USC Viterbi School of Engineering

USC

Architectural Degradation

The Plague of Maturing Software Systems

Nenad Medvidović

Center for Systems and Software Engineering
Computer Science Department
Viterbi School of Engineering
University of Southern California

Motivation

If you

design software,

implement software,

test software,

maintain software,

manage software projects,

or just use software,

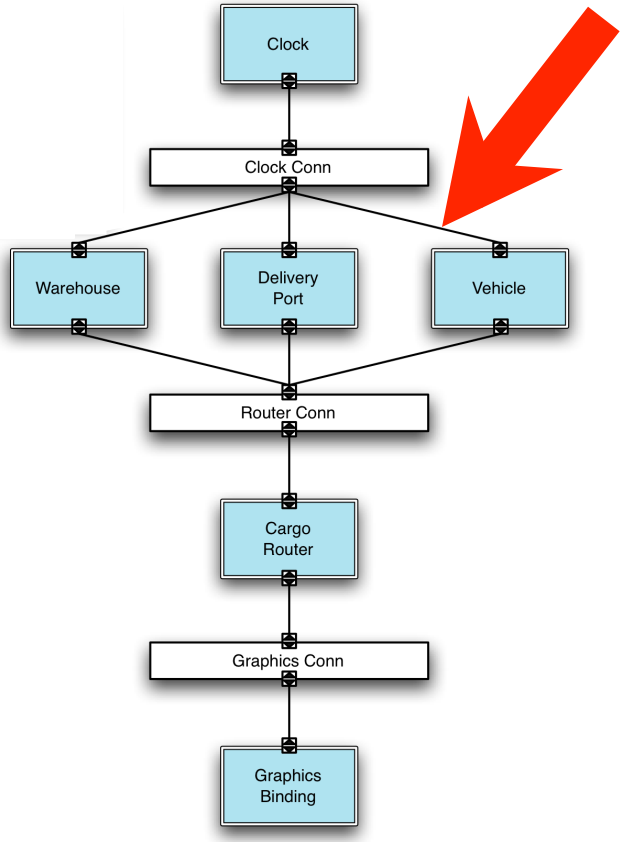
you may be handling a ticking bomb.

USC

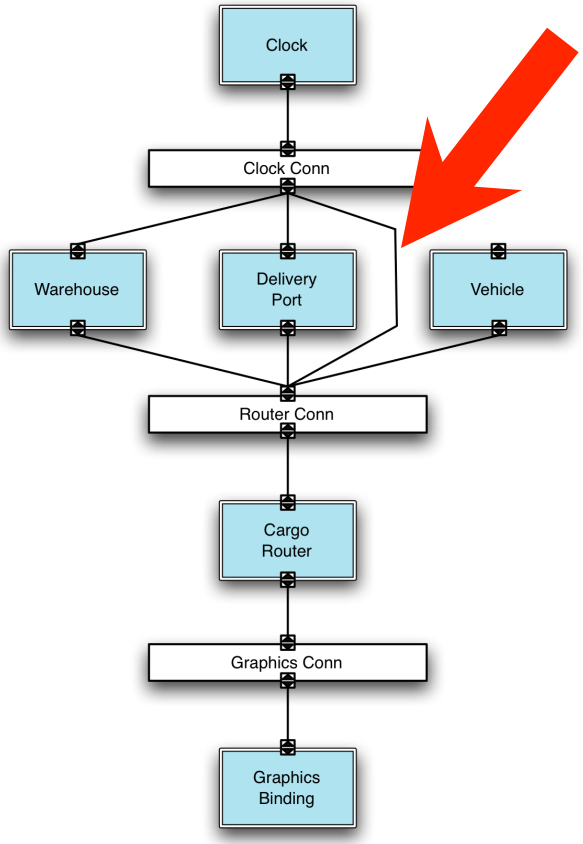
Some Terminology

- **Software architecture**
  - Set of principal design decisions *P* about a software system
- **Prescriptive architecture**
  - Set of architectural design decisions *P* made at time *t* that reflect architects' intent
  - "as designed"
- **Descriptive architecture**
  - Set of artifacts *A* that realize the design decisions *P*
  - "as implemented"

Two Architectures Side-by-Side



Prescriptive Architecture



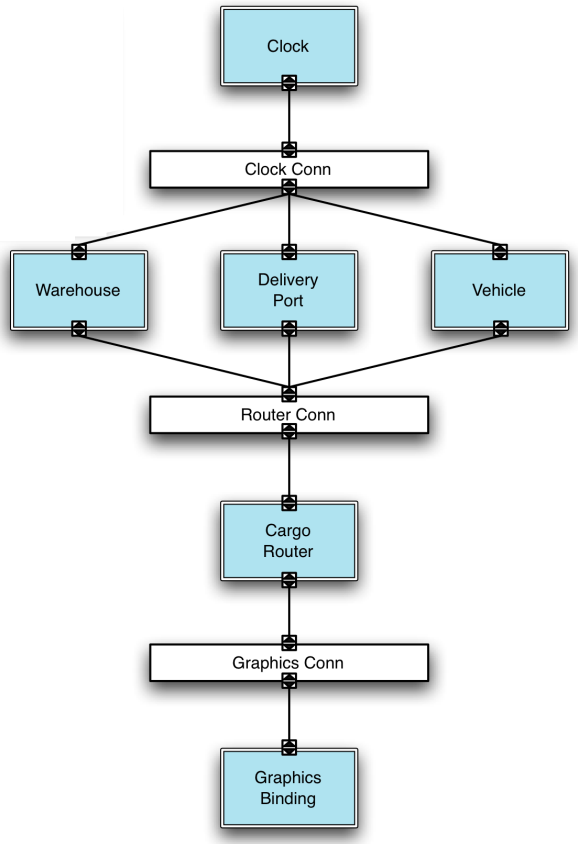
Descriptive Architecture

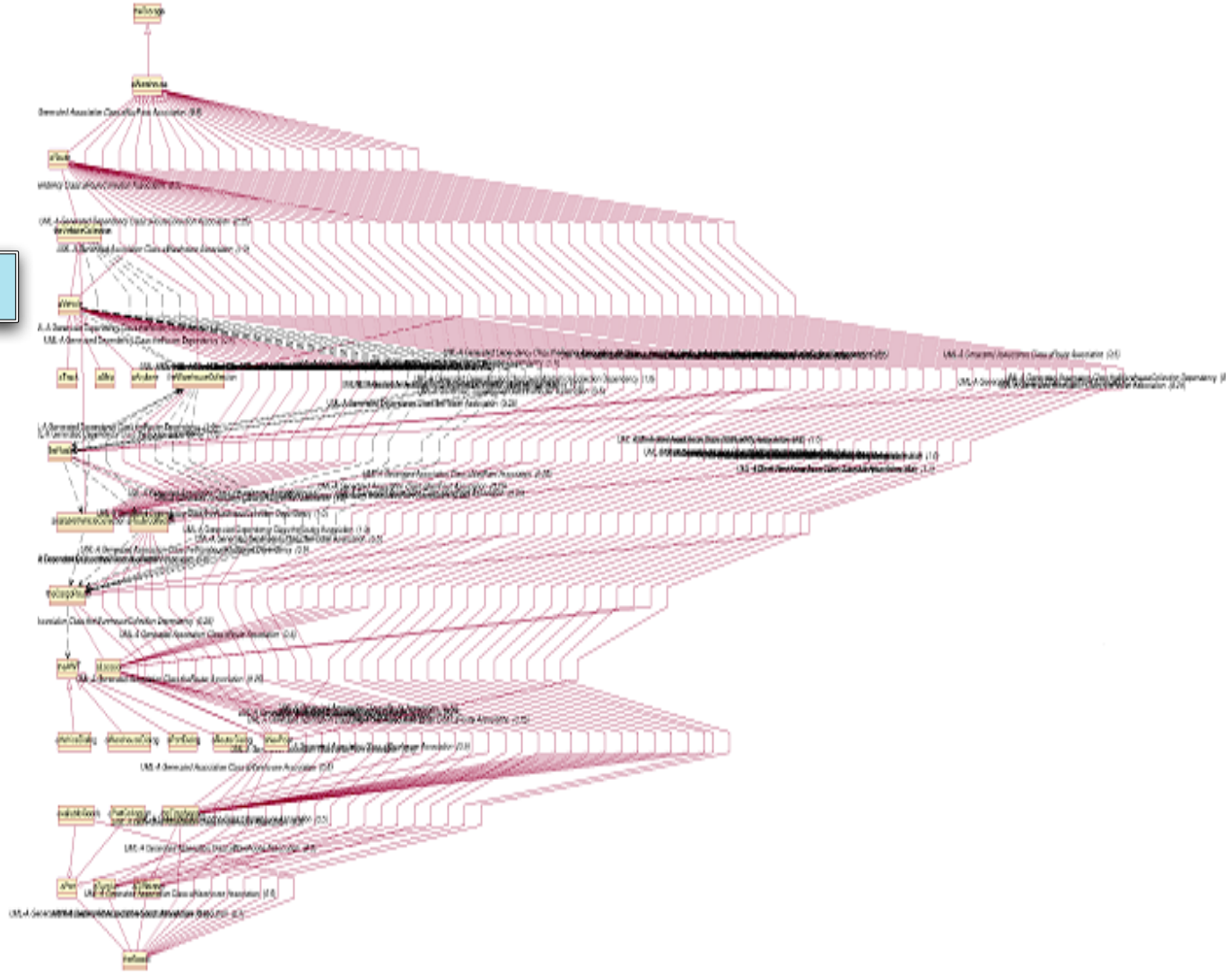
What Happened?

- **Architectural drift**
  - Introduction of design decisions into a system's descriptive architecture that are not included in, encompassed by, or implied by the prescriptive architecture
- **Architectural erosion**
  - Introduction of design decisions into a system's descriptive architecture that violate its prescriptive architecture

USC

Why Do We Care?

