Corporate Structure, Component Teams & Conway's Law

How your organization influences your code and ability to deliver

Based on the July 18 AgileTO talk by

Craig Larman

co-inventor of LeSS

Craig Larman gave a 2-hour talk to the AgileTO agile Meet-up group on Tuesday July 18, 2018.

The talk was divided into three parts: 1) Formal, 2) Informal, 3) Q&A.

The formal piece used a slide deck and its primary thesis was that LeSS was not a scaling framework, but a de-scaling framework: large organizations adopt LeSS to reduce organizational overhead.

The informal piece was done via fat-markers and poster paper. He spent about 45 minutes conveying the concepts shown in this deck, up to slide 12.

The Q&A session was to run 45 minutes, CT did not stick around to see this.

BMW is adopting LeSS

for their Autonomous Driving Initiative

Craig name dropped that he is working with the BMW Autonomous Driving Group. There is a short video available on YouTube from BMW that talks about this group at a high level. Craig did not use it in his talk.

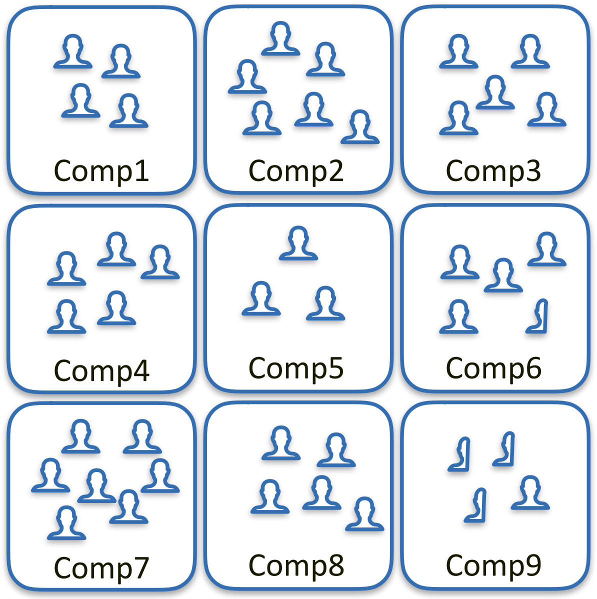
Welcome to the BMW Group Autonomous Driving Campus - YouTube
https://www.youtube.com/watch?v=Hbm6IcD78R0

An important success factor in LeSS is feature teams

Component Based Teams

- Traditional software development is usually done with component based teams
  - E.g. “database team”, “GUI team”

Simple Example



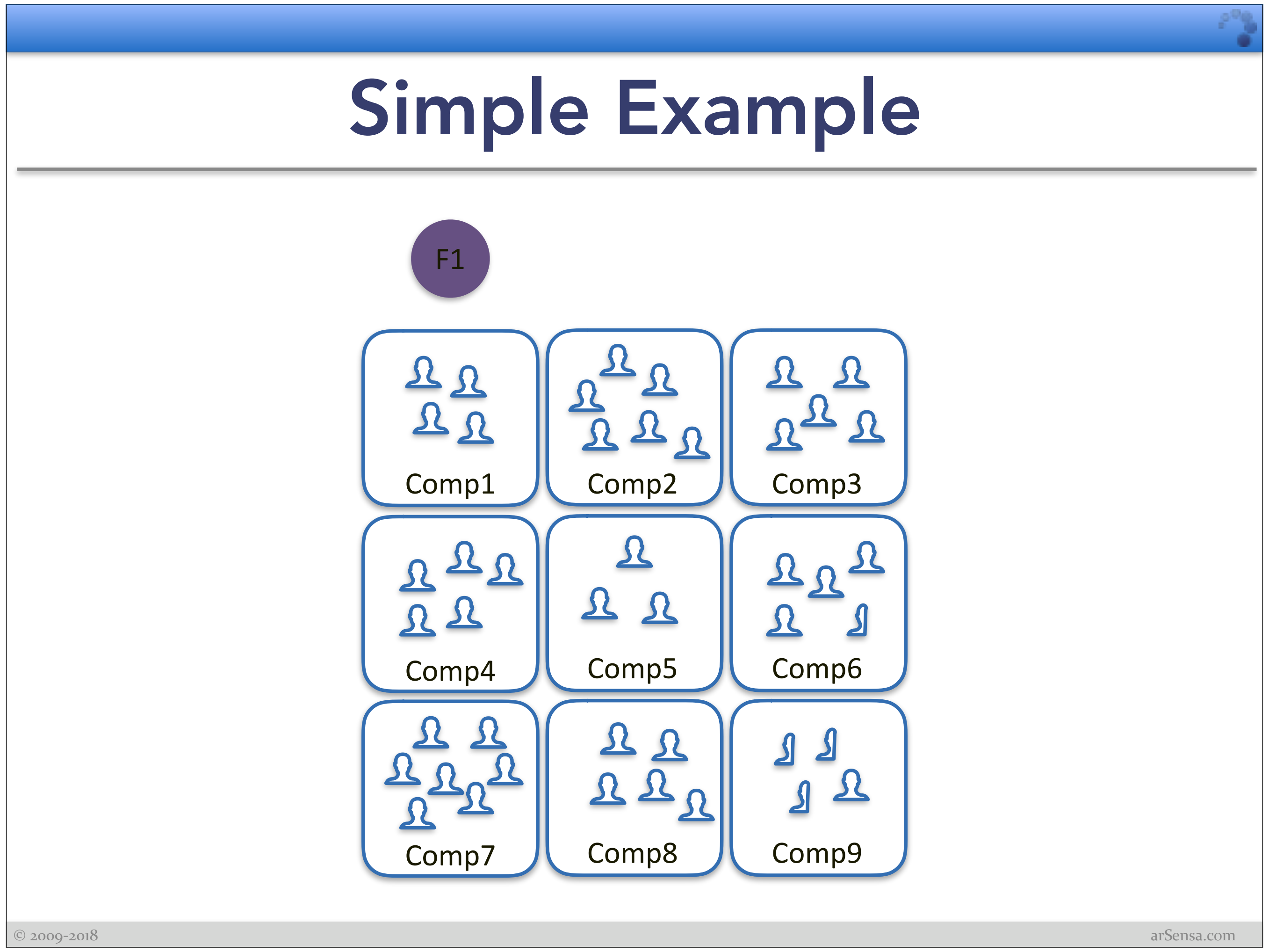
Large organizations often have a significant number of components. There isn't always a 1:1 relationship between component and team, but the organizational basic structures often reflect the component structure.

Craig has worked with organizations that have 100s, if not 1000s of components.

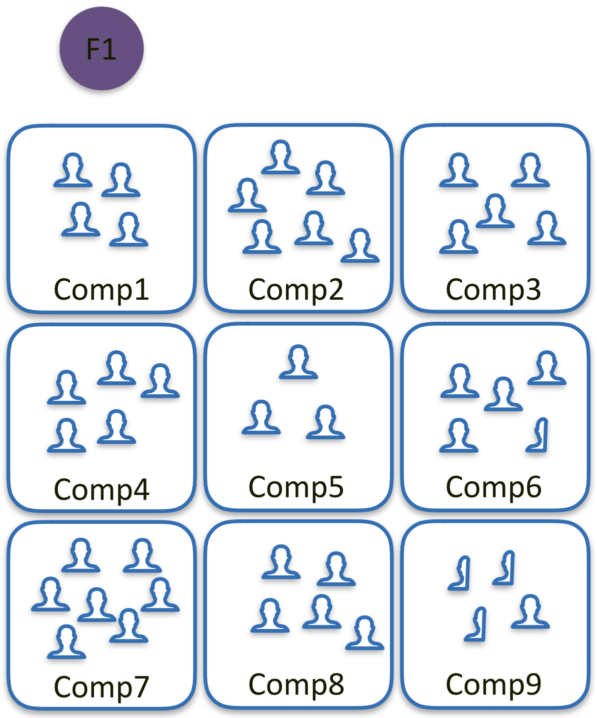
To keep the example simple, consider a system with 9 components.

