A Service Oriented Architecture (SOA) Primer

All modern systems are distributed

1. Challenges with Distributed Systems
2. Consistency
3. Other Problems
4. Architecture Patterns – this is where SOA comes in
5. API and Framework Considerations

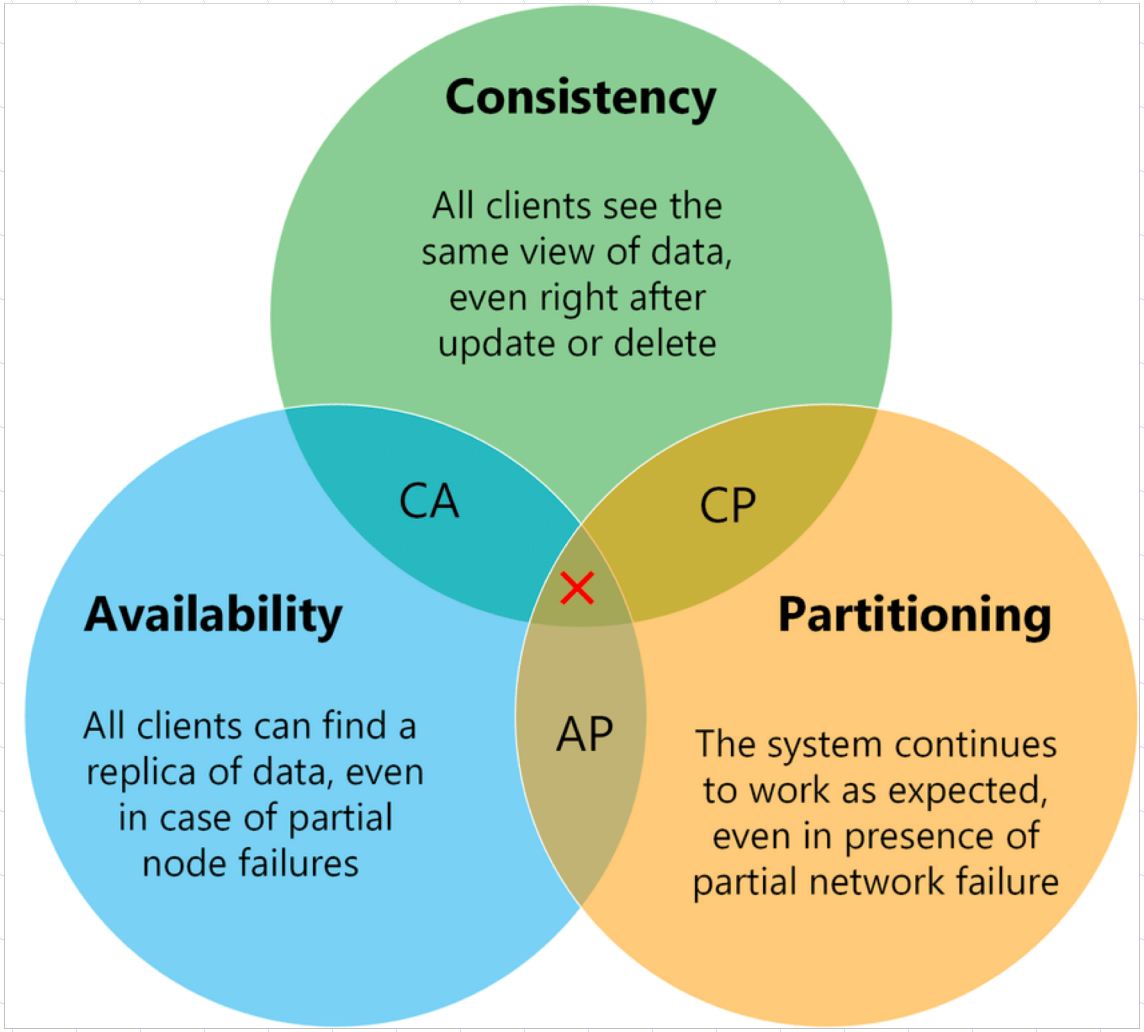
Challenges with Distributed Systems

These are generally known as the "fallacies of distributed computing"

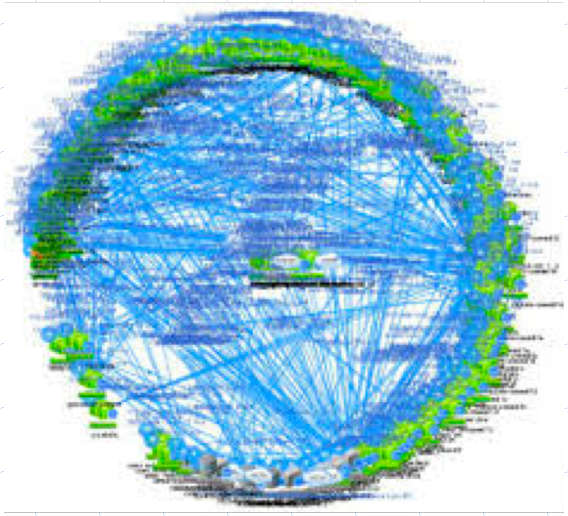
1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

http://en.wikipedia.org/wiki/Fallacies_of_distributed_computing

CAP Theorem – Can only pick 2



Other Challenges



The many nodes in a distributed system cant guarantee that they see the same ordering of events (or messages)