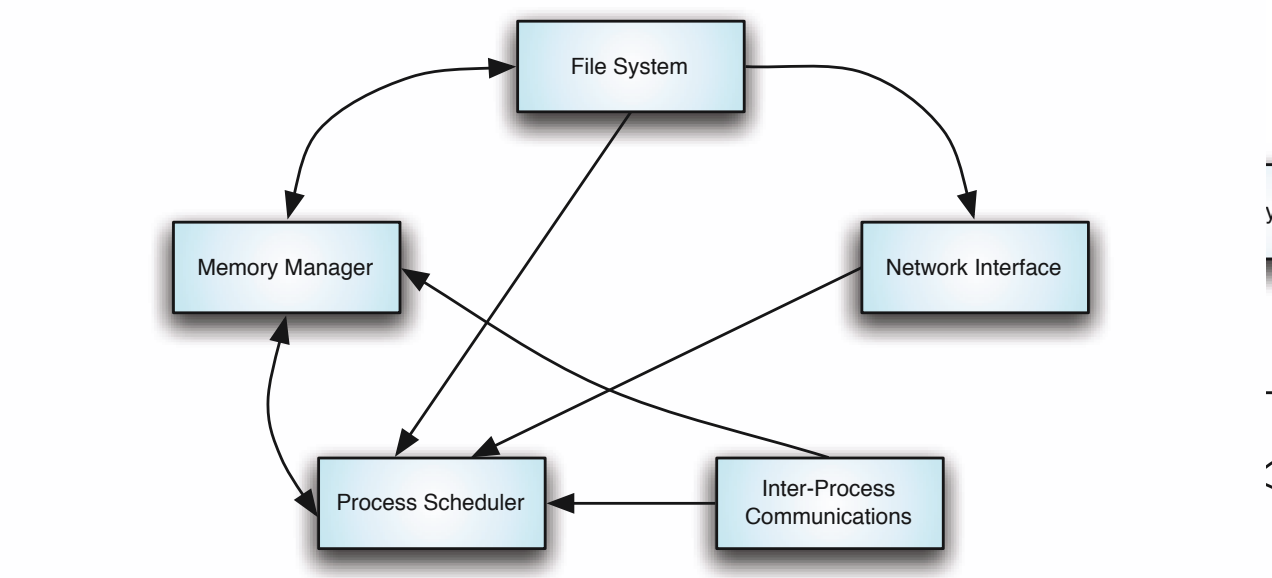




USC

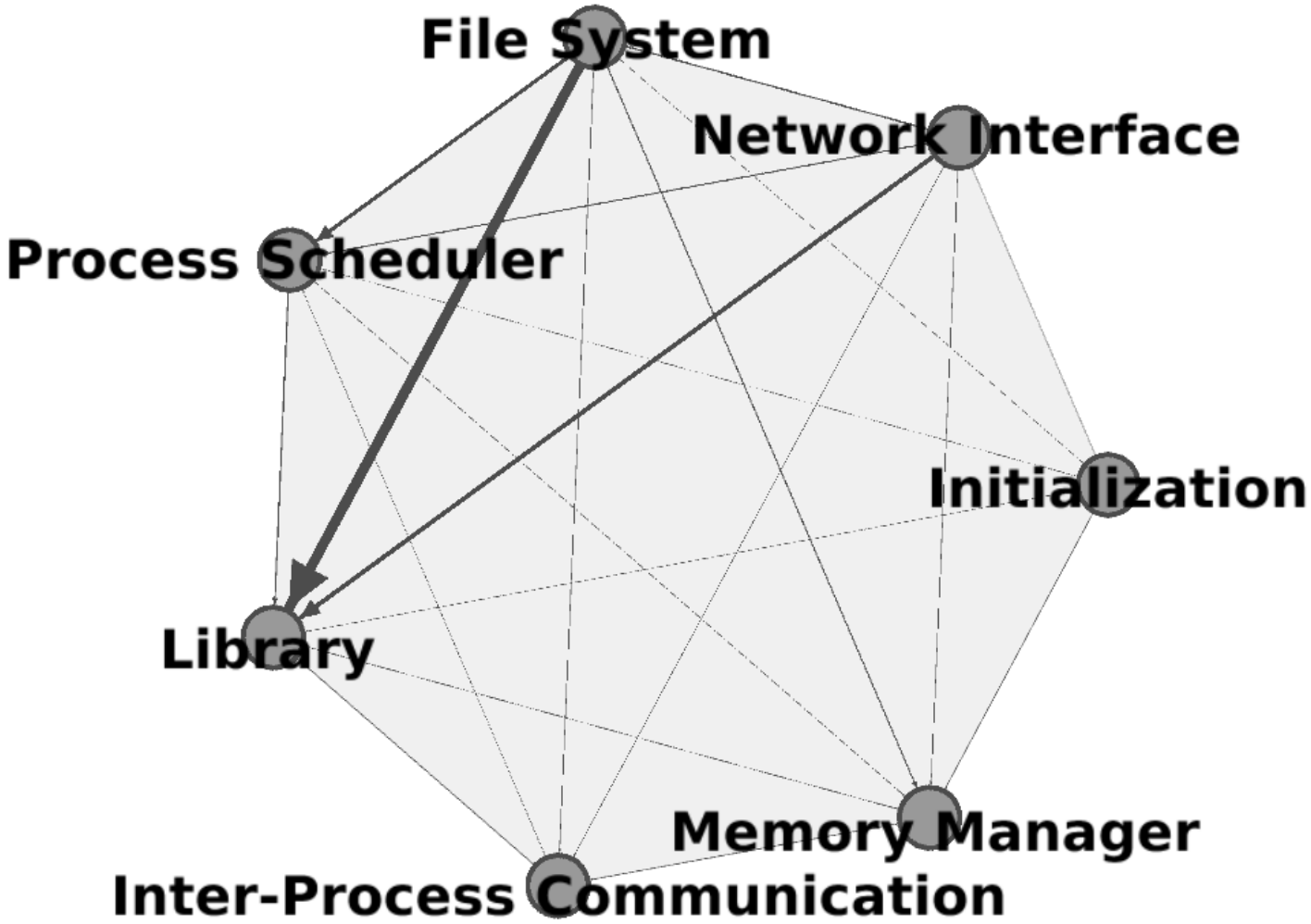
What about "Real" Examples?