Consider feature F1. It touches several different components. Now consider feature F2, it also touches several components, some of which are the same components as F1. This causes orchestration problems.

Craig's diagram went on to make the screen a mess, highlighting what this would mean for 100s of features.



Simple Example



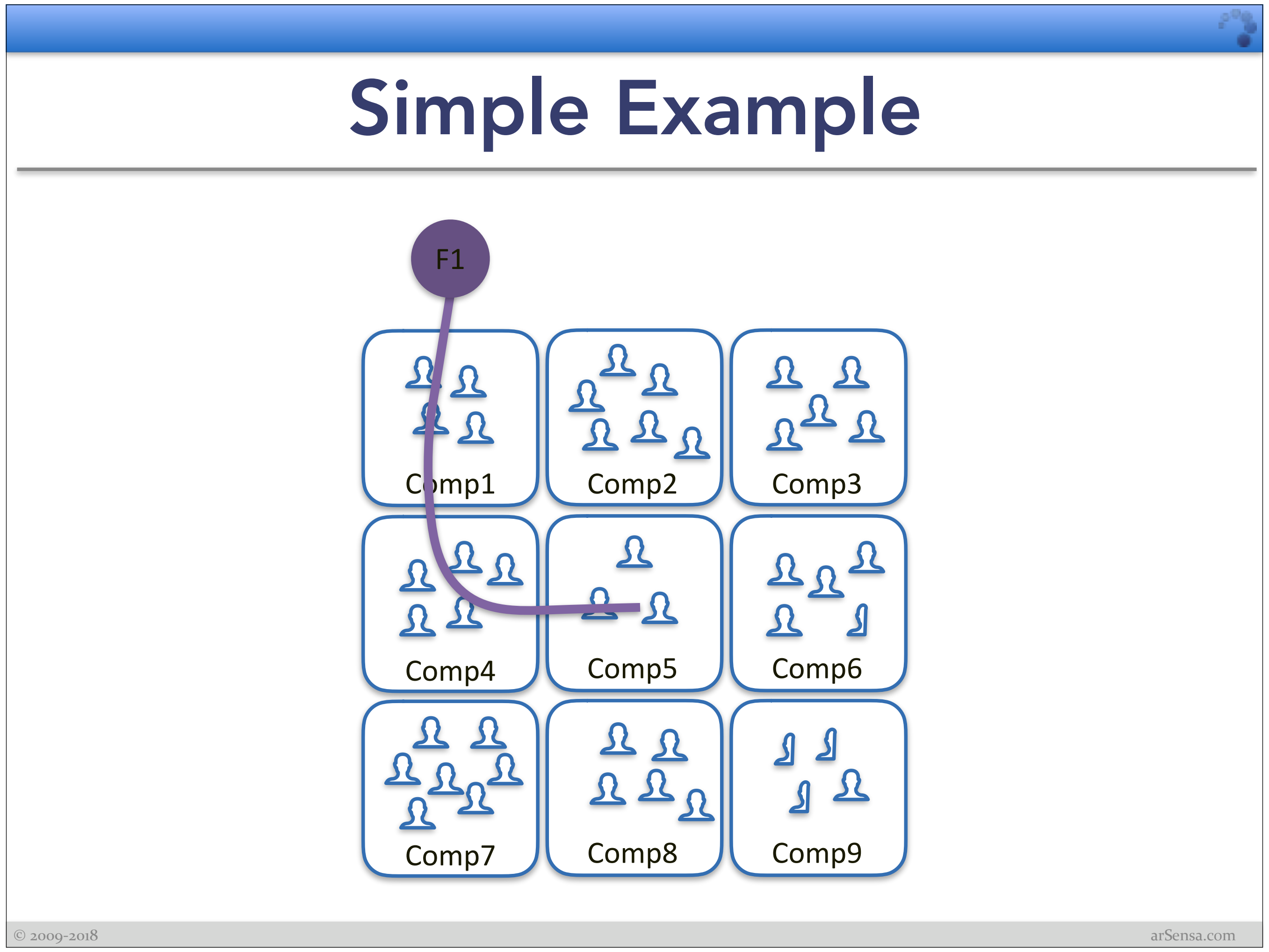
Large organizations often have a significant number of components. There isn’t always a 1:1 relationship between component and team, but the organizational basic structures often reflect the component structure.

Craig has worked with organizations that have 100s, if not 1000s of components.

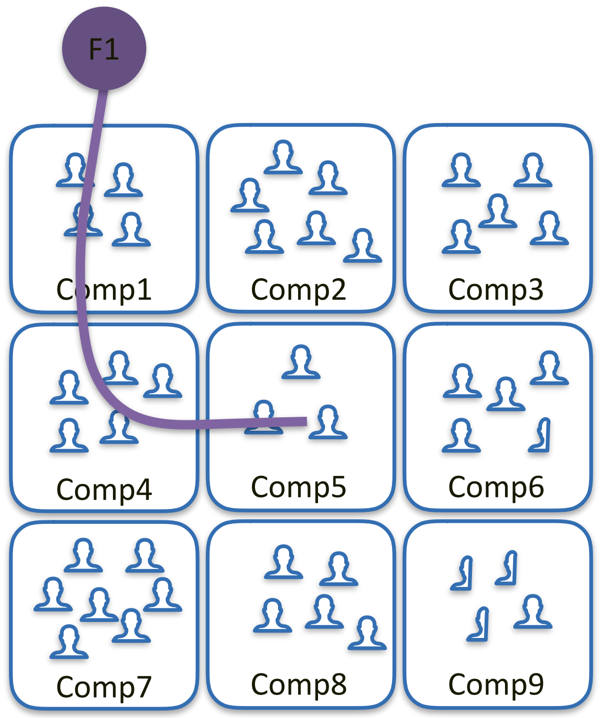
To keep the example simple, consider a system with 9 components.

Consider feature F1. It touches several different components. Now consider feature F2, it also touches several components, some of which are the same components as F1. This causes orchestration problems.

Craig’s diagram went on to make the screen a mess, highlighting what this would mean for 100s of features.