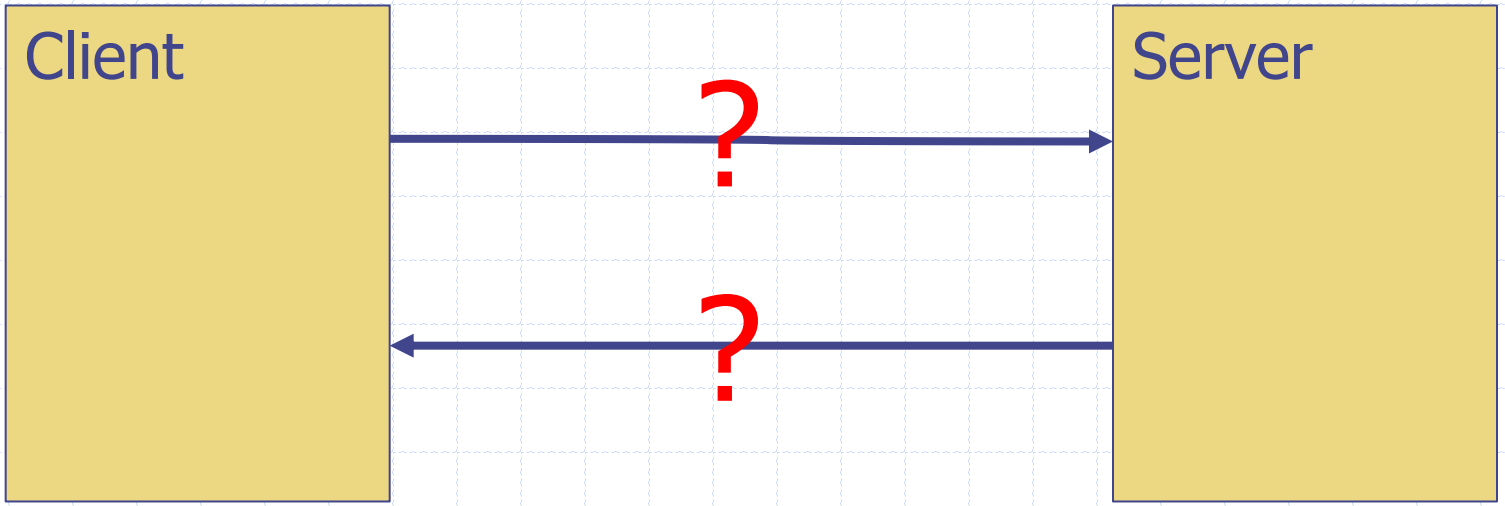
Nor can they agree or synchronize based on time

Security Challenges

Also, the various nodes in a distributed system should not trust each other

Unreliable Network Challenges

Client and Servers in a distributed system cannot make certain assumptions about each other

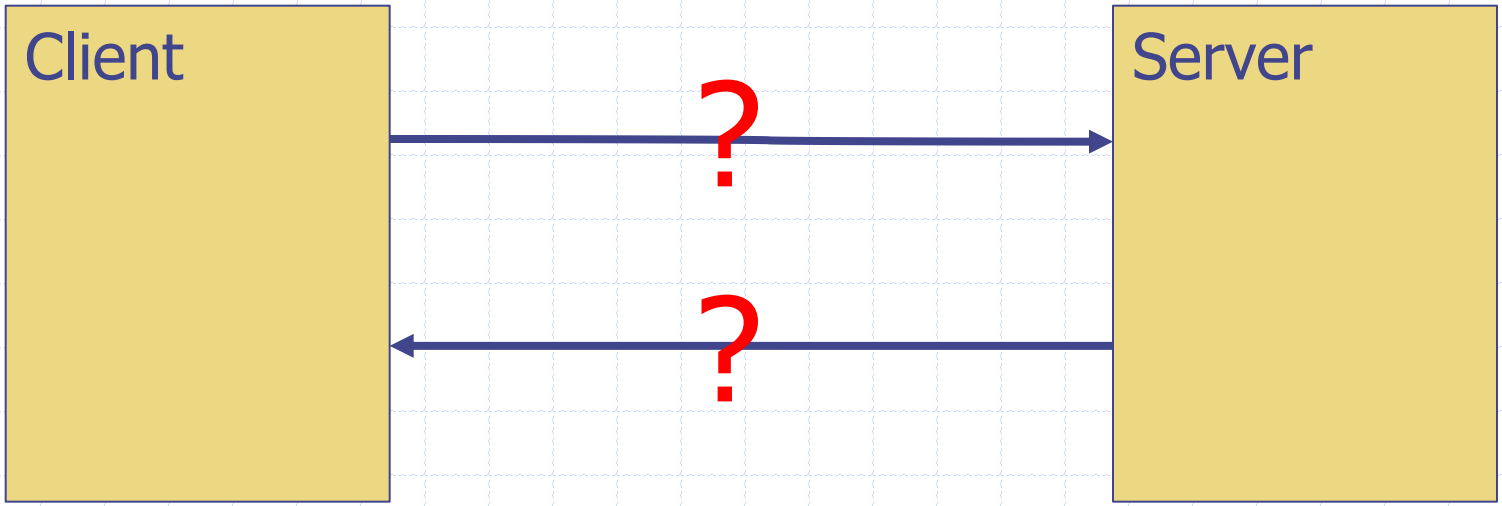


Did the server get my message?
Is the server slow?
Did the server die?

Is the client still there?
Did the client get my response?

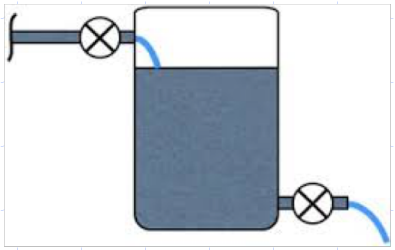
These problems can be partially addressed using timeouts

Flow Control Challenges



Can the client keep up with the data being returned from the server

Can the server keep up with the client volume?



