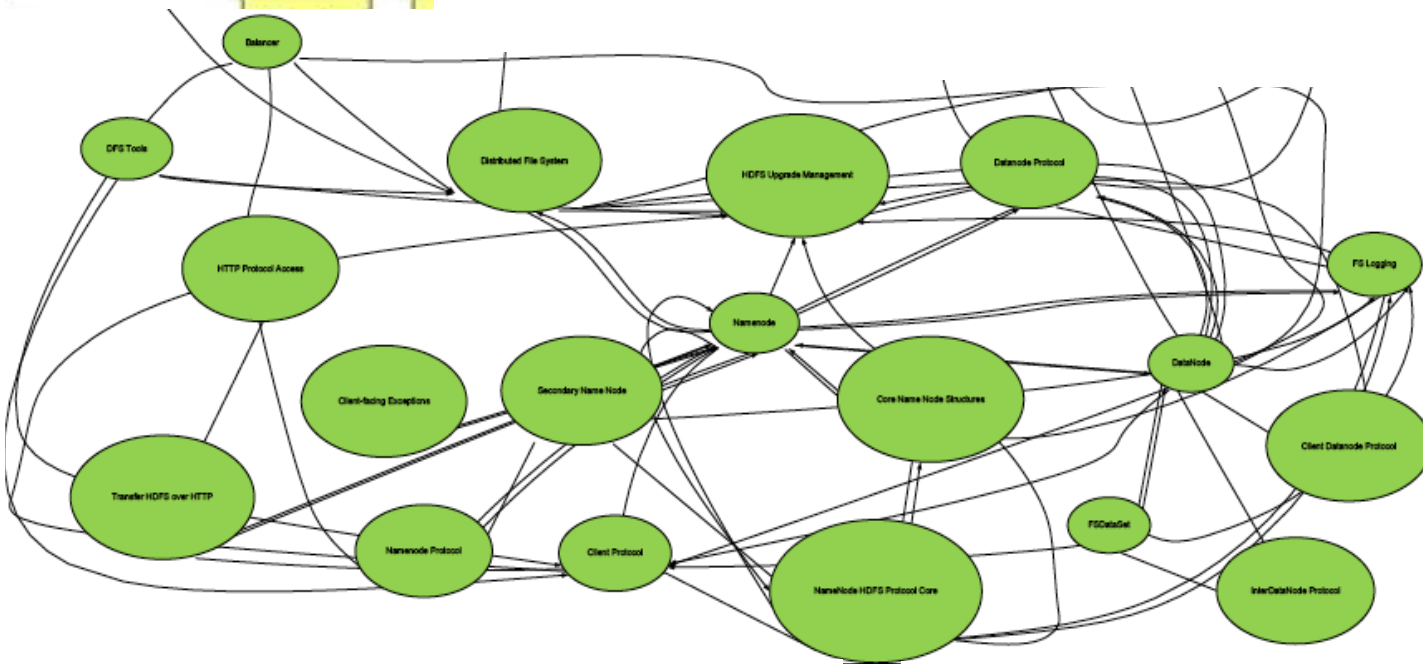




Yet Another Example

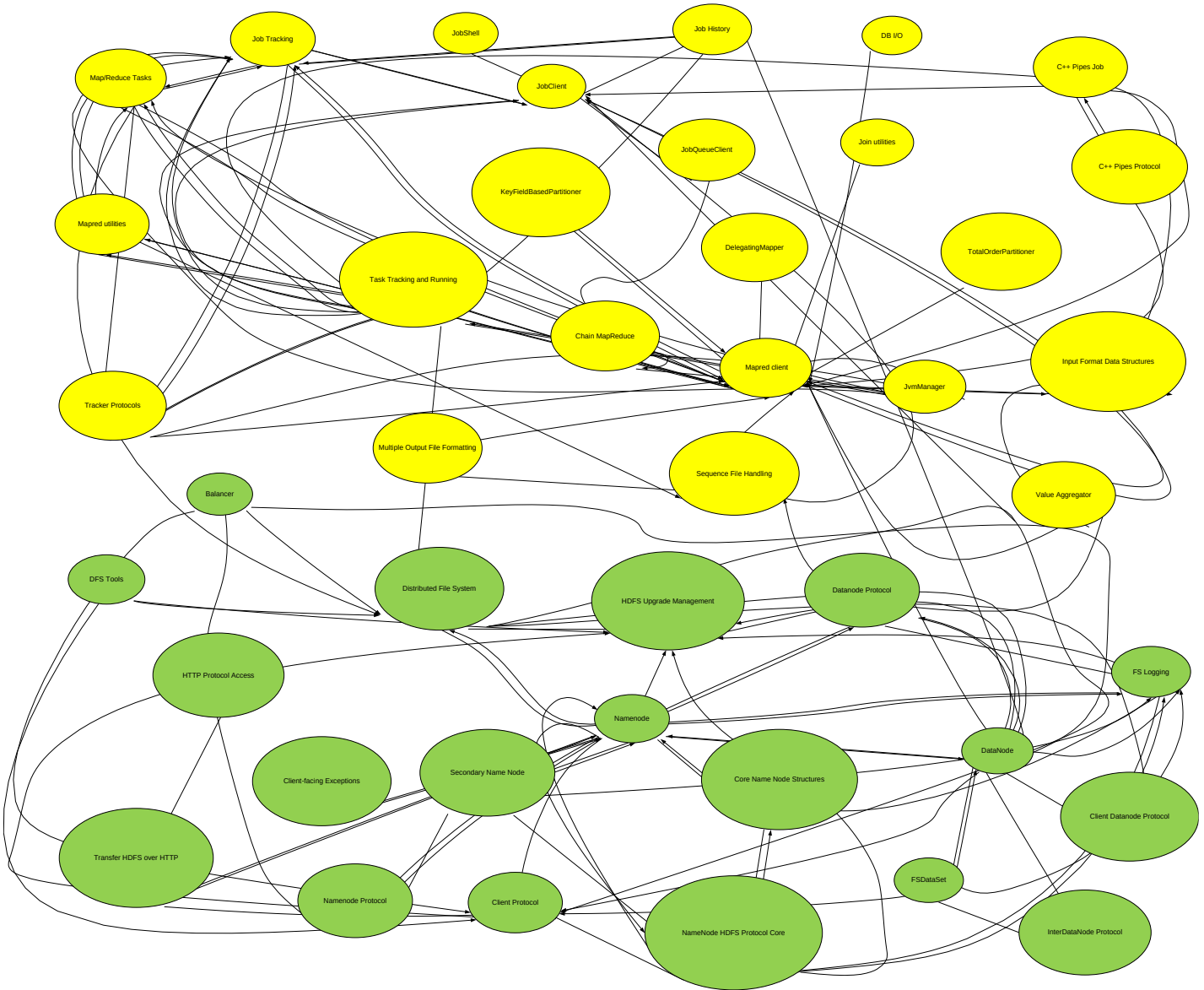
Hadoop Distributed File System – Prescriptive Architecture