Simple Example



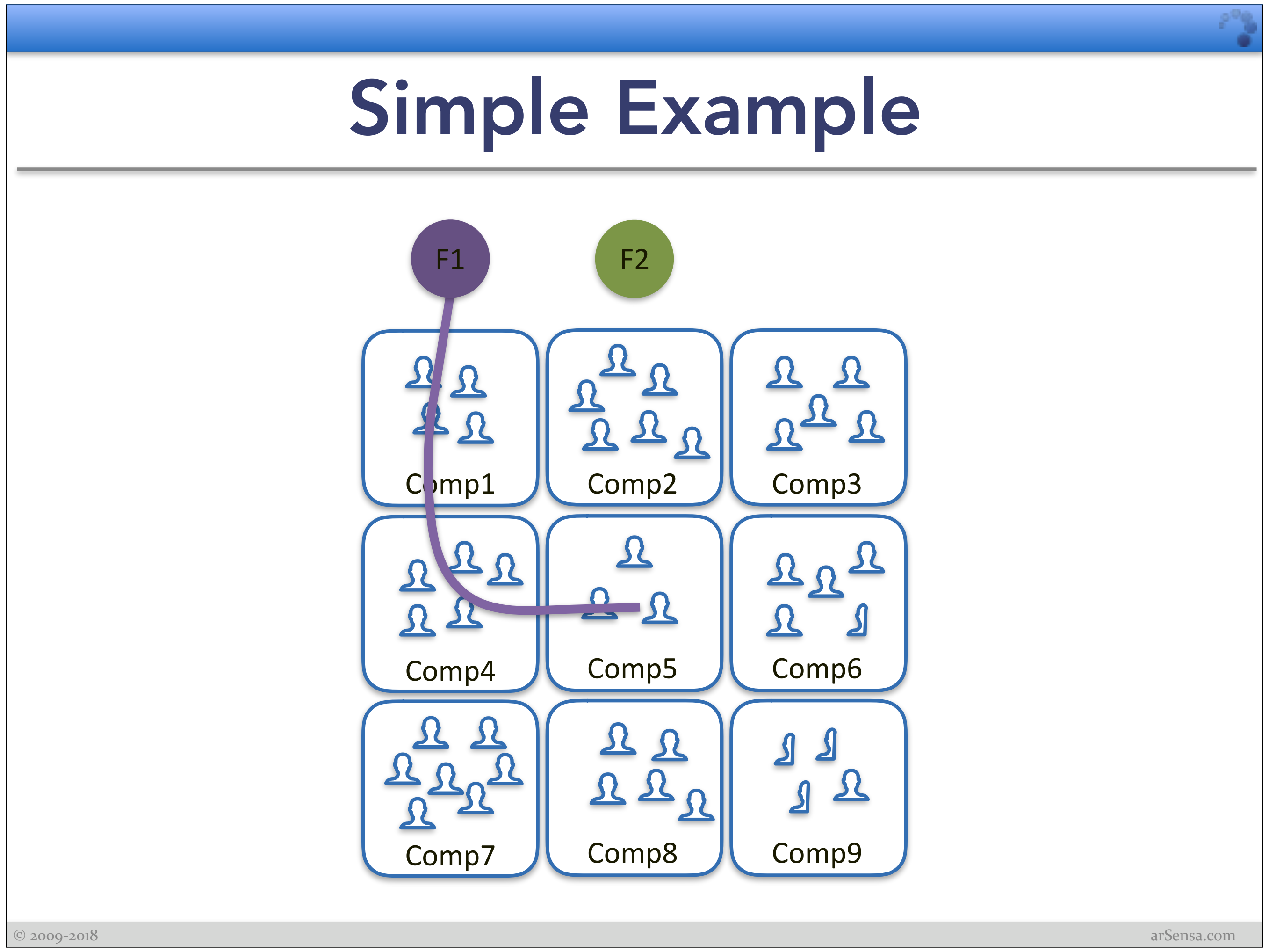
Large organizations often have a significant number of components. There isn't always a 1:1 relationship between component and team, but the organizational basic structures often reflect the component structure.

Craig has worked with organizations that have 100s, if not 1000s of components.

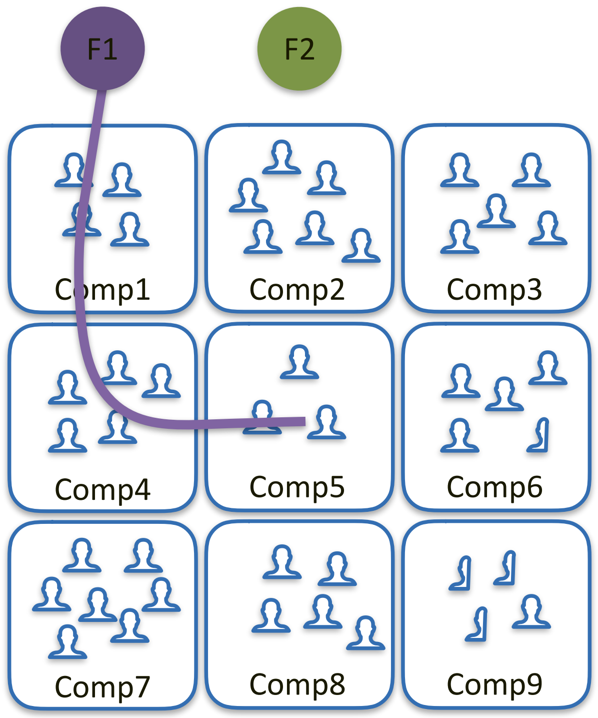
To keep the example simple, consider a system with 9 components.

Consider feature F1. It touches several different components. Now consider feature F2, it also touches several components, some of which are the same components as F1. This causes orchestration problems.

Craig's diagram went on to make the screen a mess, highlighting what this would mean for 100s of features.



Simple Example



Large organizations often have a significant number of components. There isn't always a 1:1 relationship between component and team, but the organizational basic structures often reflect the component structure.

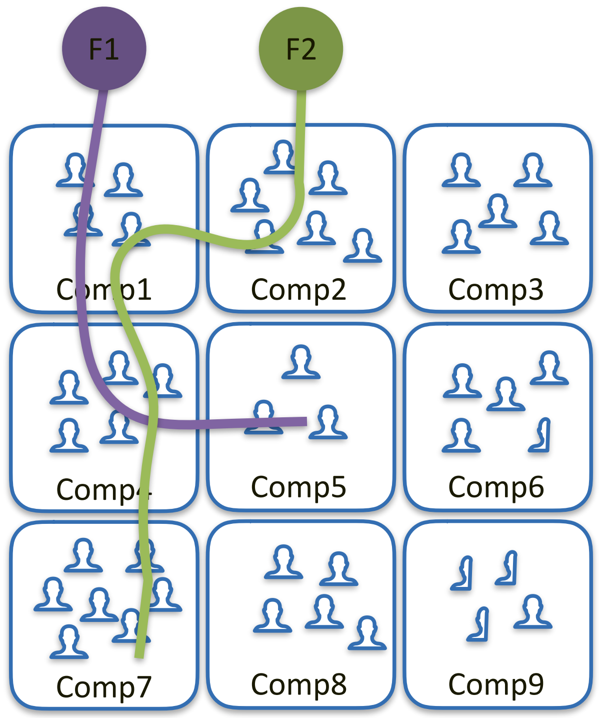
Craig has worked with organizations that have 100s, if not 1000s of components.

To keep the example simple, consider a system with 9 components.

Consider feature F1. It touches several different components. Now consider feature F2, it also touches several components, some of which are the same components as F1. This causes orchestration problems.

Craig's diagram went on to make the screen a mess, highlighting what this would mean for 100s of features.

Simple Example



Large organizations often have a significant number of components. There isn't always a 1:1 relationship between component and team, but the organizational basic structures often reflect the component structure.

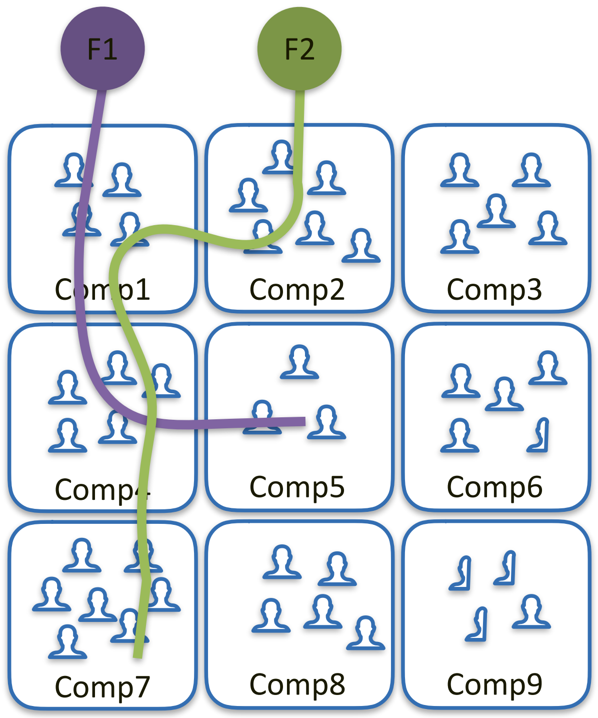
Craig has worked with organizations that have 100s, if not 1000s of components.