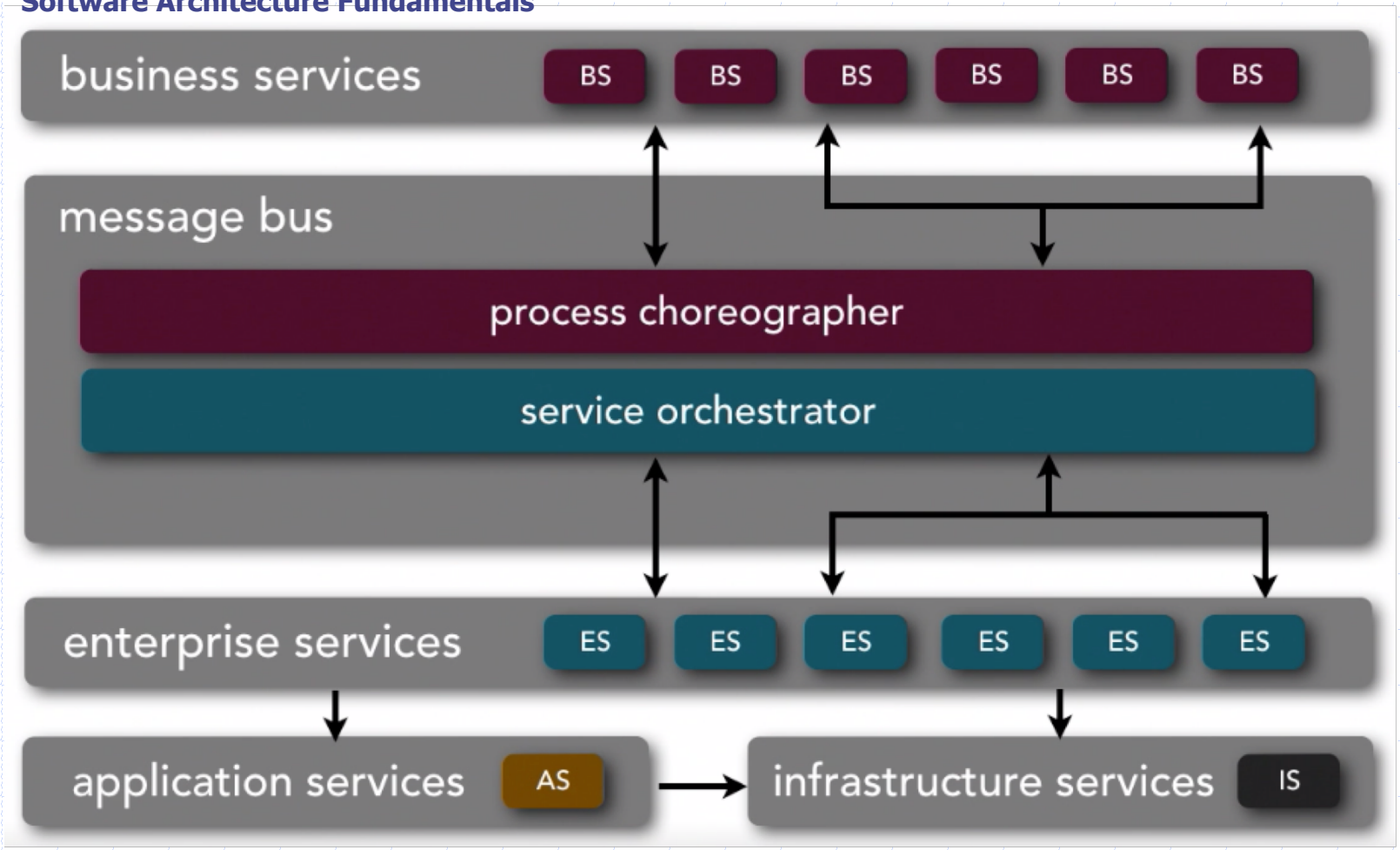
These problems are dealt with using patterns like back pressure and circuit breakers