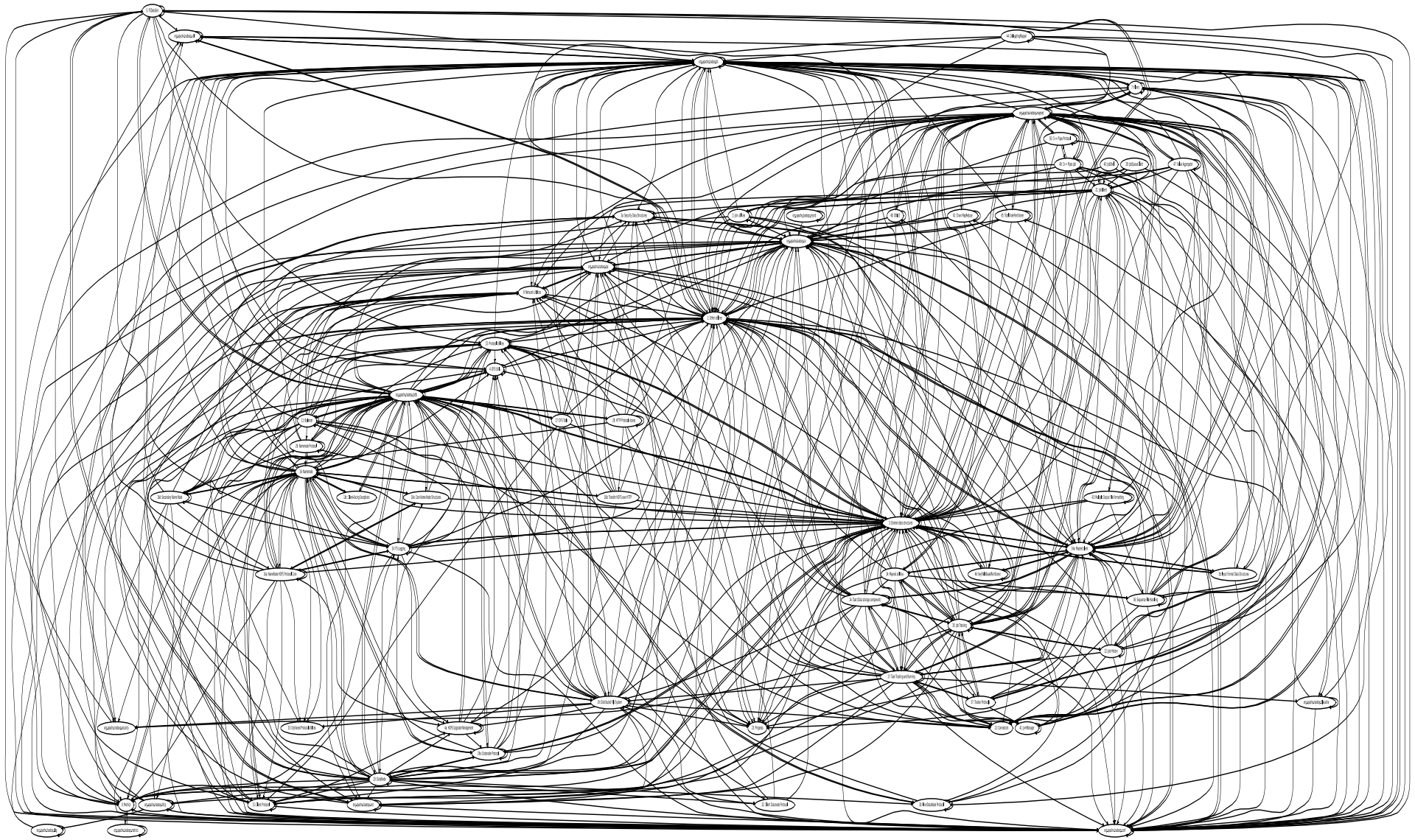


HDFS – Descriptive Architecture

Hadoop – HDFS + MapReduce

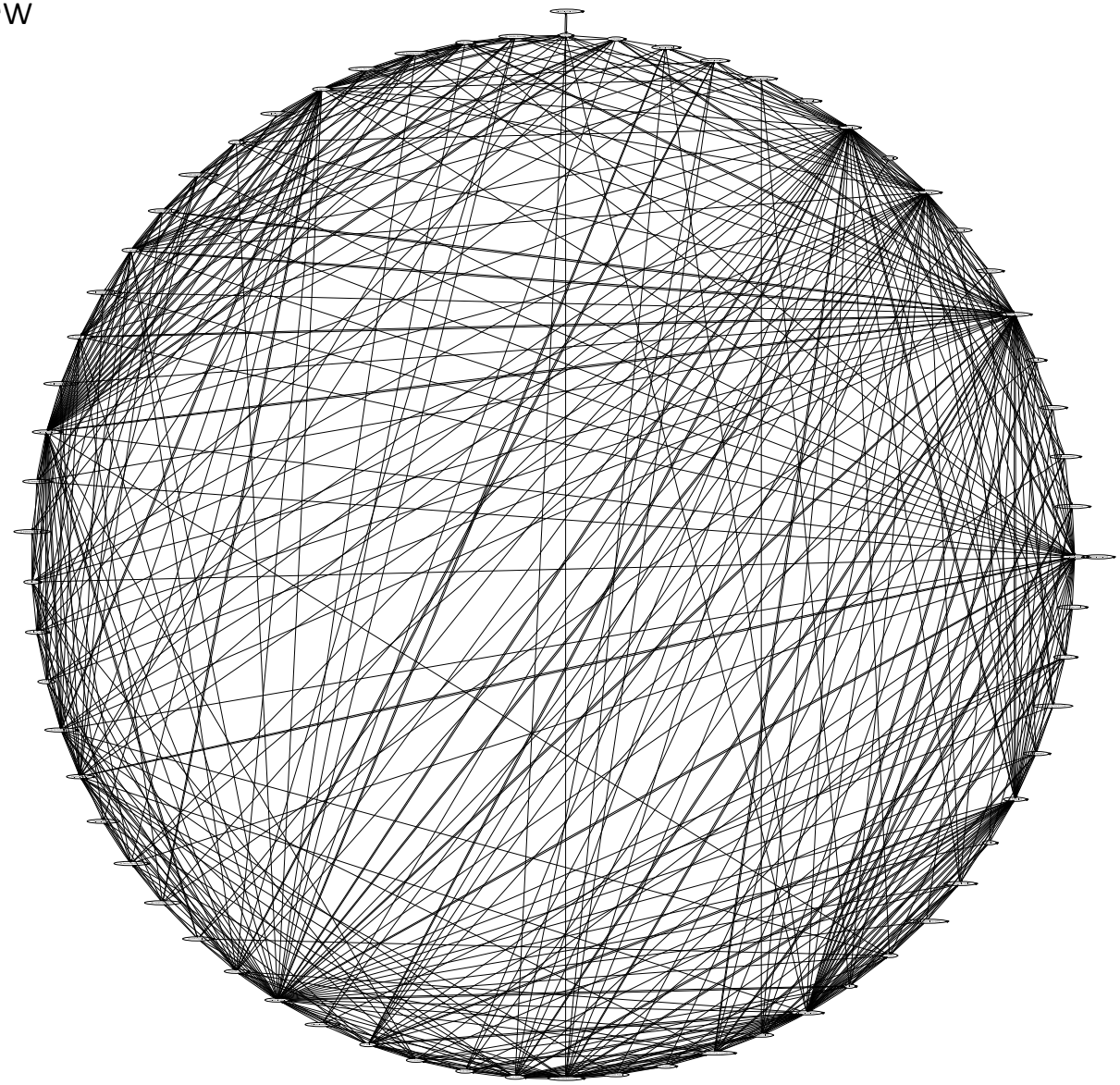


Hadoop – Complete Architecture



Hadoop – Complete Architecture