Linux – Prescriptive Architecture



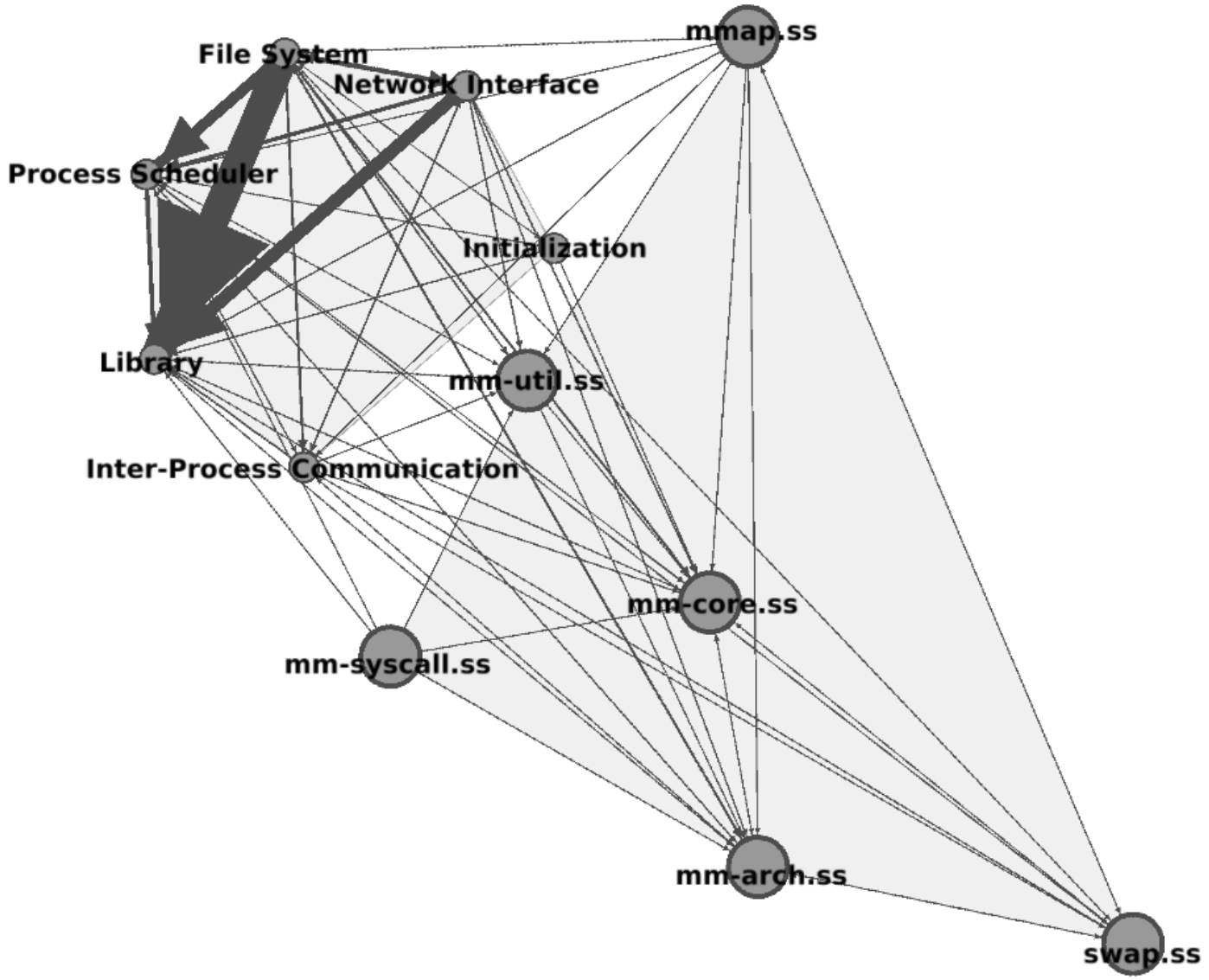
Linux – Descriptive Architecture

Top-Level Architecture – Another View



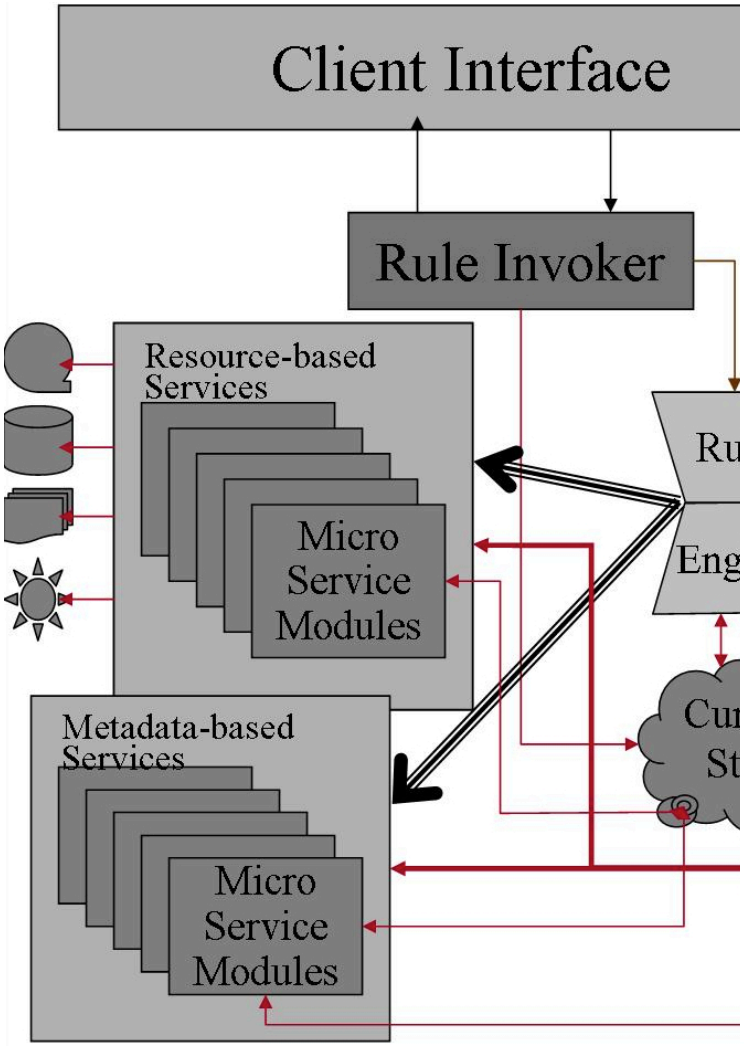
USC

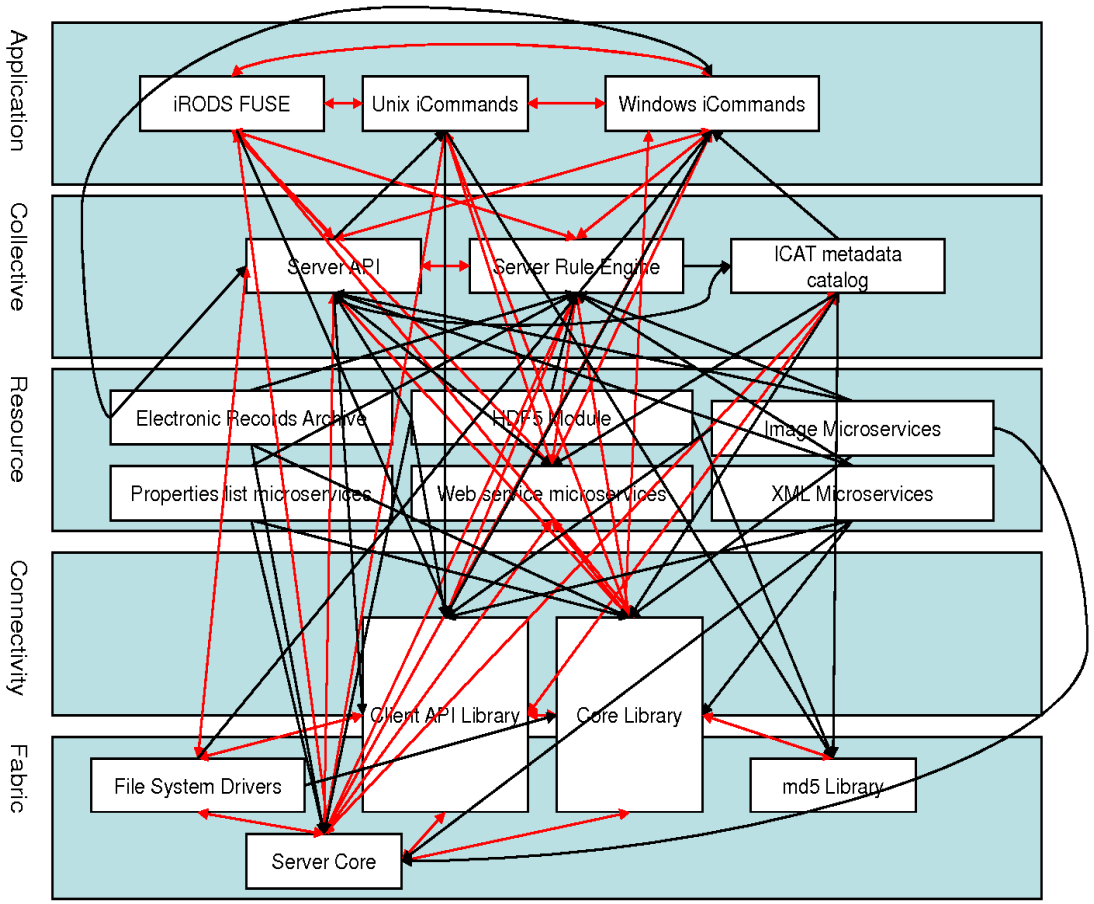
Memory Manager Subsystem



Another Example