Architecture Styles

1. Service Oriented Architecture, versus
2. Service Based Architecture, versus
3. Microservices Architecture

Service Oriented Architecture Style

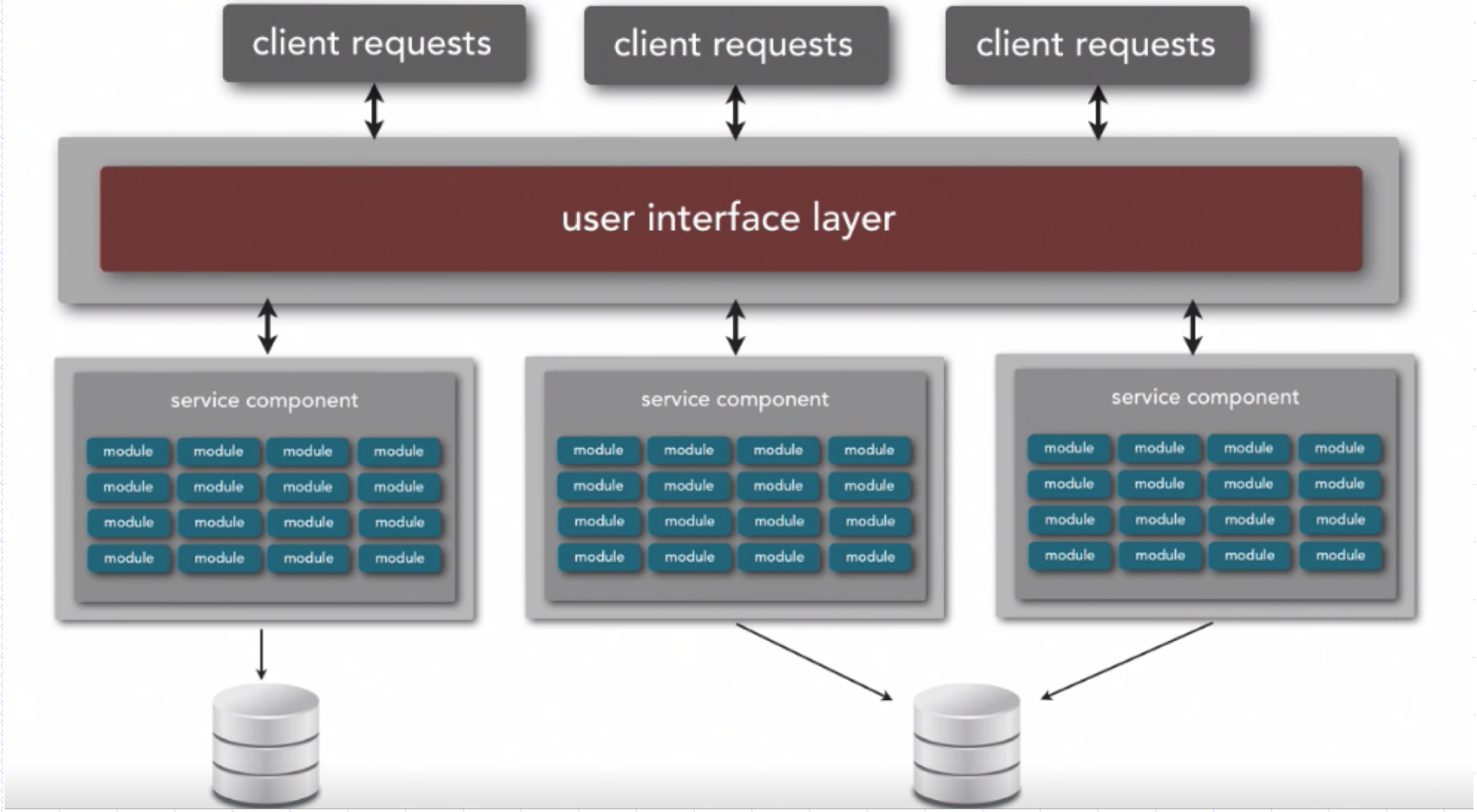
Ref: Neil Ford & Mark Richards
Software Architecture Fundamentals



Smart Middleware – Sometimes called an ESB

Service Based Style

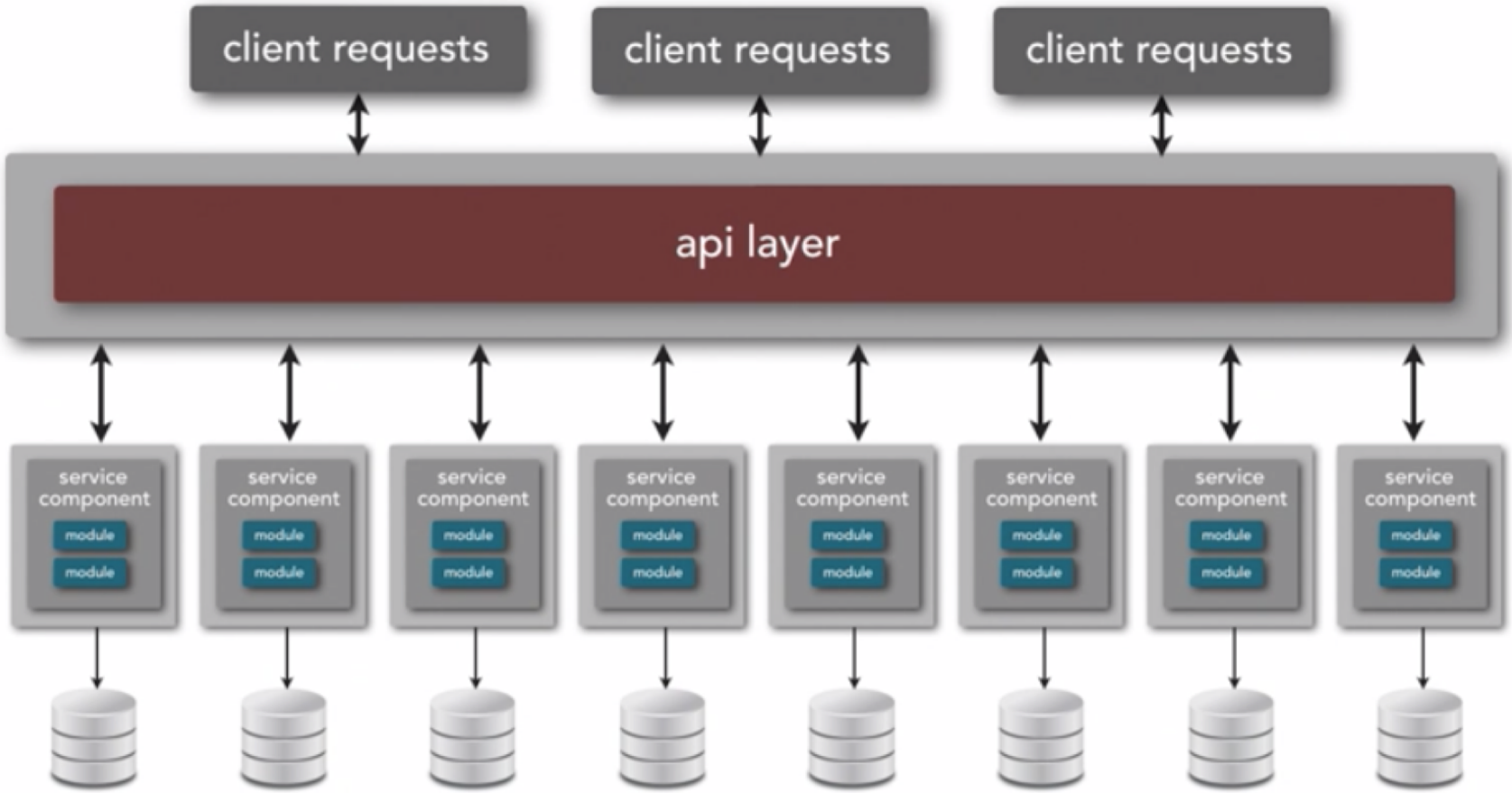
Ref: Neil Ford & Mark Richards
Software Architecture Fundamentals