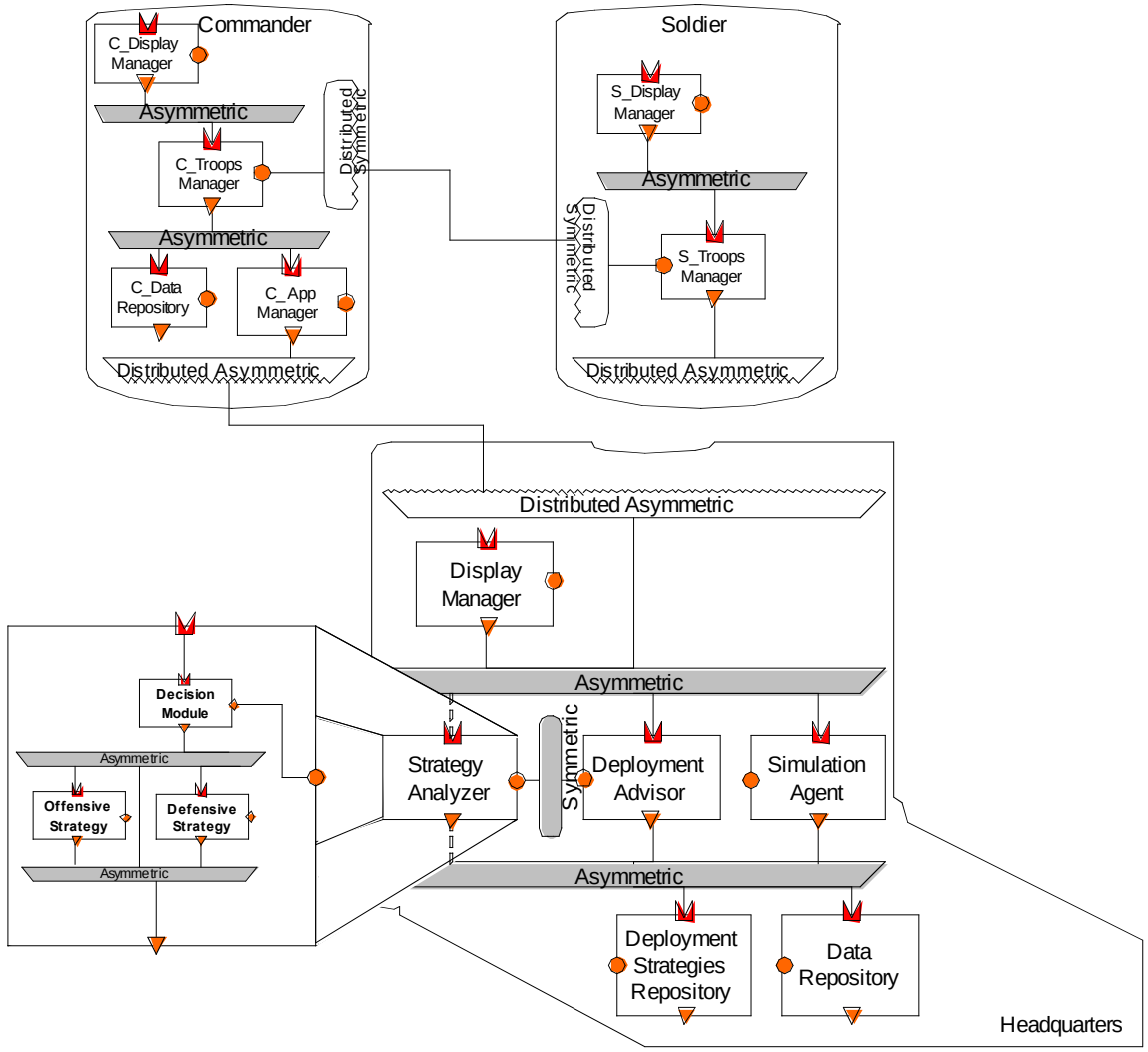
Another View



Deployment

- A software system cannot fulfill its purpose until it is *deployed*
  - Executable modules are physically placed on the hardware devices on which they are supposed to run
- The deployment view of an architecture can be critical in assessing whether the system will be able to satisfy its requirements
- Possible assessment dimensions
  - Available memory
  - Power consumption
  - Required network bandwidth