To keep the example simple, consider a system with 9 components.

Consider feature F1. It touches several different components. Now consider feature F2, it also touches several components, some of which are the same components as F1. This causes orchestration problems.

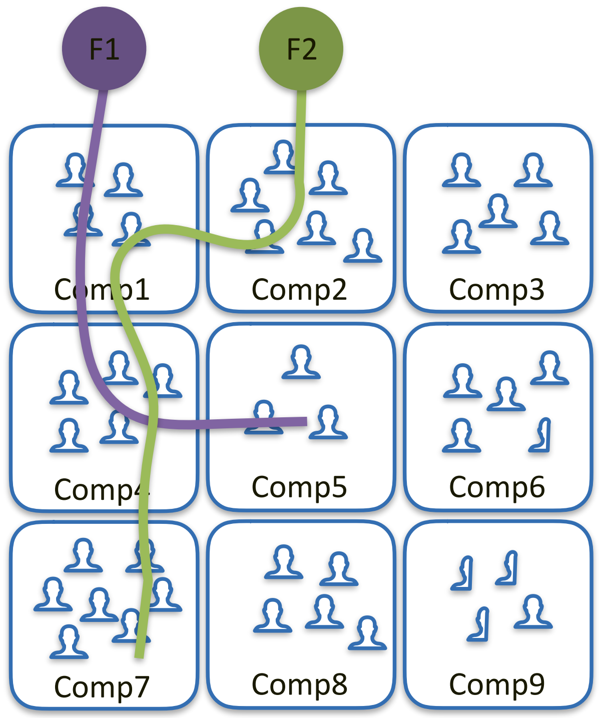
Craig's diagram went on to make the screen a mess, highlighting what this would mean for 100s of features.

Simple Example



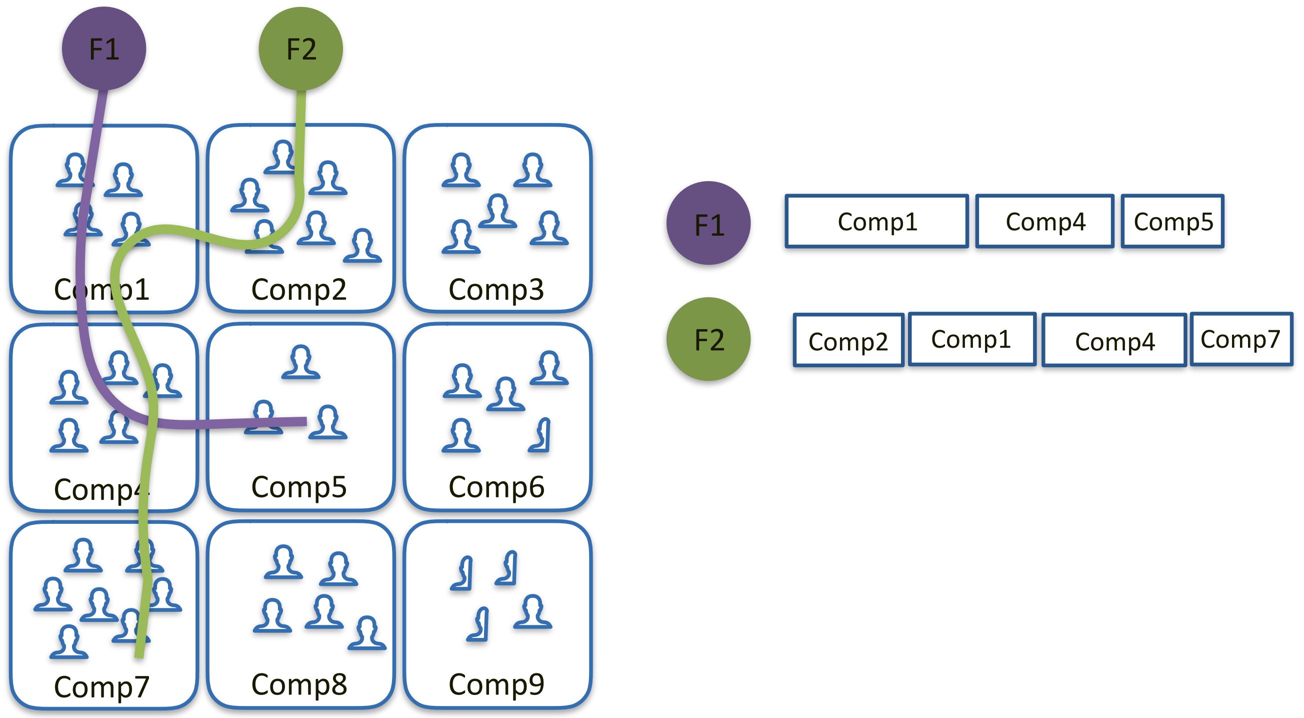
Assume that dependancies mean that the components must be worked on in order of the line threading through them. Consider that there is no relationship between the component, the feature and the amount of time being spent — just because different features both require work on the same component does not mean the amount of work in the component will be consistent.

Simple Example



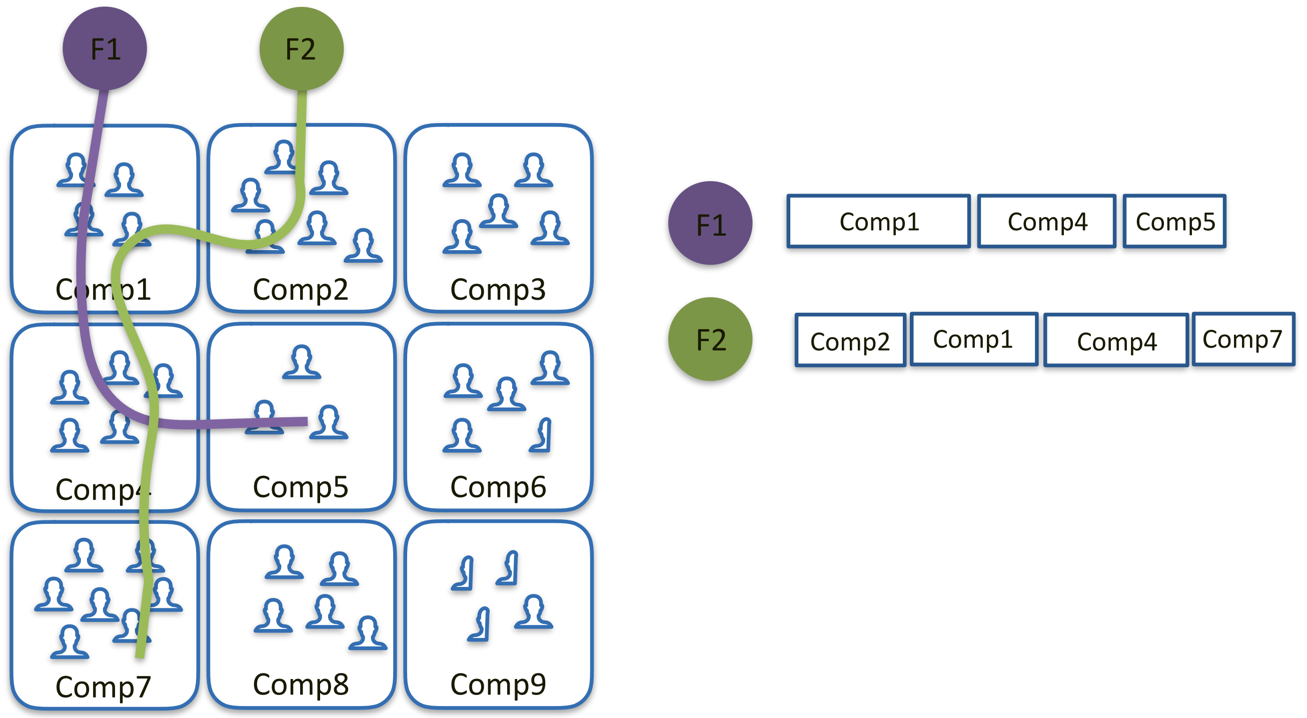
Assume that dependancies mean that the components must be worked on in order of the line threading through them. Consider that there is no relationship between the component, the feature and the amount of time being spent — just because different features both require work on the same component does not mean the amount of work in the component will be consistent.