Works Well To Partition Functionality

Microservice Architecture

Ref: Neil Ford & Mark Richards
Software Architecture Fundamentals



Basic Idea – SHARE NOTHING

Programming Models and Considerations

1. Framework and Tools
   - Scalability
   - Resiliency
   - Programming Language Support
2. Programming Model
   - Functional versus annotation based
   - Type Safety
   - Sync- or Async-
   - Streaming Support

The SOA Samples

https://github.com/ArchitectingSoftware/cs575-soa-demo.git

Example	Framework	Notes
graphql-typescript	GraphQL	Demonstrates the use of the graphql framework for type-safe queries to services
grpc	Google – GRPC & Protocol Buff	Type safe, high performance, polyglot rpc based messaging system. Async and streaming
rest-go	Golang based framework gonic-gin	Async golang based REST Service
rest-java	Spring	Spring boot 2.0 annotation based server
rest-kotlin	Spring	Spring boot 2.0 functional based serfver – Kotlin Programming language

The SOA Samples

https://github.com/ArchitectingSoftware/cs575-soa-demo.git

Example	Framework	Notes
rest-ktor	Ktor	Kotlin Ktor REST Service interface, uses Kotlin Co-Routines for concurrency
rest-node-koa	Koa	Uses Koa and typescript with the latest javascript support for async/await
rest-scala	Akka Http	Async scala based REST Service – uses Akka actors
Soap-java	Spring	Spring boot SOAP Based web service

Approaches to Identify Services in a SOA Design

- 1. Business Process Decomposition
  - Businesses can be viewed in terms of their primary processes, these processes can be further subdivided into sub-processes, and then decomposed again into very granular processes. These are typically called the L1, L2, and L3 processes
  - The L3 processes are "logical units of work" that support the business
  - Logical units of work have a clear input, output and rules that transform inputs into outputs – these can be services
  - Example: EnrollCustomerInLoyaltyProgram
- 2. Business Functions
  - Similar to business processes, many organizations operate using a collection of business functions
  - Designing around business functions are less likely to be biased by the way the business processes were implemented.
  - A function takes the form of y = f(x) where inputs "x" are transformed into a well defined output(s).
  - Example: ProcessCustomerPayment(x) applies x dollars against the customers account

Approaches to Identify Services in a SOA Design

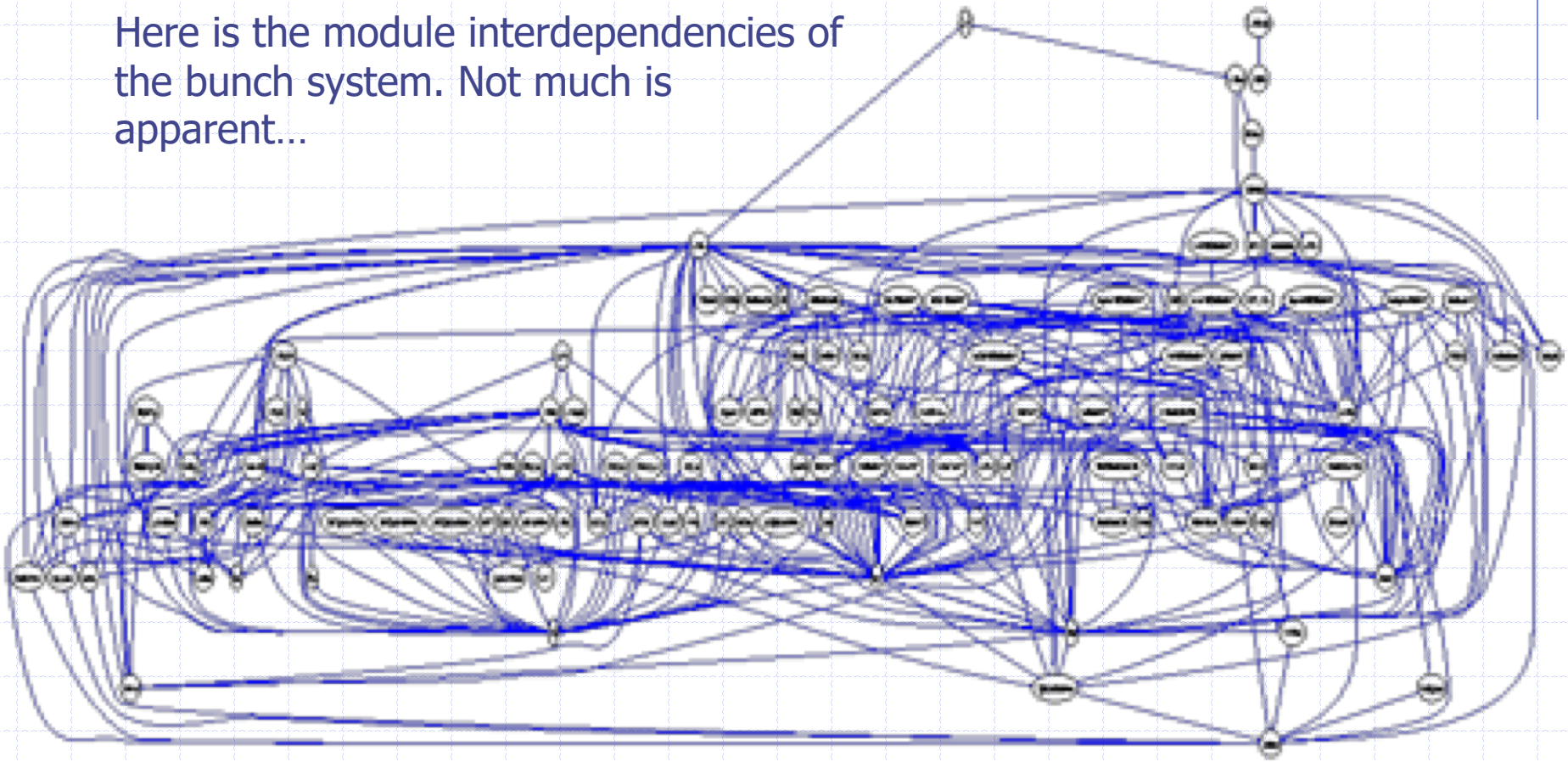
- 3. Look for "Business Entity" Objects in the problem domain
  - Most SOA solutions operate on data. Objects in an OOD work at the table level
  - SOA components operate at the business object level
  - Look for key business entities in the application domain from the perspective of (C)reate, (R)ead, (U)pdate, or (D)elete operations.
  - Examples: Customer, Payments, Order, etc
- 4. Look at "Ownership and Responsibility" to leverage reusable services built by others (where it makes sense)
  - This SOA design attribute is less about identifying a service, and more about identifying who builds, owns and maintains the service.
  - Many SOA applications are "mash ups", combining services from different parties.
  - When identifying a service, look for opportunities to reuse services that are offered by others when they are not key to differentiating you application.
  - Example: If you need a service for mapping or geo-coding would you build this yourself, or would you integrate a service offered by google?