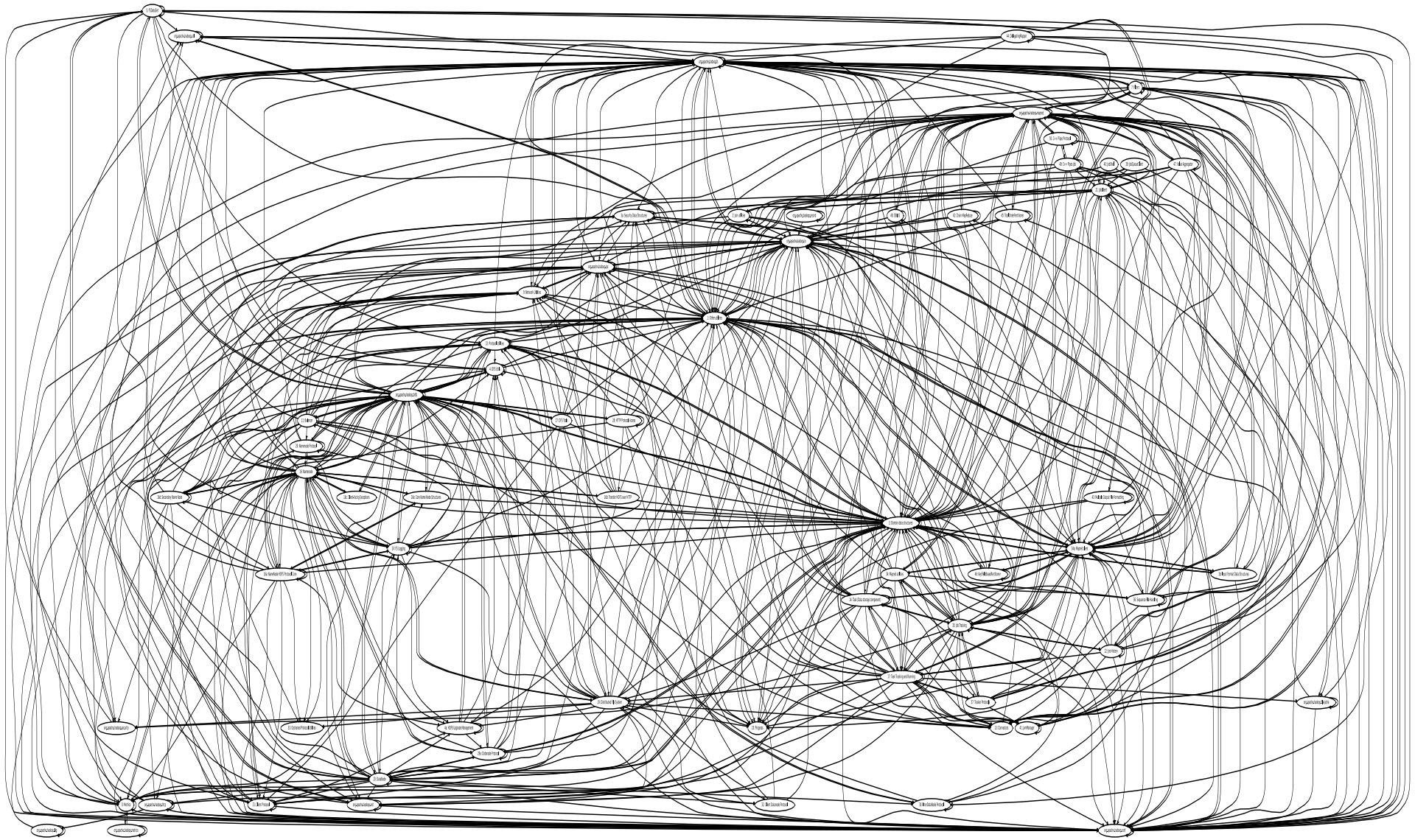
A System's Deployment Architectural Perspective



Course Syllabus – What else to do?

- Software engineering as a discipline; software process and product engineering; software development lifecycle models; agile software development;
- requirements engineering; software architecture; software design; Unified Modeling Language (UML); design patterns
- software construction; testing; verification and validation; software metrics; software project management;
- advanced software engineering topics such as reuse, reengineering and evolution.
- Android Development

Hadoop – Complete Architecture



What is Software Architecture? – This course's view Software Architecture's Elements

- A software system's architecture typically is not (and should not be) a uniform monolith
- A software system's architecture should be a composition and interplay of different elements
  - Processing
  - Data, also referred as information or state
  - Interaction

- 5Cs

- Context
- Components
- Connectors
- Configurations
- Constraints

Components

- Elements that encapsulate processing and data in a system's architecture are referred to as *software components*
- **Definition**
  - A *software component* is an architectural entity that
    - encapsulates a subset of the system's functionality and/or data
    - restricts access to that subset via an explicitly defined interface
    - has explicitly defined dependencies on its required execution context
- Components typically provide application-specific services

Connectors

- In complex systems *interaction* may become more important and challenging than the functionality of the individual components
- **Definition**
  - A *software connector* is an architectural building block tasked with effecting and regulating interactions among components
- In many software systems connectors are usually simple procedure calls or shared data accesses
  - Much more sophisticated and complex connectors are possible!
- Connectors typically provide application-independent interaction facilities