Simple Example



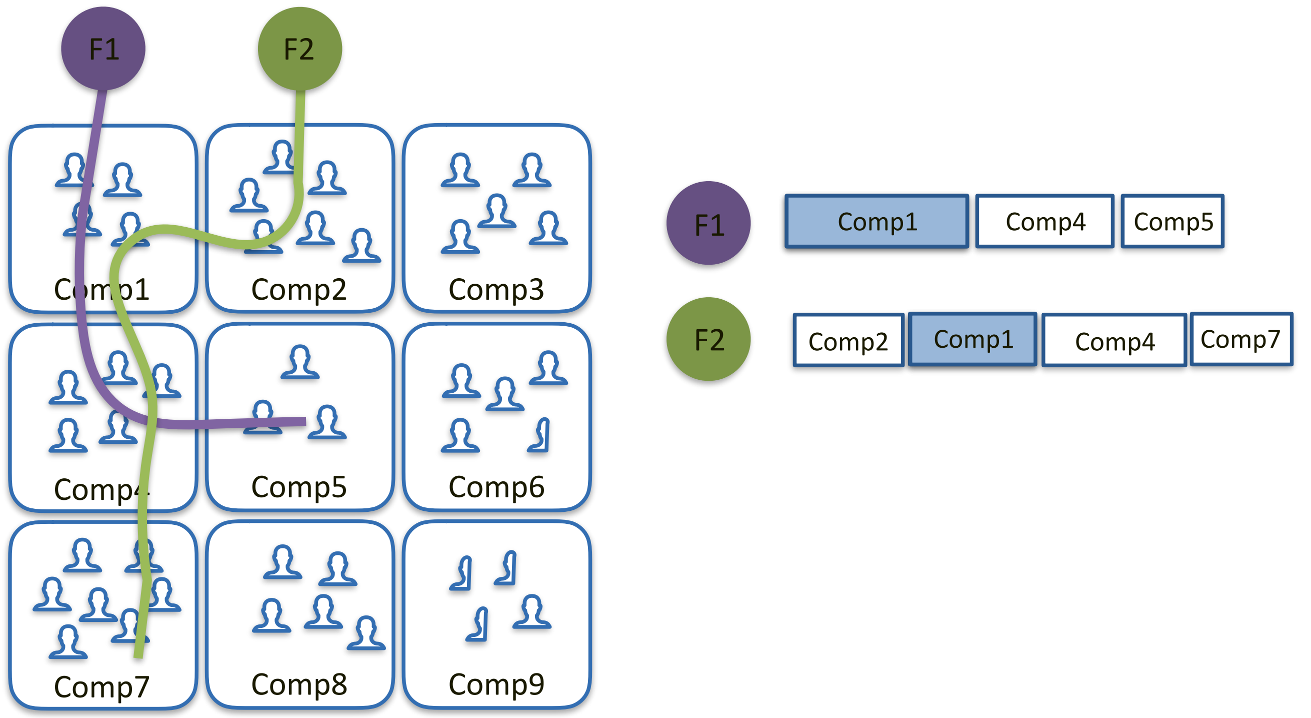
Assume that dependancies mean that the components must be worked on in order of the line threading through them. Consider that there is no relationship between the component, the feature and the amount of time being spent — just because different features both require work on the same component does not mean the amount of work in the component will be consistent.

Simple Example

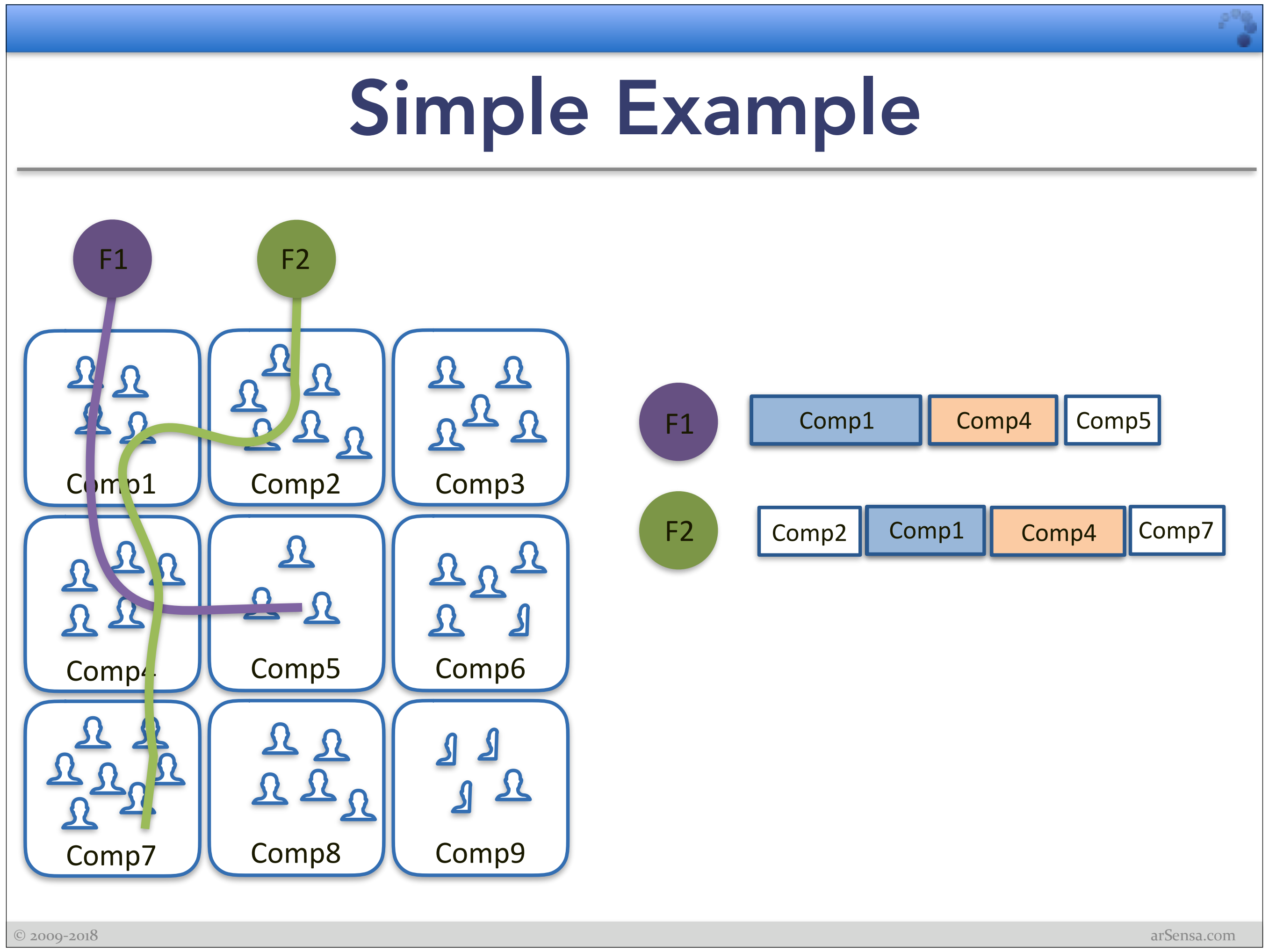


There is now a scheduling problem for F1 & F2 because they touch components 1 & 4. If a team only works on a single feature at a time, we have a gapping problem. If the teams work on multiple features at a time we have a co-ordination problem.