Approaches to Identify Services in a SOA Design

- 5. Goal-Driven Service identification
  - Identify goals of your business or application that can be realized via automated support.
  - Example: Goal: "Improve customer transparency into service pricing"; Service: create an "Estimator" service that allows customer to estimate costs before they make a purchase decision
- 6. Component-Based
  - Use a traditional software engineering approach to identify services
    - Create a conceptual architecture view of the target application
    - Look for natural boundaries that adhere to the principles of maximizing cohesion and minimizing coupling.
    - These identify candidate components
  - For each candidate component see if they well defined responsibility, clear ownership, and if they can be distributed to run in different address spaces
  - Example: The bunch software clustering tool offers a clustering service where the modularization quality calculation is externalized as a service.

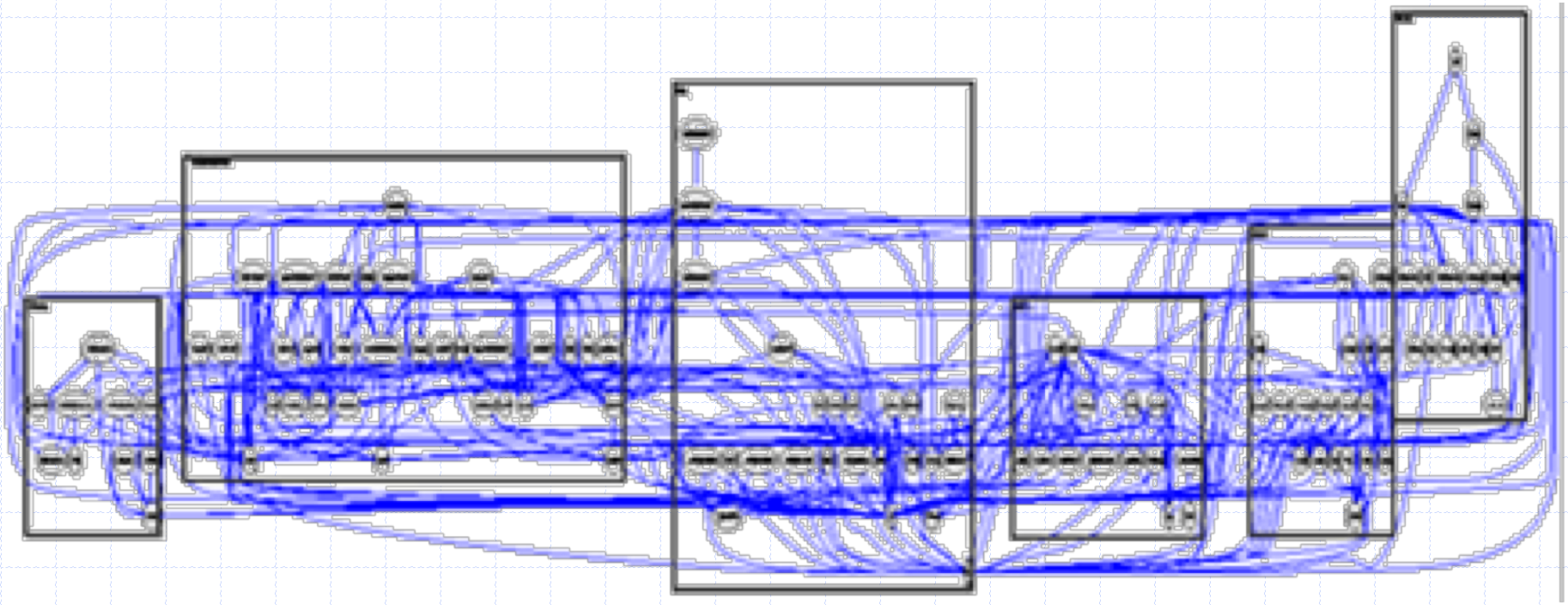
Component-Based Decomposition Example – The bunch system