iRODS – Prescriptive Architecture

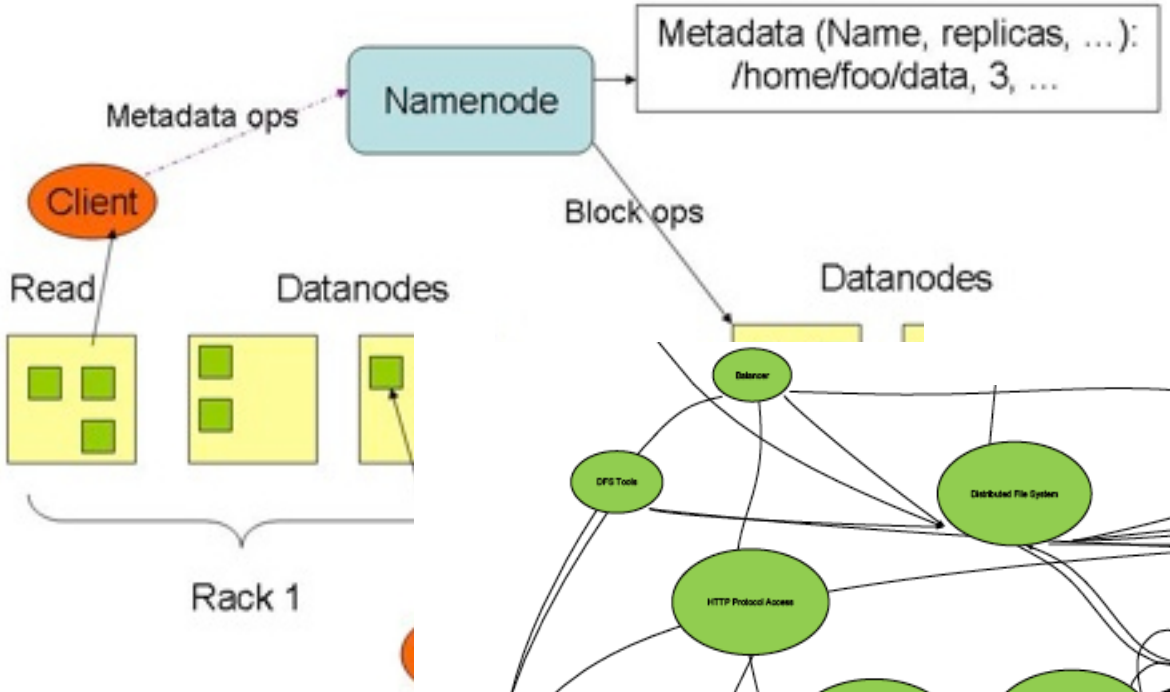


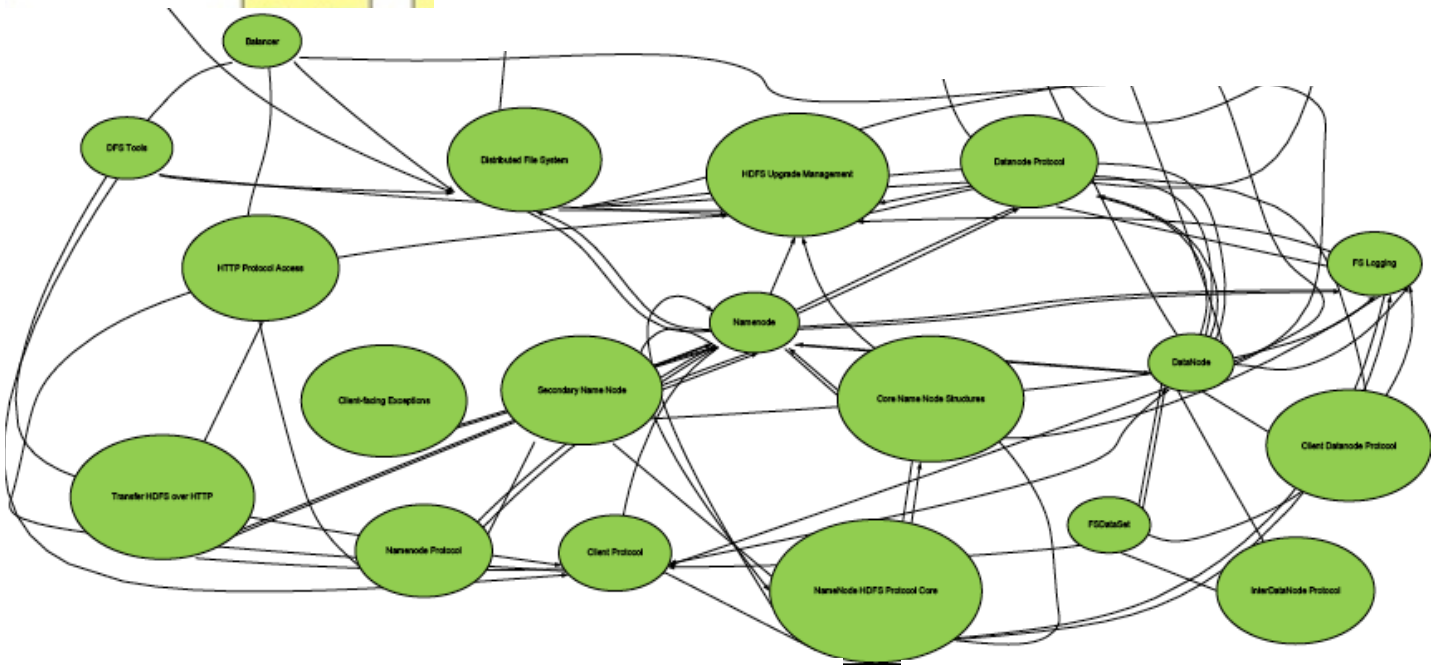


iRODS – Descriptive Architecture

And Another

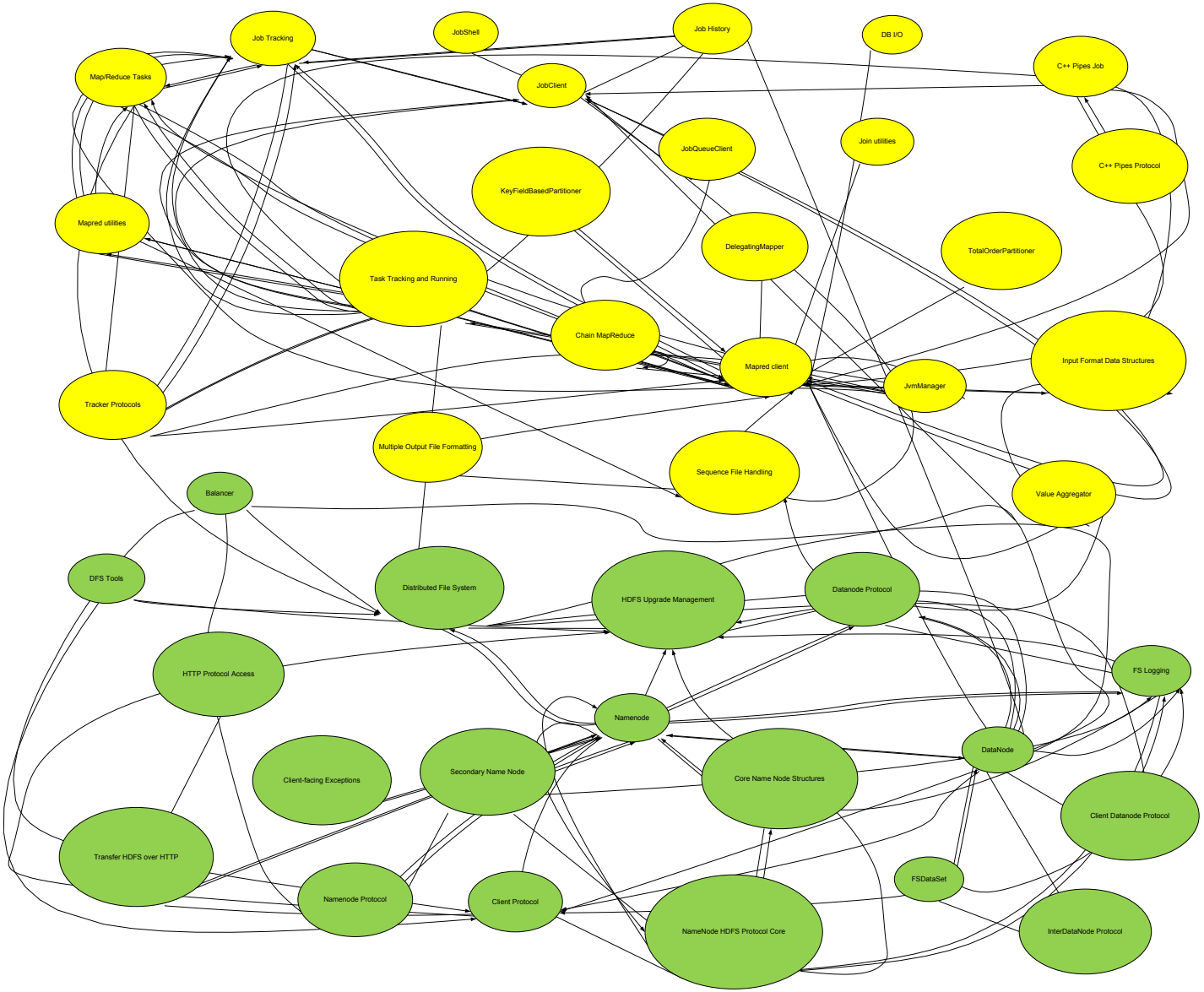
Hadoop Distributed File System – Prescriptive Architecture