Simple Example

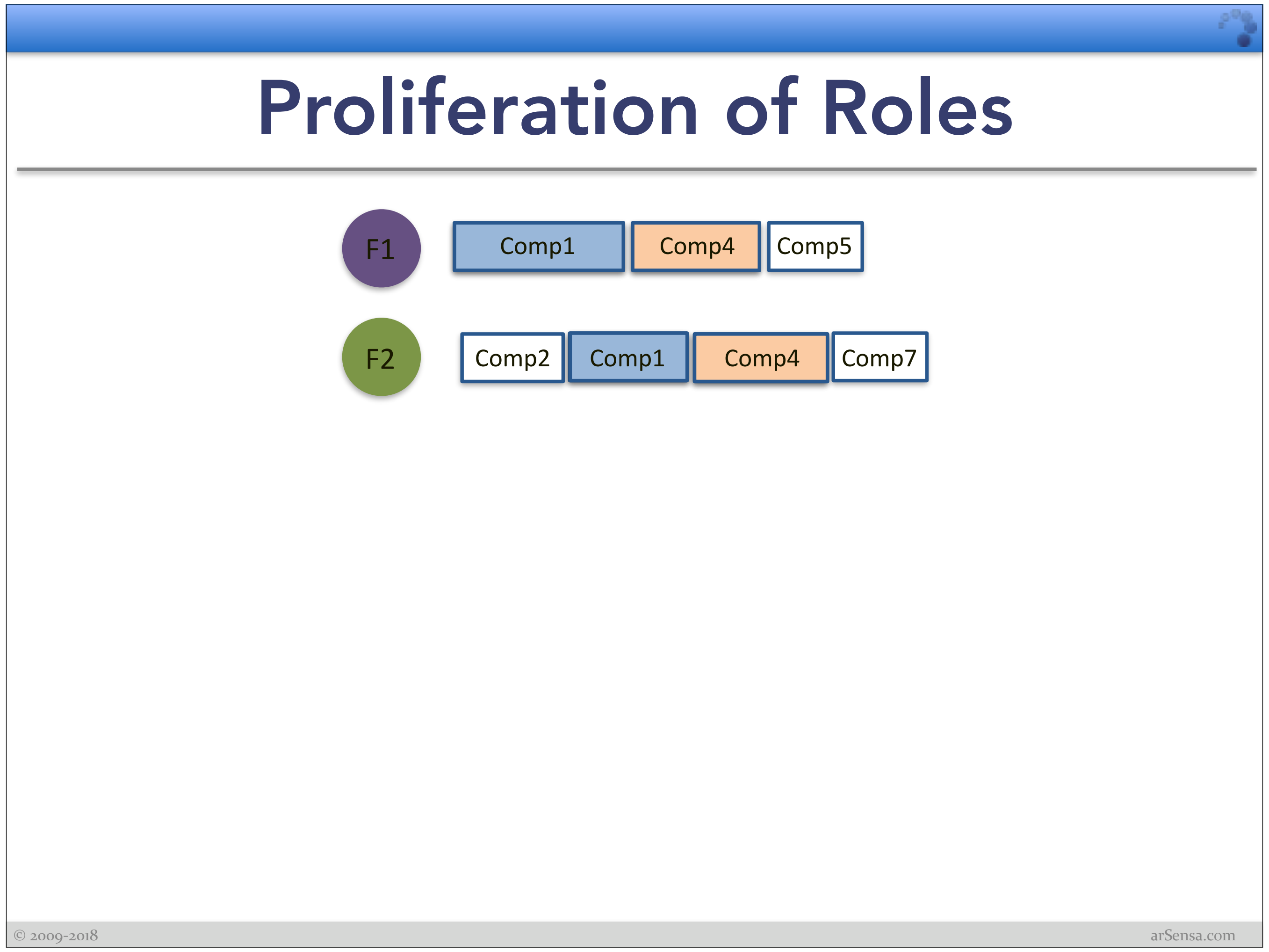


There is now a scheduling problem for F1 & F2 because they touch components 1 & 4. If a team only works on a single feature at a time, we have a gapping problem. If the teams work on multiple features at a time we have a co-ordination problem.

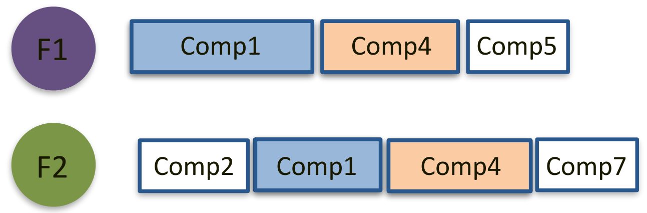


Simple Example

There is now a scheduling problem for F1 & F2 because they touch components 1 & 4. If a team only works on a single feature at a time, we have a gapping problem. If the teams work on multiple features at a time we have a co-ordination problem.



Proliferation of Roles



In traditional software development, this problem is addressed by introducing roles to help with co-ordination.

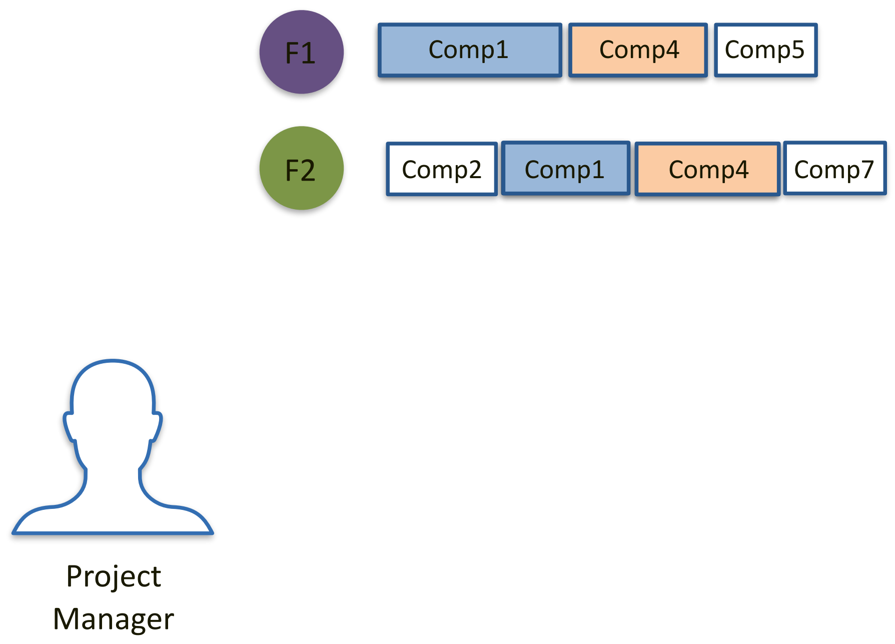
Project managers operate at the feature level and attempt co-ordination between the component teams. It is difficult for them to schedule work and any delay impacts other features and other component teams.

There is an additional problem about how the features' requirements within a given component interact with each other. To help with this problem organizations often introduce a Technical BA role to manage and coordinate low level requirements across features and components.

Large systems with many components means a lot of potential interactions. Organizations introduce Architects to deal with this problem.

There are often large gaps of time when a feature might not be actively worked on due to unavailability of a component team. Further, integration testing can't be completed until all components are delivered. This introduces complexities in testing.