Here is the module interdependencies of the bunch system. Not much is apparent…



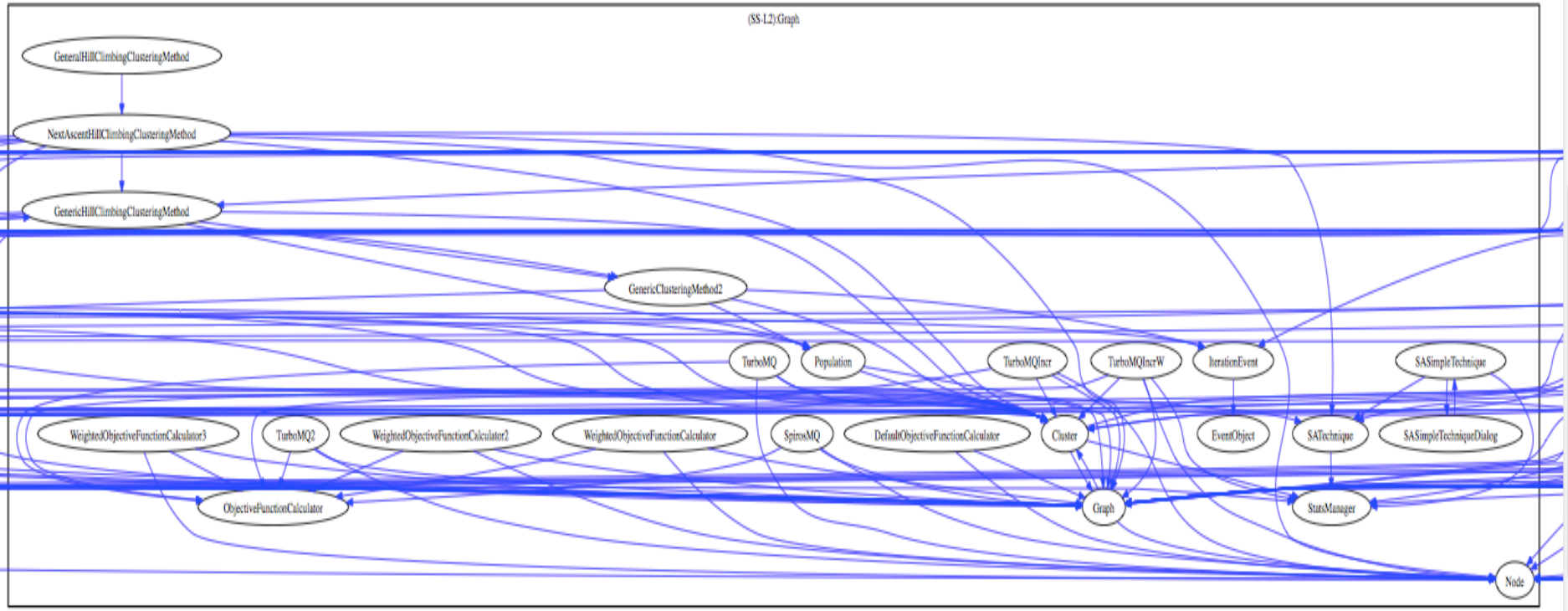
Component-Based Decomposition Example – The bunch system

Here is the module interdependencies of the bunch system. Lets run bunch itself to cluster the system to look for some candidate subsystems…