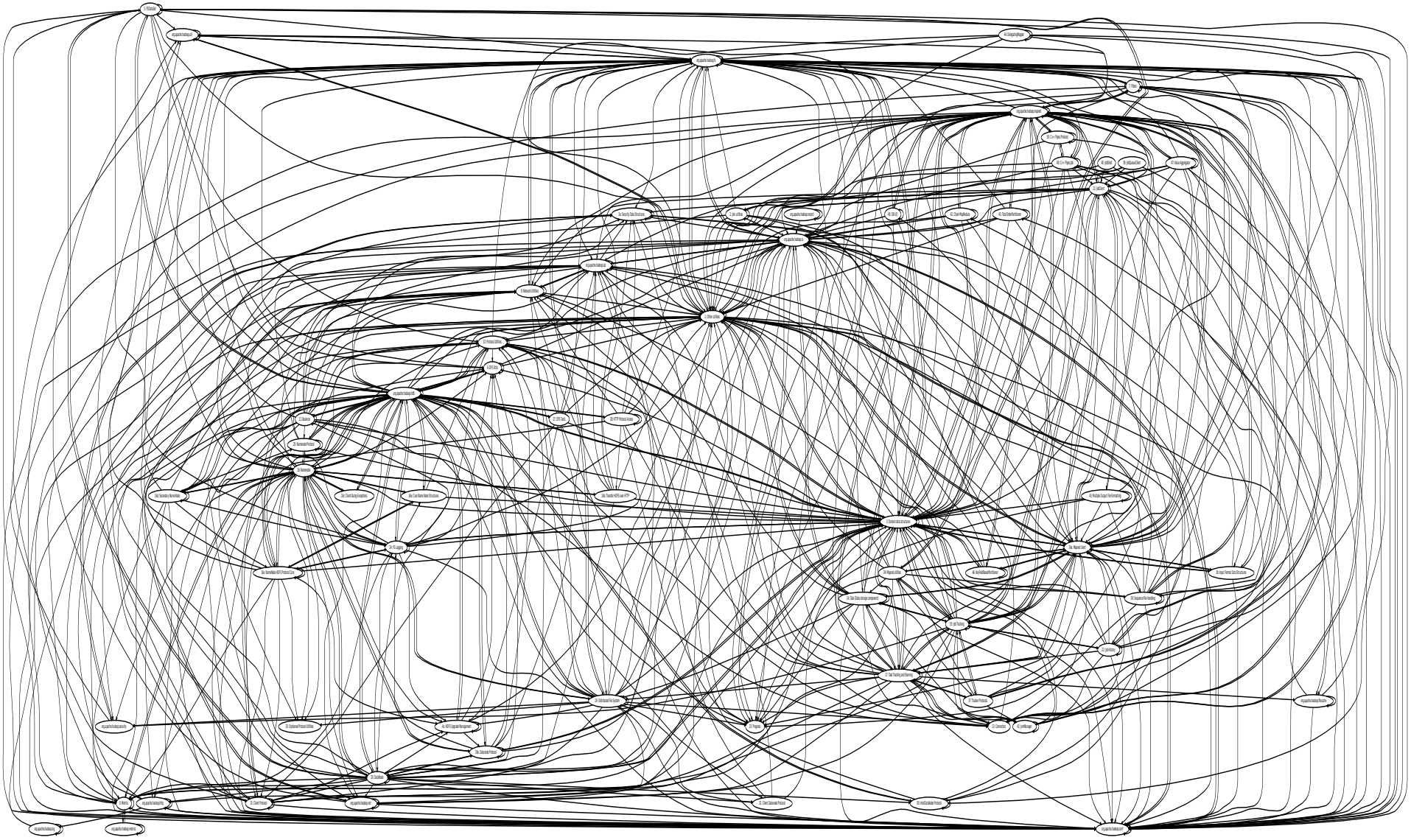
HDFS – Descriptive Architecture

Hadoop – HDFS + MapReduce

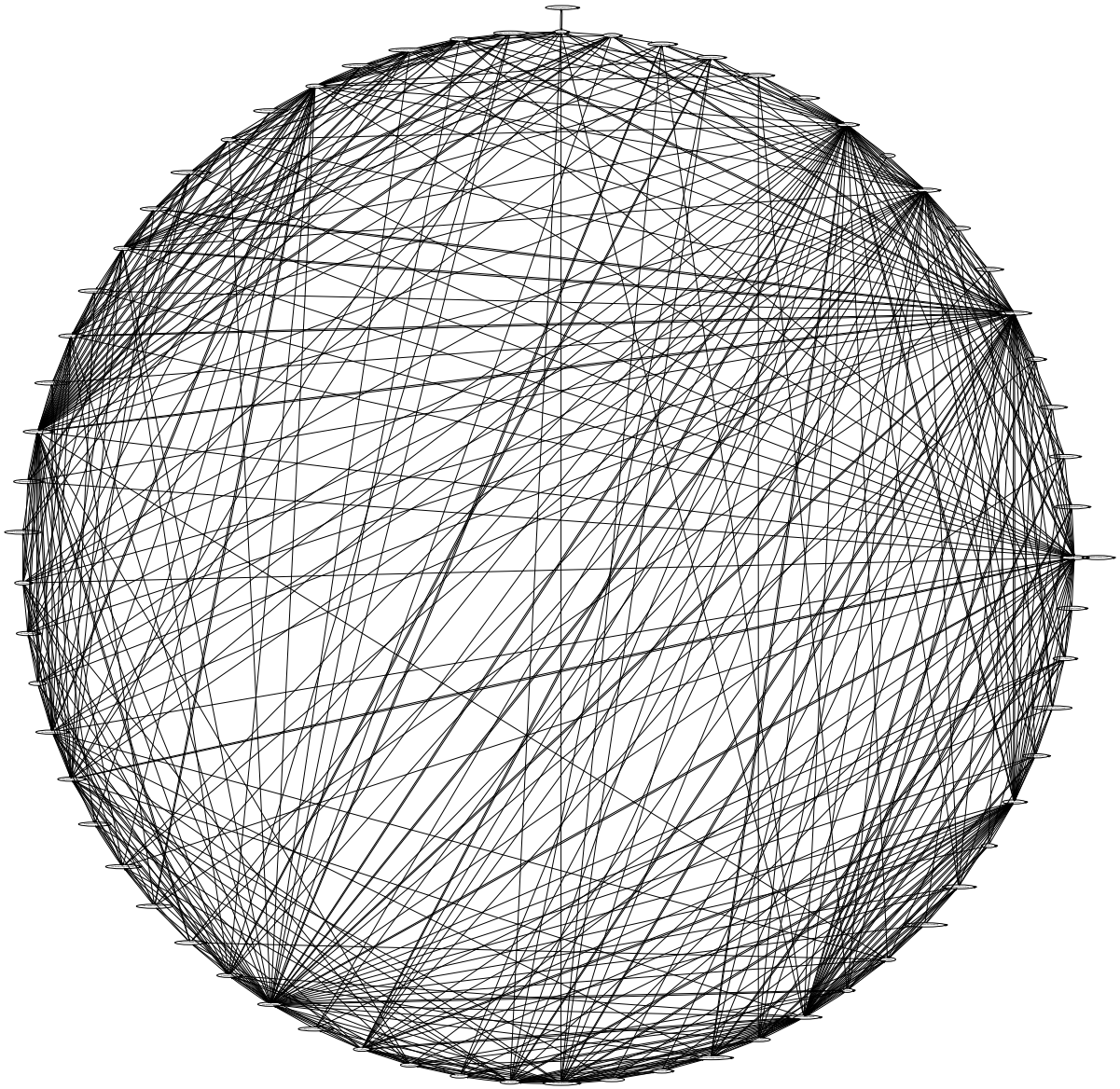


USC

Hadoop – Complete Architecture



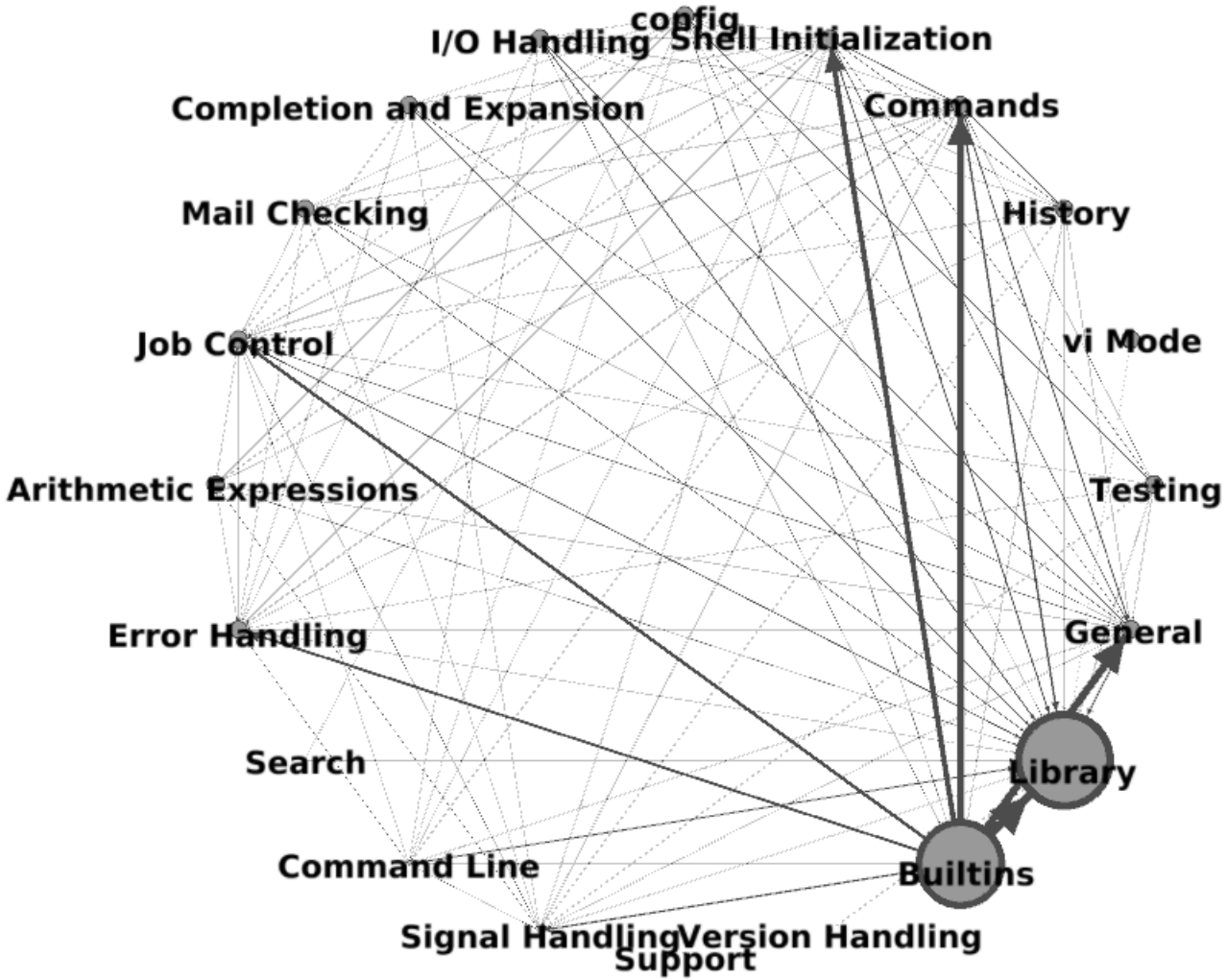
Hadoop – Complete Architecture
(Somewhat Beautified)