Proliferation of Roles



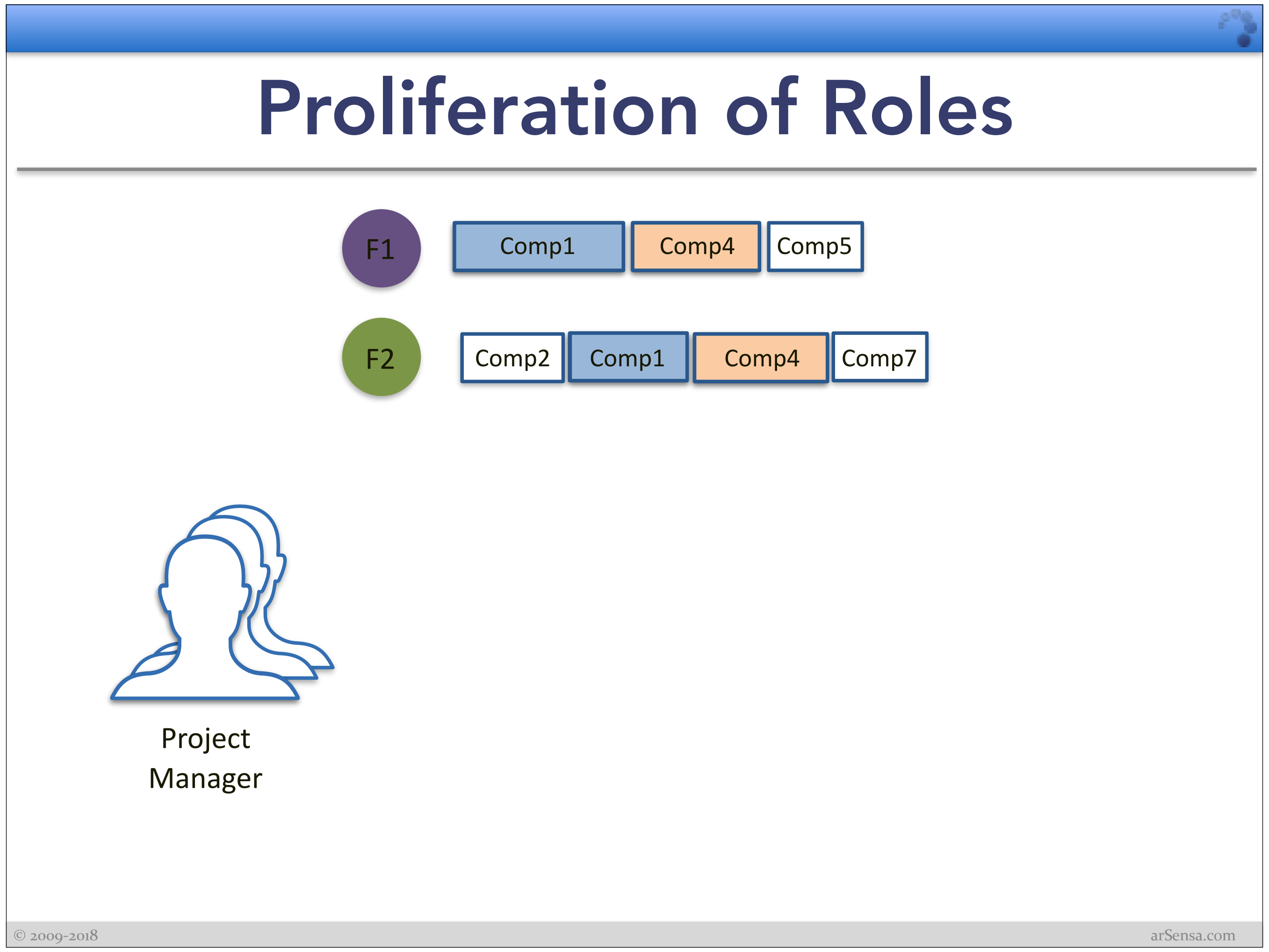
In traditional software development, this problem is addressed by introducing roles to help with co-ordination.

Project managers operate at the feature level and attempt co-ordination between the component teams. It is difficult for them to schedule work and any delay impacts other features and other component teams.

There is an additional problem about how the features’ requirements within a given component interact with each other. To help with this problem organizations often introduce a Technical BA role to manage and coordinate low level requirements across features and components.

Large systems with many components means a lot of potential interactions. Organizations introduce Architects to deal with this problem.

There are often large gaps of time when a feature might not be actively worked on due to unavailability of a component team. Further, integration testing can’t be completed until all components are delivered. This introduces complexities in testing.



Proliferation of Roles

In traditional software development, this problem is addressed by introducing roles to help with co-ordination.

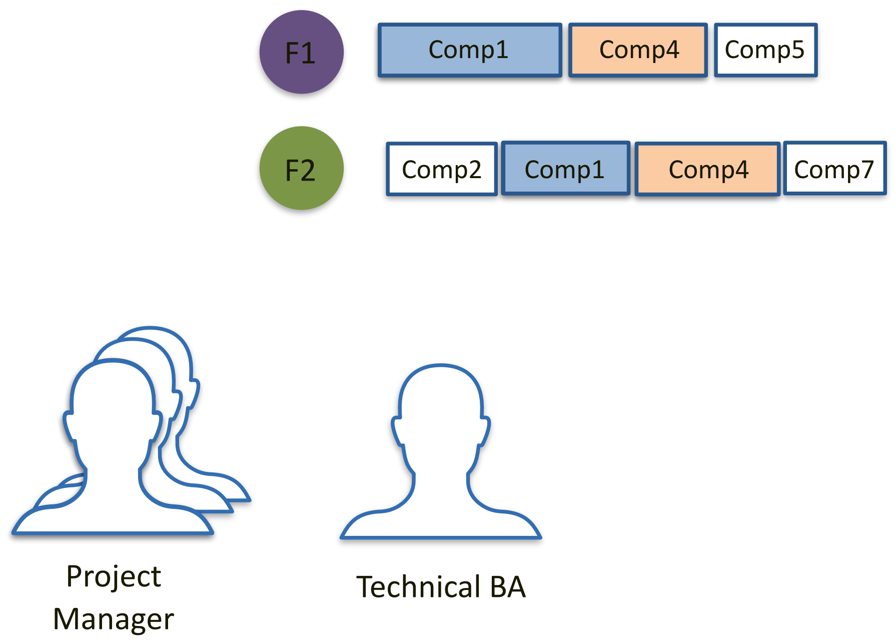
Project managers operate at the feature level and attempt co-ordination between the component teams. It is difficult for them to schedule work and any delay impacts other features and other component teams.

There is an additional problem about how the features' requirements within a given component interact with each other. To help with this problem organizations often introduce a Technical BA role to manage and coordinate low level requirements across features and components.

Large systems with many components means a lot of potential interactions. Organizations introduce Architects to deal with this problem.

There are often large gaps of time when a feature might not be actively worked on due to unavailability of a component team. Further, integration testing can't be completed until all components are delivered. This introduces complexities in testing.

Proliferation of Roles



In traditional software development, this problem is addressed by introducing roles to help with co-ordination.