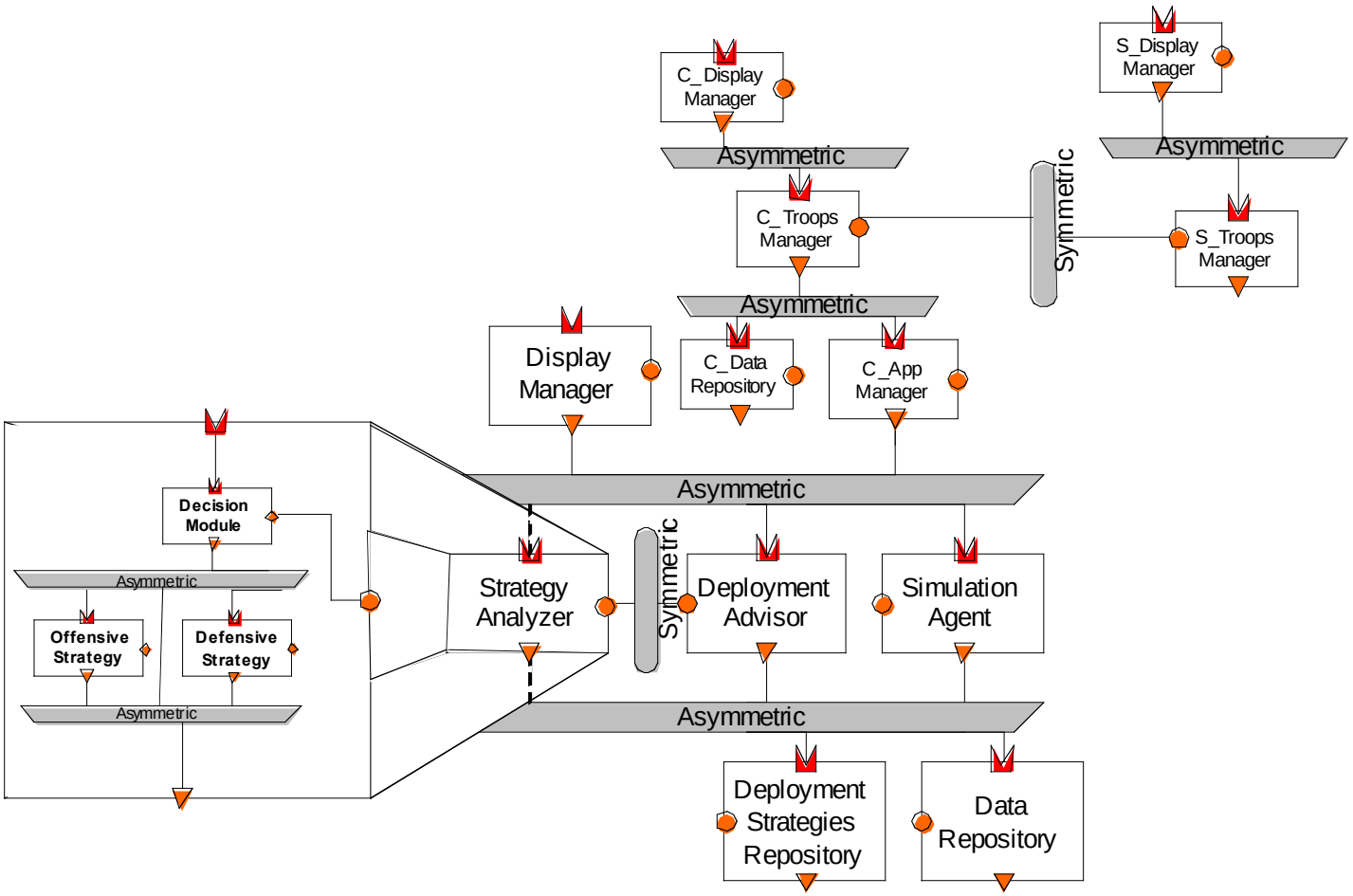
Examples of Connectors

- Procedure call connectors
- Shared memory connectors
- Message passing connectors
- Streaming connectors
- Distribution connectors
- Wrapper/adaptor connectors

Configurations

- Components and connectors are composed in a specific way in a given system's architecture to accomplish that system's objective
- **Definition**
  - An *architectural configuration*, or topology, is a set of specific associations between the components and connectors of a software system's architecture