USC

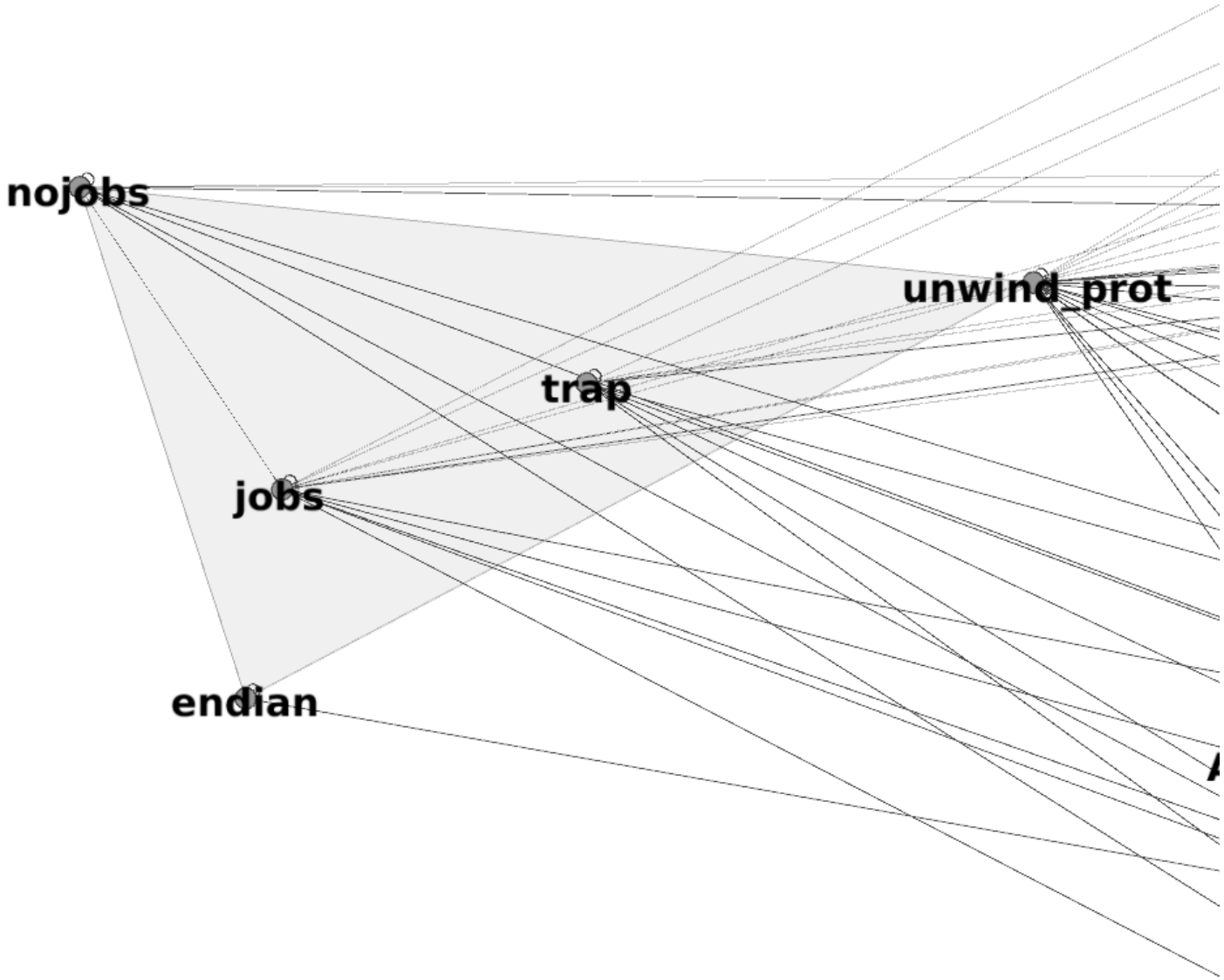
Another Example – bash

Top-Level Architecture



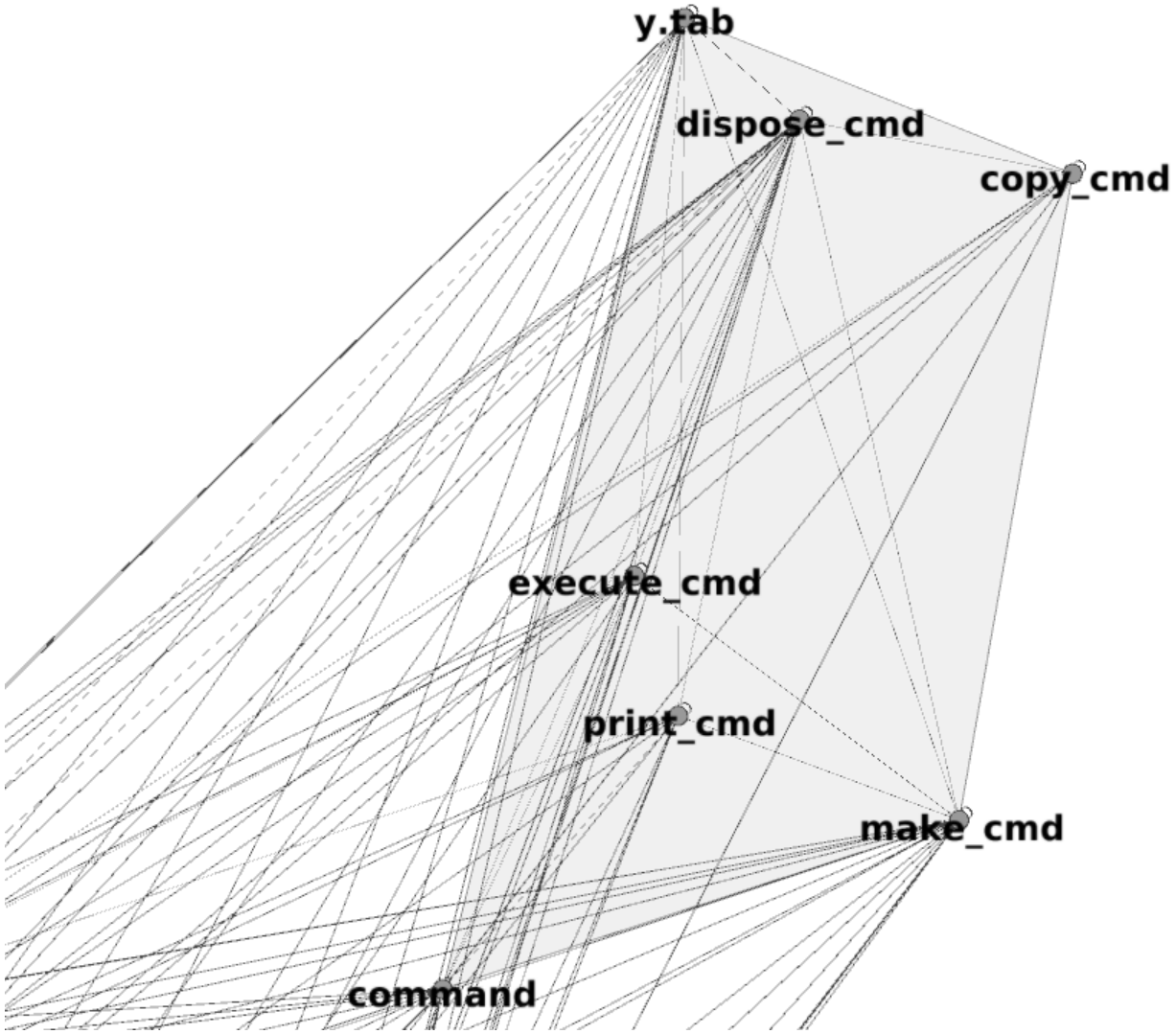
USC

bash – Job Control Component



USC

bash – Commands Component



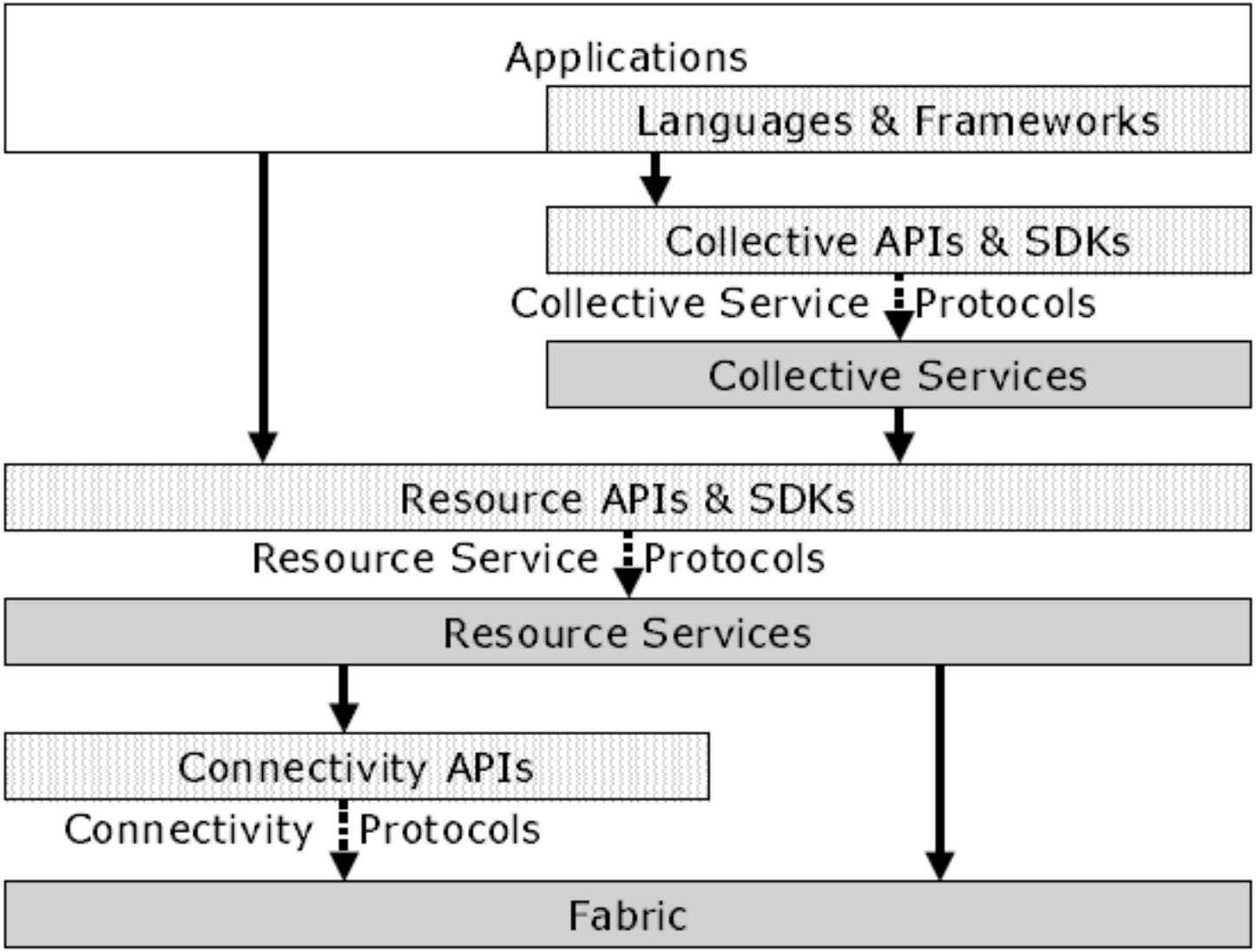
USC

How Prevalent Is Architectural Degradation?

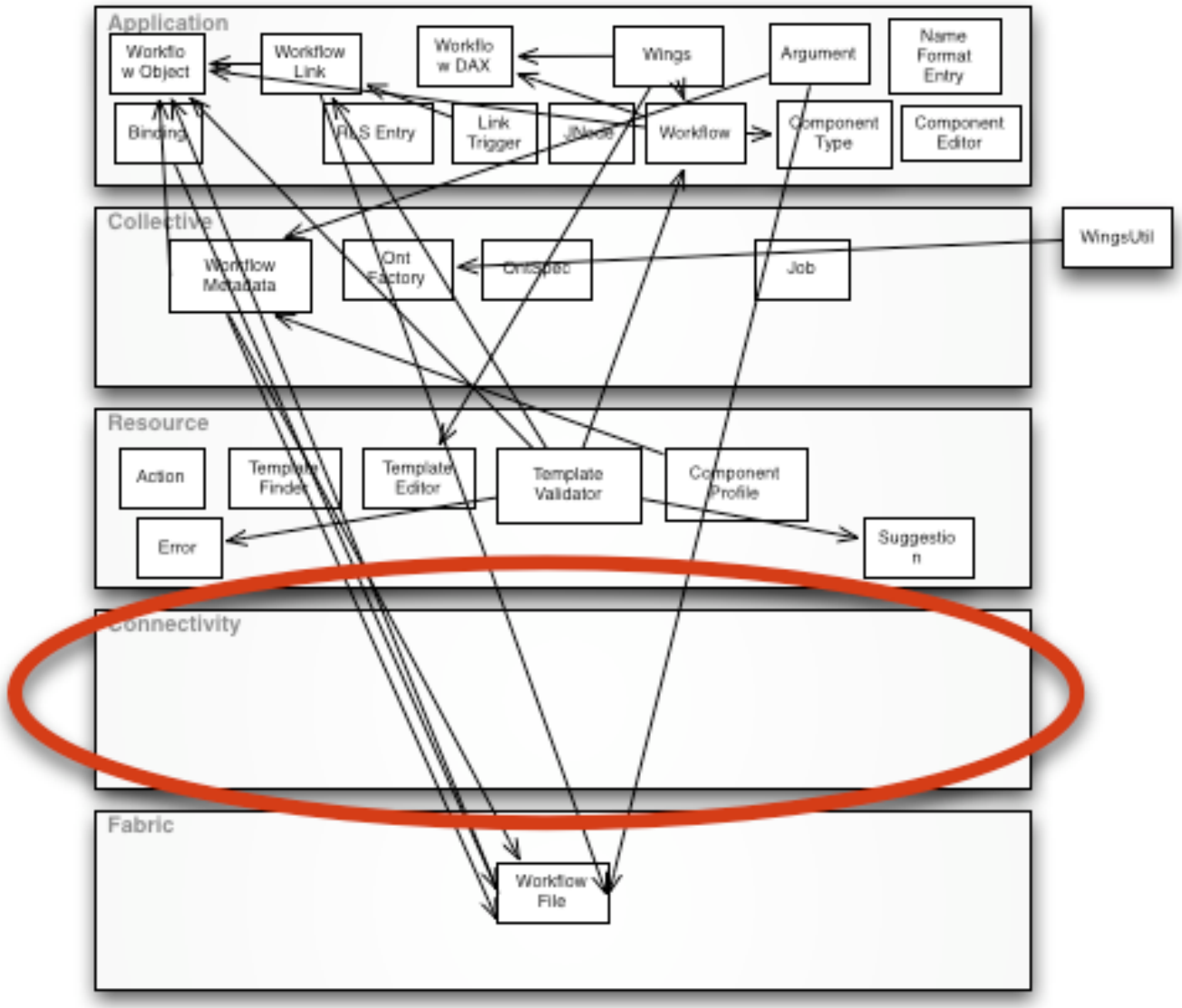
Technology	PL	KSLOC	# Modules
Alchemi	C# (.NET)	26.2	186
Apache Hadoop	Java, C/C++	66.5	1643
Apache HBase	Java, Ruby, Thrift	14.1	362
Condor	Java, C/C++	51.6	962
DSpace	Java	23.4	217
Ganglia	C	19.3	22
GLIDE	Java	2	57
Globus 4.0 (GT 4.0)	Java, C/C++	2218.7	2522
Grid Datafarm	Java, C	51.4	220
Gridbus Broker	Java	30.5	566
Jcgrid	Java	6.7	150
OODT	Java	14	320
Pegasus	Java, C	79	659
SciFlo	Python	18.5	129
iRODS	Java, C/C++	84.1	163
Sun Grid Engine	Java, C/C++	265.1	572
Unicore	Java	571	3665
Wings	Java	8.8	97