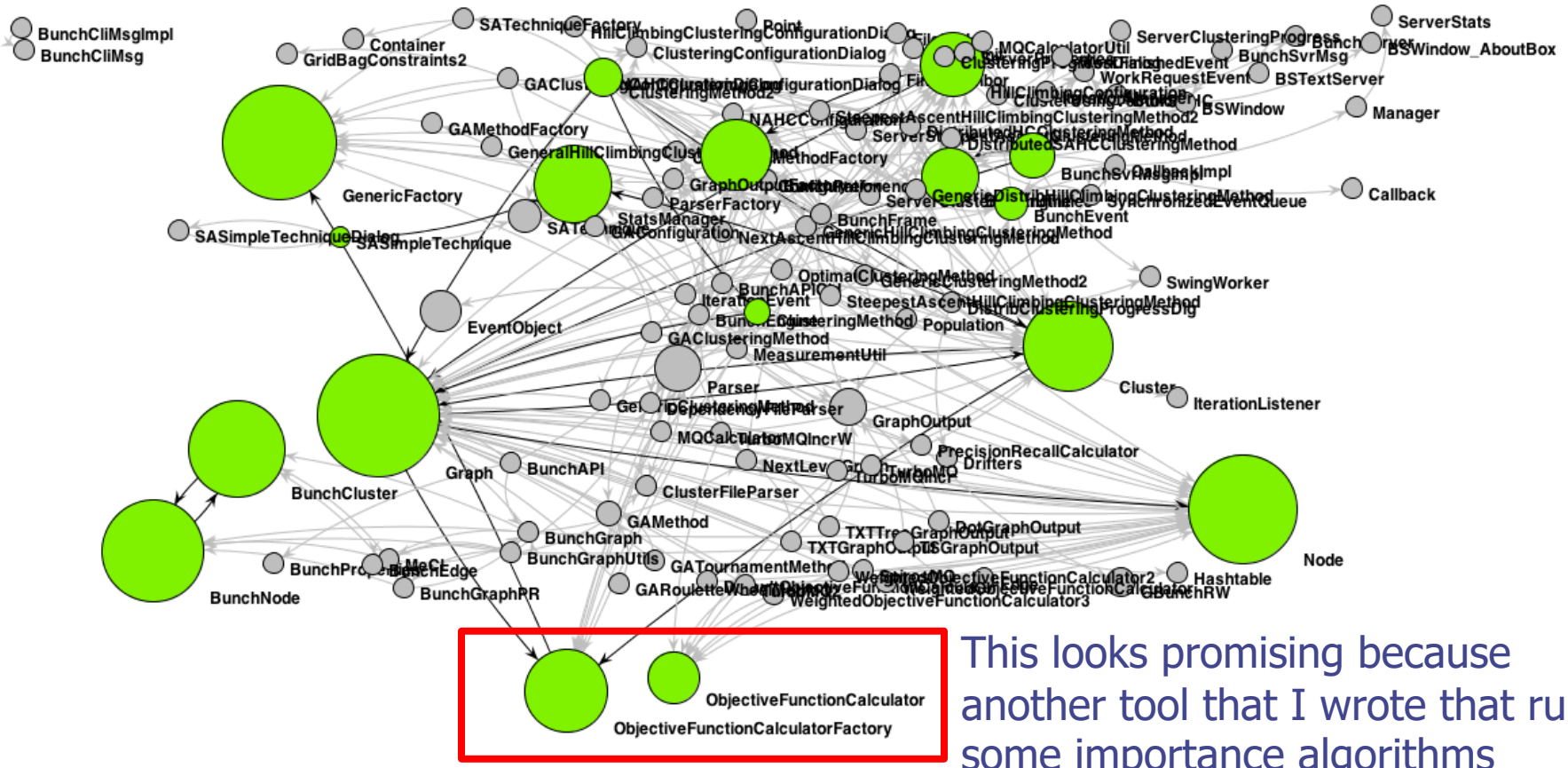


Component-Based Decomposition Example – The bunch system

This one looks promising – it has a lot of functionality related to evaluating the modularization quality function. Domain knowledge also tells us that its compute intensive so it might make for a good service…



Component-Based Decomposition Example – The bunch system



This looks promising because another tool that I wrote that runs some importance algorithms identifies some of these classes as important.

We refactored bunch to support distributed computation and describe it here…

An Architecture for Distributing the Computation of Software Clustering Algorithms

Brian Mitchell, Martin Traverso, Spiros Mancoridis
Department of Mathematics & Computer Science
Drexel University, Philadelphia, PA, USA
{bmitchel, umtraver, smancori}@mcs.drexel.edu

Abstract

Collections of general purpose networked workstations offer processing capability that often rivals or exceeds supercomputers. Since networked workstations are readily available in most organizations, they provide an economic and scalable alternative to parallel machines. In this paper we discuss how individual nodes in a computer network can be used as a collection of connected processing elements to improve the performance of a software engineering tool that we developed.

Our tool, called Bunch, automatically clusters the structure of software systems into a hierarchy of subsystems. Clustering helps developers understand complex systems by providing them with high-level abstract (clustered) views of the software structure. The algorithms used by Bunch are computationally intensive and, hence, we would like to improve our tool's performance in order to cluster very large systems. This paper describes how we designed and implemented a distributed version of Bunch, which is useful for clustering large systems.

handling applications that have significant interprocess synchronization and communication requirements. However, computationally intensive applications that can partition their workload into small, relatively independent subproblems, are good candidates to be implemented as a distributed system. One such application is a tool that we developed to recover the high-level subsystem structure of a software system directly from its source code.

More than ever, software maintainers are looking for tools to help them understand the structure of large and complex systems. One of the reasons that the structure of these systems is difficult to understand is the overwhelming number of modules, classes and interrelationships that exist between them.

Ideally, system maintainers have access to accurate requirements and design documentation that describes the software structure, the architecture and the design rationale. Unfortunately, developers often find that no accurate design documentation exists, and that the original developers of the system are not available for consultation. Without automated assistance, developers often modify the source code without a thorough understanding of how their modifications affect

Approaches to Identify Services in a SOA Design

- 7. Existing Supply – Refactoring into a SOA
  - Look at existing inventory of applications and how they work together
  - Identify existing integration interfaces – API calls, Database Transactions, Database Queries
  - Select integration points that can be migrated away from using existing integration patterns into a SOA pattern
  - Example: Refactor an application that supports customer service, identify all ODBC and SQL calls that related to customer management, refactor by creating a SOA service for "Customer"
- 8. Front Office Application Usage Analysis
  - Most enterprise have multiple applications that have some underlying redundant functionality – typically this functionality is implemented differently in each application.
  - Identify these redundant functions and see if they can be encapsulated into a common service.
  - Example: Drexel grading data is used by students, facility and the administration – it can be accessed via the web, but we would like to extend to mobile. Solution is to create a common grading service that can be used by multiple applications. Imagine a service like "HoldGrades()" that is called from the billing system

Approaches to Identify Services in a SOA Design

- 9. Infrastructure
  - Although not common, there are cases when services are useful to take advantage of sharing infrastructure-specific capabilities.
  - Consider that you have a special piece of hardware that performs a very specific function and you want to share that capability with a lot of consuming applications.
  - Consider when you need to isolate a specific function to a certain piece of hardware or network zone for compliance purposes
  - Create a service that hides the specifics of the special infrastructure and expose it to other applications. Example: Use PayPal's payment services instead of doing your own credit card processing – eliminates the need to deal with PCI compliance.
- 10. Look at non-functional requirements.
  - Examine the set of non-functional requirements and look for opportunities where centralizing the implementation of certain capabilities helps enable the non-functional requirements.
  - Common examples are from security and performance set of non-functionals.
  - Example: Create a service to encapsulate making authorization decisions abstracting the complex dataflow of the oAuth protocol; another example is the distribution of the MQ calculation in the Bunch tool (from Method #6)

Reference: "Ten Ways to Identify Services", http://searchsoa.techtarget.com/tip/Ten-ways-to-identify-services