An Example Configuration



Architecture Business Cycle

Architecture Business Cycle- Stakeholders

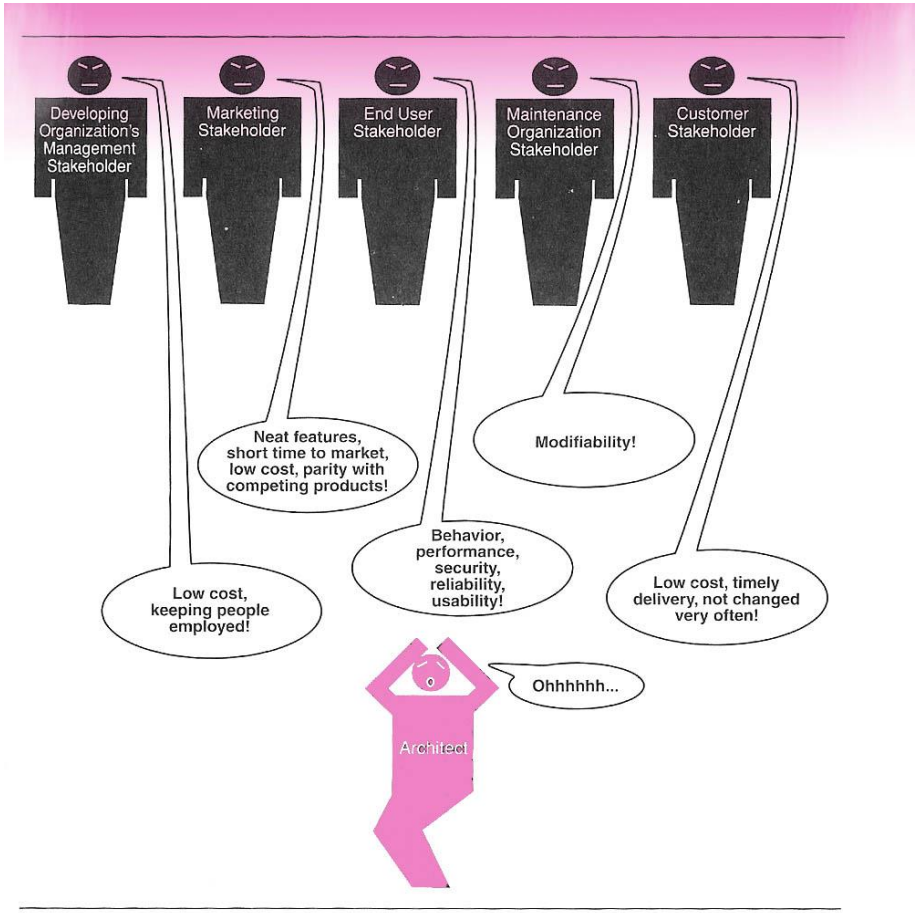
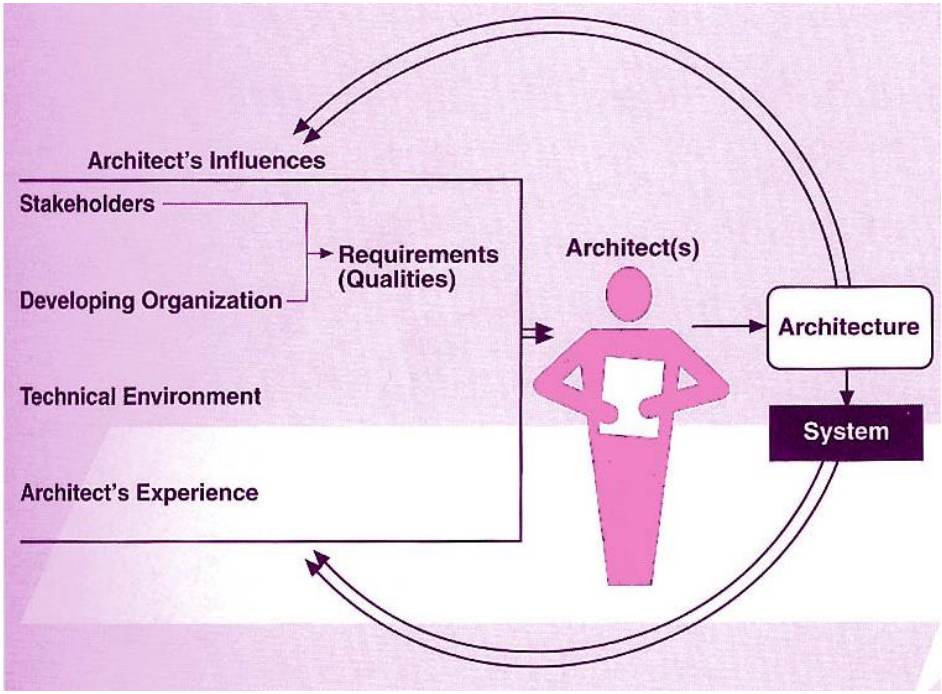


FIGURE 1.2 Influence of stakeholders on the architect

- Developing Organization Management
- Marketing
- End User
- Maintainenace Organization
- Customer

Architecture Business Cycle

- Stakeholders
- Developing Organization
- Technical Environment
- Architect's Experience



Architecture Activities

- Creating business case
- Understanding requirements
- Creating or selecting architecture
- Documenting and communicating architecture
- Analyzing architecture
- Implementing system
- Ensuring conformance of implementation

Architecture Process Advice (1/2)

1. Architecture should be product of a single architect of small group with identified leader
2. Architect should have functional requirements and a prioritized list of quality attributes
3. Architecture should be well-documented with at least one static and one dynamic view

Architecture Process Advice (2/2)

4. Architecture should be circulated to stakeholders, who are active in review
5. Architecture should be analyzed (quantitatively and quality) before it is too late.
6. System should be developed incrementally from an initial skeleton that includes major communication paths
7. Architecture should result in a small number of specific resource contention areas

"Good" Architecture Rules of Thumb (1/2)

1. Use information hiding to hide computing infrastructure
2. Each module should protect its secrets with a good interface
3. Use well-known architecture tactics to achieve quality attributes
4. Minimize and isolate dependence on a particular version of a commercial product or tool.