Reference Architecture for the Grid

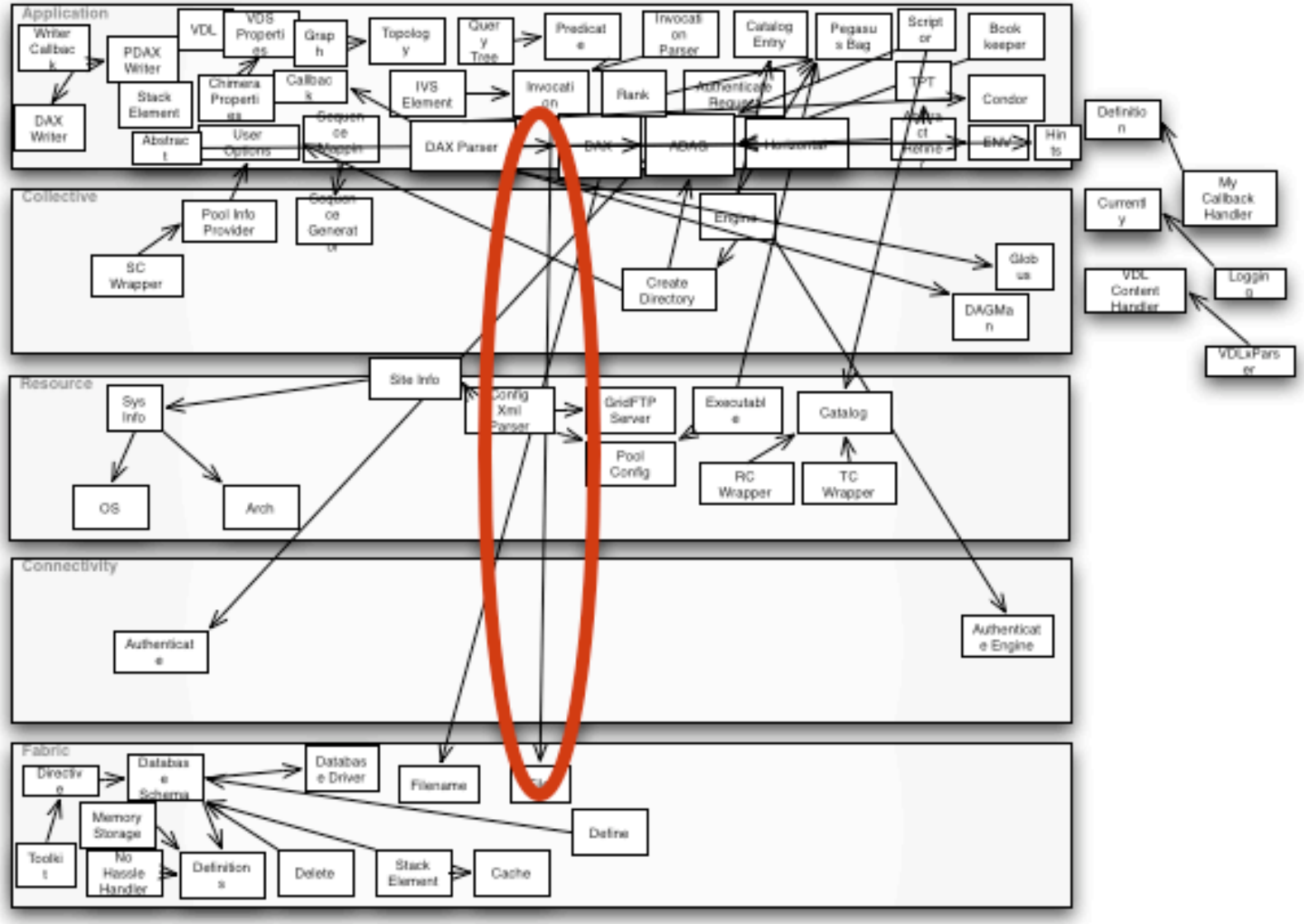
K



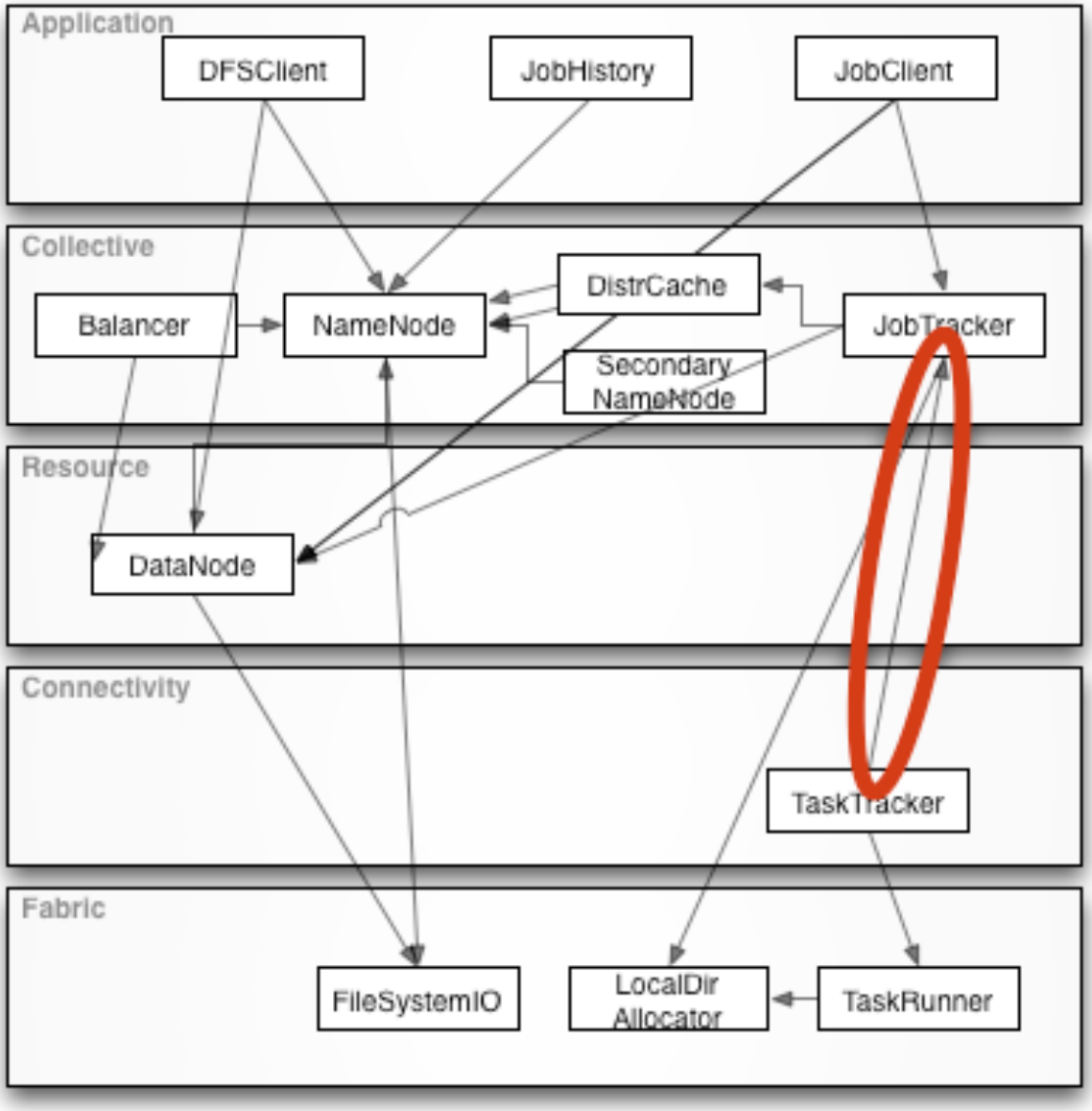
Empty Layers – Wings



Skipped Layers – Pegasus

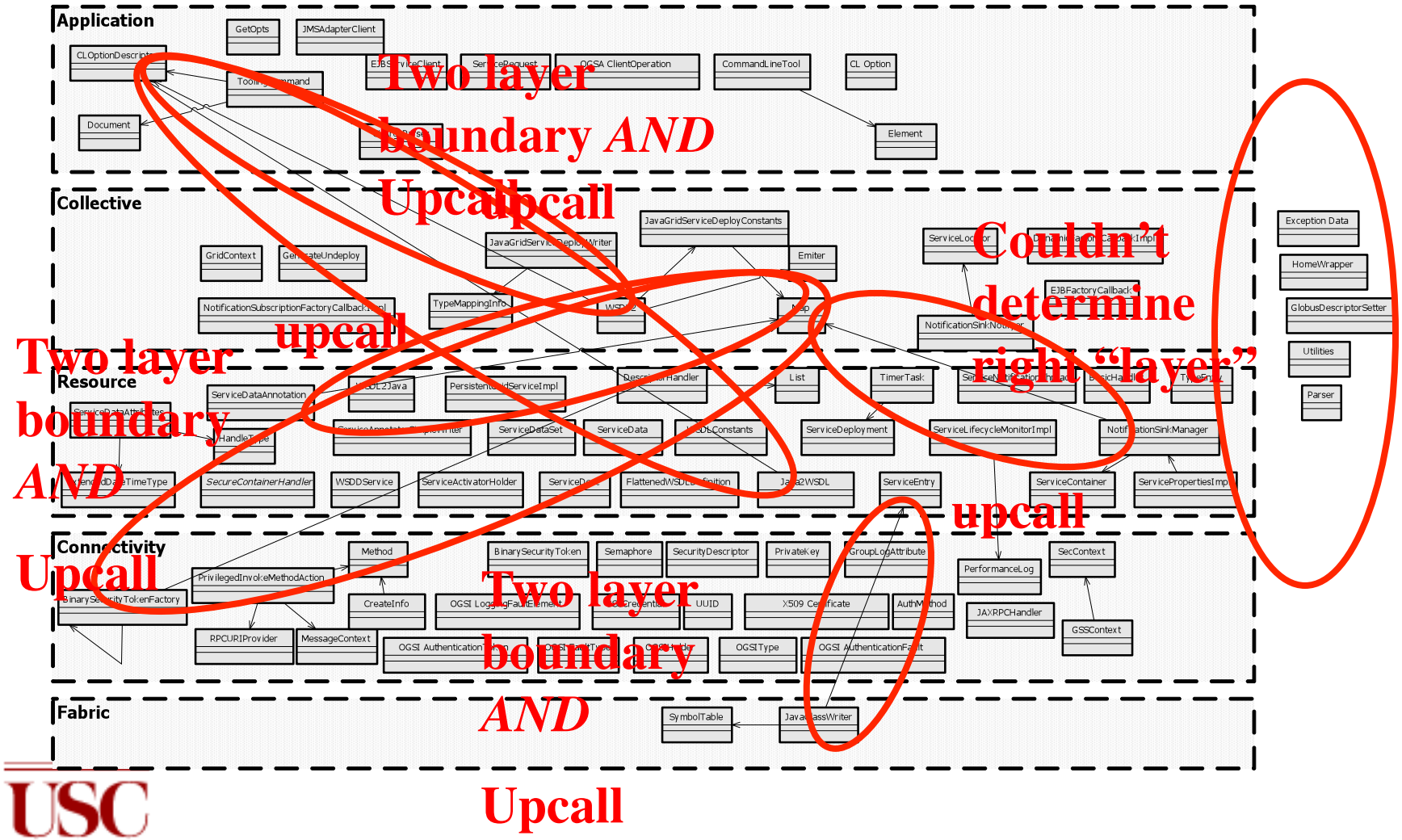


Upcalls – Hadoop



Multi-Layer Components – iRods

A Bit of Everything in Globus



Degradation Tally

Technology	UC	SL	ML	EL
Alchemi	2	2	2	0
Apache Hadoop	2	6	0	1
Apache HBase	5	7	0	0
Condor	2	0	0	2
DSpace	1	4	1	0
Ganglia	2	2	0	1
GLIDE	2	5	0	0
Globus 4.0	6	3	0	0
Grid Datafarm	14	16	0	0
Gridbus Broker	4	4	0	1
Jcgrid	4	9	0	0
OODT	3	7	0	0
Pegasus	6	6	0	0
SciFlo	1	1	0	0
iRODS	36	46	2	0
Sun Grid Engine	1	2	0	1
Unicore	4	13	0	0
Wings	6	10	0	1
Totals:	101	143	5	7

USC

Can We "Smell" Architectural Decay?

- A successful system is maintained over multiple years
- After a while a system becomes like sausage
- When and where do we refactor its architecture?