"Good" Architecture Rules of Thumb (2/2)

4. Separate producer modules from consumer modules.
5. For parallel-processing, use well-defined processes or tasks.
7. Assignment of tasks or processes to processors should be easily changeable (even at runtime)
8. Use a small number of simple interaction patterns

Documenting the Architecture

Some of the material in these slides is taken from Software Architecture in Practice, 2nd edition by Bass, Clements and Kazman.

The ADL slides are based on a presentation by T. Cook of Microelectronics and Computer Technology Corporation

Different Needs of Stakeholders

- Different stakeholder groups have different needs
- Should provide different views to satisfy those needs
- Often create one document with different roadmaps for different groups

	Module Views				C&C Views	Allocation Views	
Stakeholder	Decomposition	Uses	Class	Layer	Various	Deployment	Implementation
Project Manager	s	s		s		d	
Member of Development Team	d	d	d	d	d	s	s
Testers and Integrators		d	d		s	s	s
Maintainers	d	d	d	d	d	s	s
Product Line Application Builder		d	s	o	s	s	s
Customer					s	o	
End User					s	s	
Analyst	d	d	s	d	s	d	
Infrastructure Support	s	s		s		s	d
New Stakeholder	x	x	x	x	x	x	x
Current and Future Architect	d	d	d	d	d	d	s

Key: d = detailed information, s = some details, o = overview information, x = anything

Choosing Relevant views

- Produce a candidate view list
- Combine views
- Prioritize

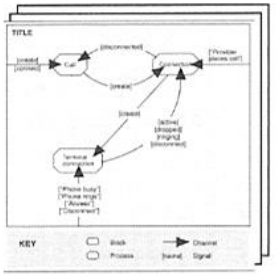
Suggested View Documentation Template

1. Primary presentation – elements and their relationships
2. Element catalog -- explains the picture
3. Context diagram -- how the system relates to its environment
4. Variability guide – how to exercise any variation points
5. Architecture background – why the design reflected in the view came to be
6. Glossary of terms used
7. Other information

Pictorially:

Views

Section 1. Primary Presentation of the View



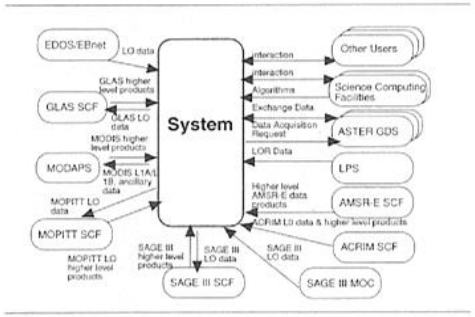
OR

Textual version of the primary presentation

Section 2. Element Catalog

Section 2.A Elements and their properties
Section 2.B Relations and their properties
Section 2.C Element interfaces
Section 2.D Element behavior

Section 3. Context Diagram



Section 4. Variability Guide

Section 5. Architecture Background

Section 5.A Design rationale
Section 5.B Analysis of results
Section 5.C Assumptions

Section 6. Glossary of Terms

Section 7. Other Information

Documenting Interfaces

1. Interface identity - unique name
2. Resources provided
3. Locally defined data types - if used
4. Exception definitions - including handling
5. Variability provided - for product lines
6. Quality attribute characteristics - what is provided?
7. Element requirements
8. Rationale and design issues - why these choices
9. Usage guide - protocols

Architecture Description Languages

ADL -definition

"form of expression used for the description of architectures"

- ISO/IEC 42010

ADLs - Positives

- Provide a formal way of representing architecture
- Intended to be human and machine readable
- Support describing a system at a higher level than previously possible
- Permit analysis of architectures – completeness, consistency, ambiguity, and performance
- Can support automatic generation of software systems

ADLs - Negatives

- No universal agreement on what ADLs should represent, particularly as regards the behavior of the architecture
- Representations currently in use are relatively difficult to parse and are not supported by commercial tools
- Most ADL work today has been undertaken with academic rather than commercial goals in mind
- Most ADLs tend to be very vertically optimized toward a particular kind of analysis

Candidate ADLs

- Leading candidates
  - ACME (CMU/USC)
  - Rapide (Stanford)
  - Wright (CMU)
  - Unicon (CMU)
- Secondary candidates
  - Aesop (CMU)
  - MetaH (Honeywell)
  - C2 SADL (UCI)
  - SADL (SRI)