➢ **Architectural smell**

- A commonly made architectural design decision that negatively impacts a system's lifecycle properties
- A smell is not a bug; it doesn't break your system – but that may be the very thing that makes it dangerous

USC

Inspiration

- ***Refactoring: Improving the Design of Existing Code***

  by Martin Fowler

- **Code smells are implementation structures that negatively affect system lifecycle properties**
- **Defined in terms of *implementation-level* constructs**
  - Examples: long parameter list, large methods, …
- **Code smells do not address *architectural decisions***
  - They are not necessarily even correlated with architectural degradation [Macia et al. 2012]

USC

A Catalogue of Architectural Smells

- Brick Concern Overload
- Brick Use Overload
- Brick Dependency Cycle
- Unused Interface
- Ambiguous Interface
- Duplicate Component Functionality
- Scattered Functionality
- Component Envy
- Connector Envy
- Connector Chain
- Extraneous Adjacent Connector

USC

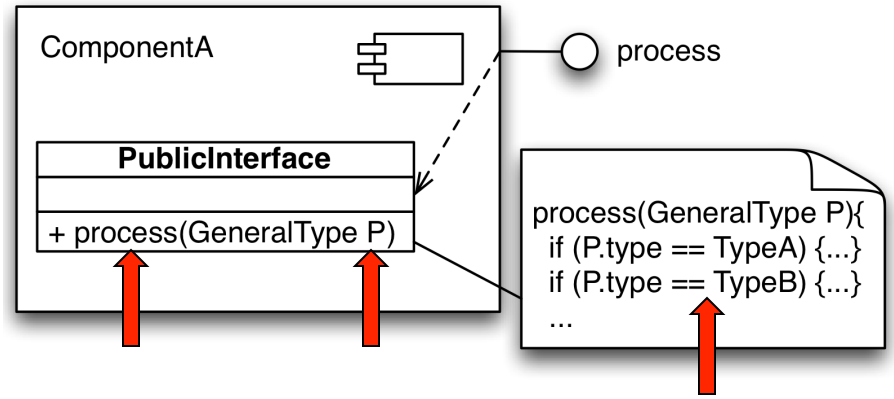
A Catalogue of Architectural Smells

- Brick Concern Overload
- Brick Use Overload
- Brick Dependency Cycle
- Unused Interface
- **Ambiguous Interface**
- Duplicate Component Functionality
- **Scattered Functionality**
- Component Envy
- **Connector Envy**
- Connector Chain
- **Extraneous Adjacent Connector**

USC

Ambiguous Interface – Description

- **Three facets of Ambiguous Interface**
  - internal dispatch to multiple services
  - offers only one public interface
  - accepts only general type



➢ **Must inspect component implementation to know its offered services or its impact**

USC

Ambiguous Interface – Example

Java Messaging Service

```
public void onMessage(Message msg)
  {
        String msgText;
        if (msg instanceof TextMessage) {
        ...
        } else { // If it is not a TextMessage...
        ...
        }
        if (msgText.equalsIgnoreCase("quit")) {
        ...
        }
        ...
        ...
  }
```

Ambiguous Interface – Example

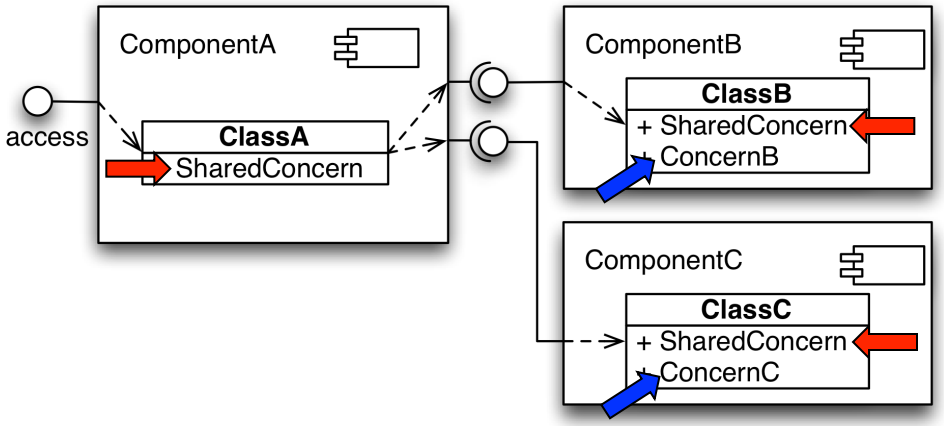
JSP servlet

```
void service(Request req)
  {
        String searchType =
          req.getParameter( “searchPref“ );
        if (searchType.equals( “zip” )) {
        ...
        } else if (searchType.equals( “type” )) {
        ...
        } else {
        ...
        }
  }
```

Scattered Functionality

- **Multiple components are responsible for realizing the same high-level concern**
- **Some of those components are responsible for orthogonal concerns**

➢ **Negatively affects reusability, understandability, testability**

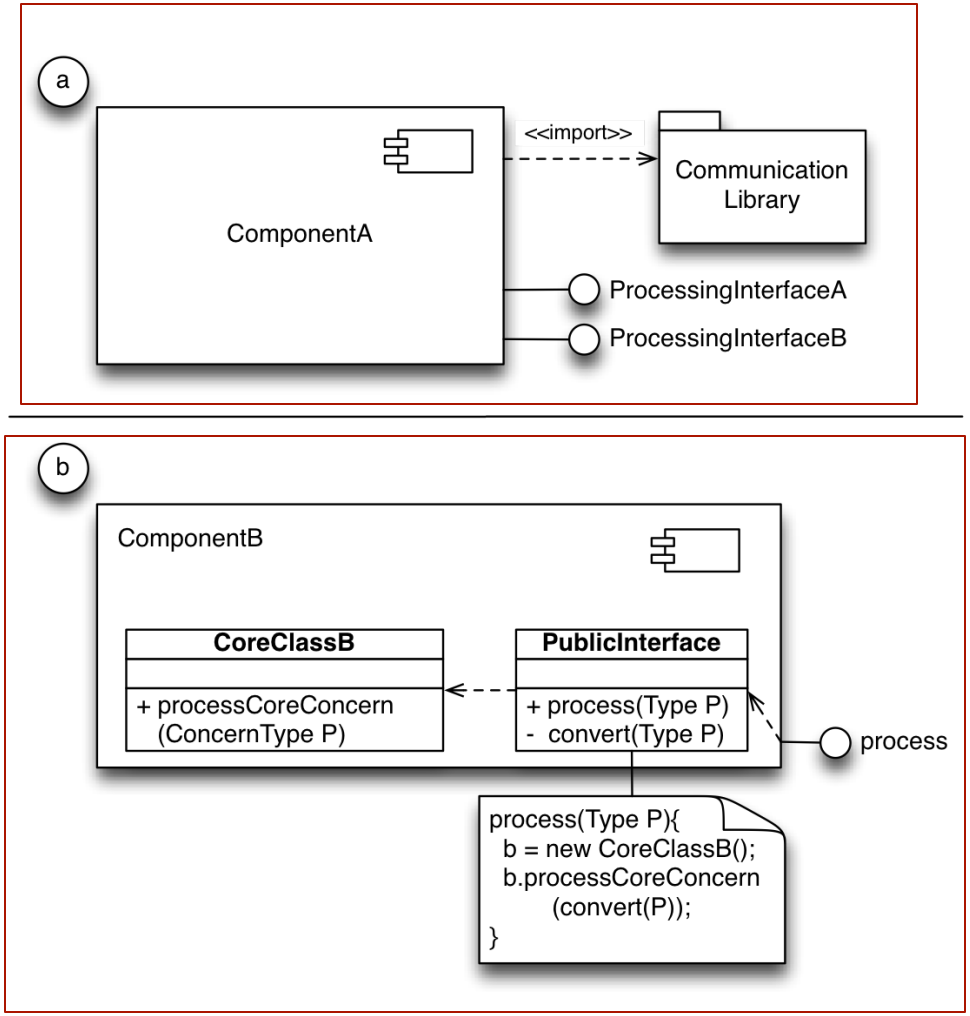


USC

Connector Envy

- **Components provide interaction-related services**
  - Communication
  - Coordination
  - Conversion
  - Facilitation

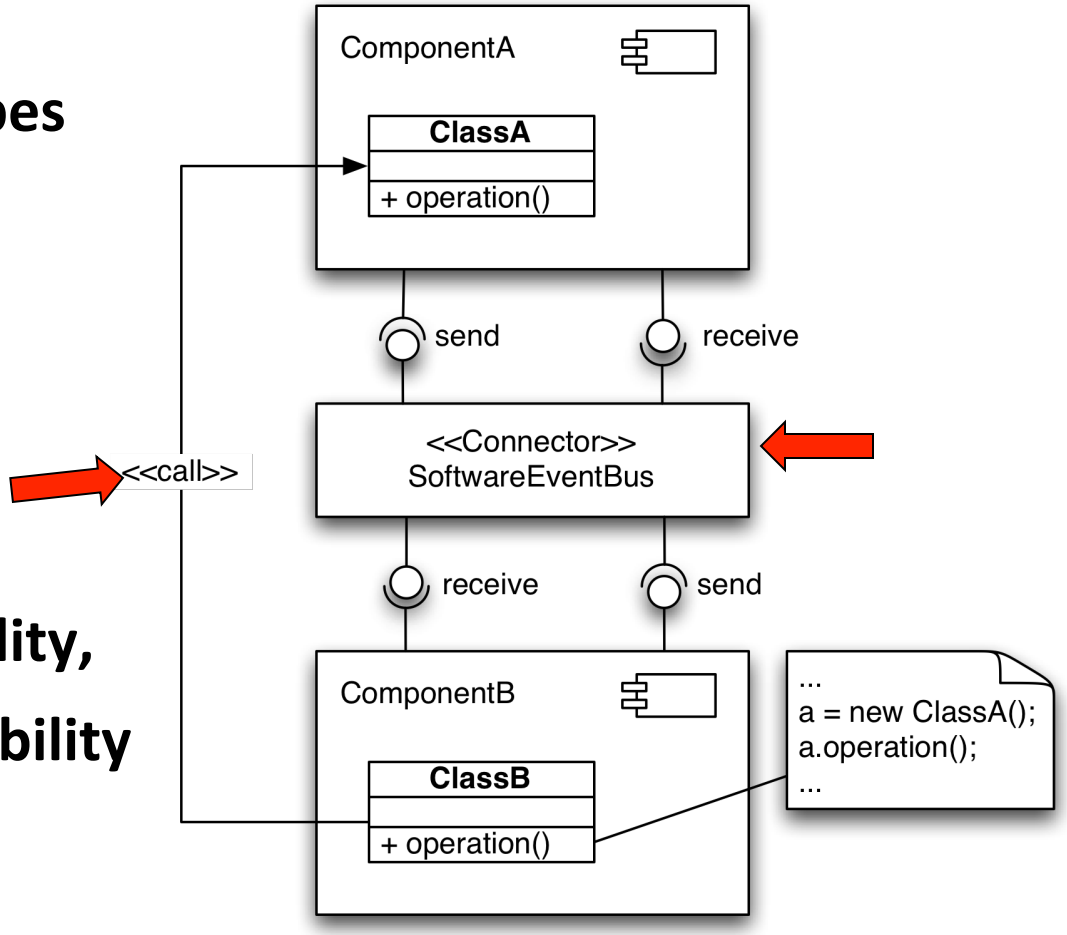
➢ **Negatively affects reusability, understandability, maintainability**



Extraneous Adjacent Connector

- **Connectors of different types link a pair of components**
- **Connectors' benefits may cancel each other**

➢ **Negatively affects reusability, understandability, adaptability**



Conclusion

- **Your architecture will degrade and get "smelly"**
  - Sometimes it starts off that way
- **Refactoring is extremely expensive**
- **You may have no choice**
- **But how do you know when, where, and how to refactor?**
- **Understanding and cataloguing smells is only the first step**
- **We are also working on**
  - Improving architectural recovery by exploiting system concerns
  - Identifying smell patterns in systems
  - Understanding their correlation with code smells

USC

Acknowledgments

- **Support from NSF, Bosch RTC**
- **Project participants and collaborators**
  - Yuanfang Cai, Drexel University
  - Chris Douglas, Yahoo Labs
  - Alessandro Garcia, PUC-Rio
  - **Joshua Garcia, USC**
  - Muzzammil Imam, USC
  - **Ivo Krka, USC**
  - **Isela Macia, PUC-Rio**
  - Chris Mattmann, NASA/JPL
  - Daniel Popescu, Google
  - Arndt von Staa, PUC-Rio

USC