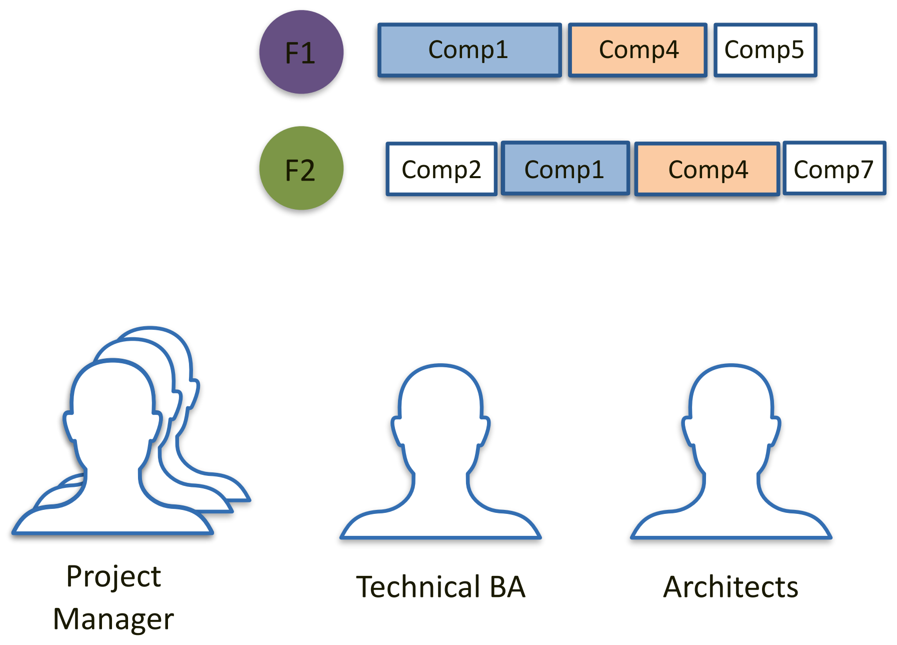
Project managers operate at the feature level and attempt co-ordination between the component teams. It is difficult for them to schedule work and any delay impacts other features and other component teams.

There is an additional problem about how the features' requirements within a given component interact with each other. To help with this problem organizations often introduce a Technical BA role to manage and coordinate low level requirements across features and components.

Large systems with many components means a lot of potential interactions. Organizations introduce Architects to deal with this problem.

There are often large gaps of time when a feature might not be actively worked on due to unavailability of a component team. Further, integration testing can't be completed until all components are delivered. This introduces complexities in testing.

Proliferation of Roles



In traditional software development, this problem is addressed by introducing roles to help with co-ordination.

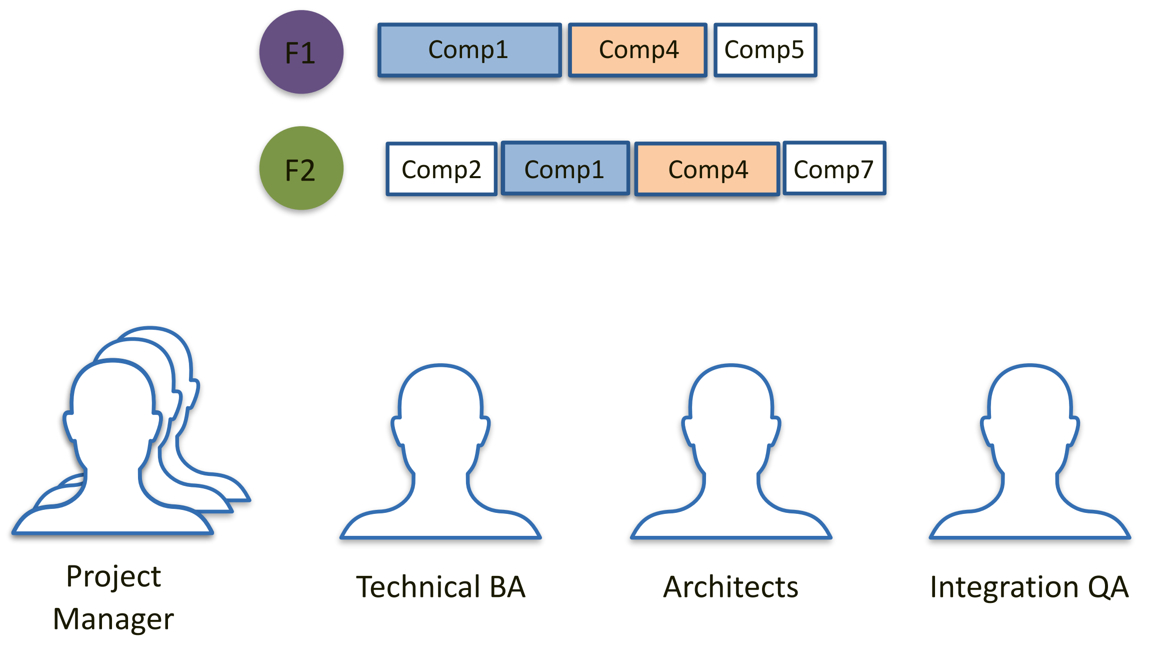
Project managers operate at the feature level and attempt co-ordination between the component teams. It is difficult for them to schedule work and any delay impacts other features and other component teams.

There is an additional problem about how the features' requirements within a given component interact with each other. To help with this problem organizations often introduce a Technical BA role to manage and coordinate low level requirements across features and components.

Large systems with many components means a lot of potential interactions. Organizations introduce Architects to deal with this problem.

There are often large gaps of time when a feature might not be actively worked on due to unavailability of a component team. Further, integration testing can't be completed until all components are delivered. This introduces complexities in testing.

Proliferation of Roles



In traditional software development, this problem is addressed by introducing roles to help with co-ordination.

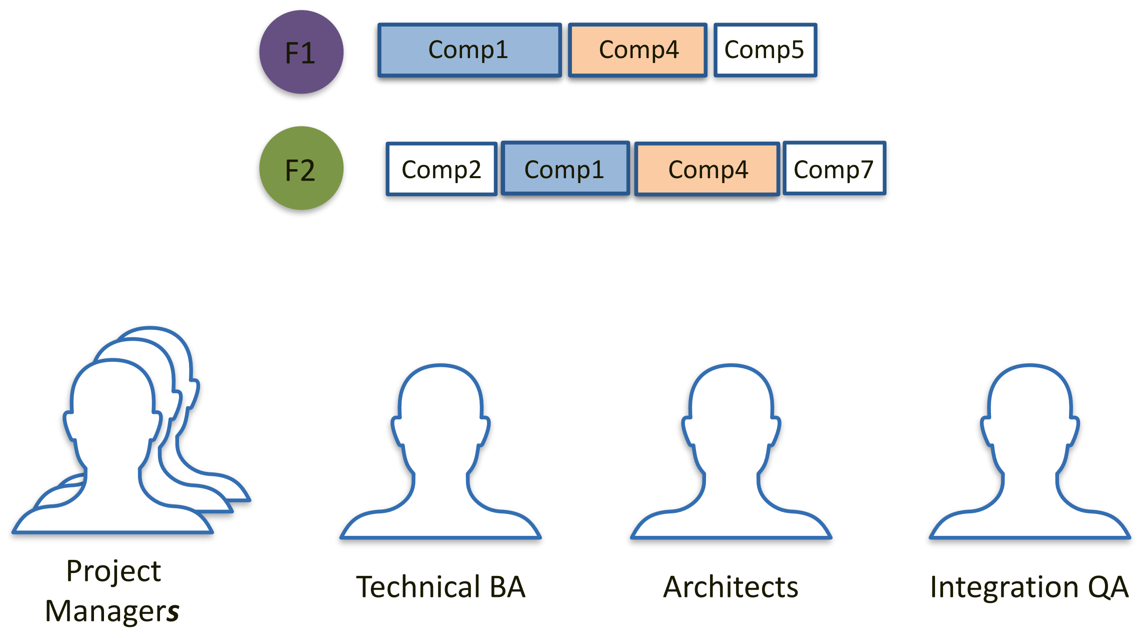
Project managers operate at the feature level and attempt co-ordination between the component teams. It is difficult for them to schedule work and any delay impacts other features and other component teams.

There is an additional problem about how the features' requirements within a given component interact with each other. To help with this problem organizations often introduce a Technical BA role to manage and coordinate low level requirements across features and components.

Large systems with many components means a lot of potential interactions. Organizations introduce Architects to deal with this problem.

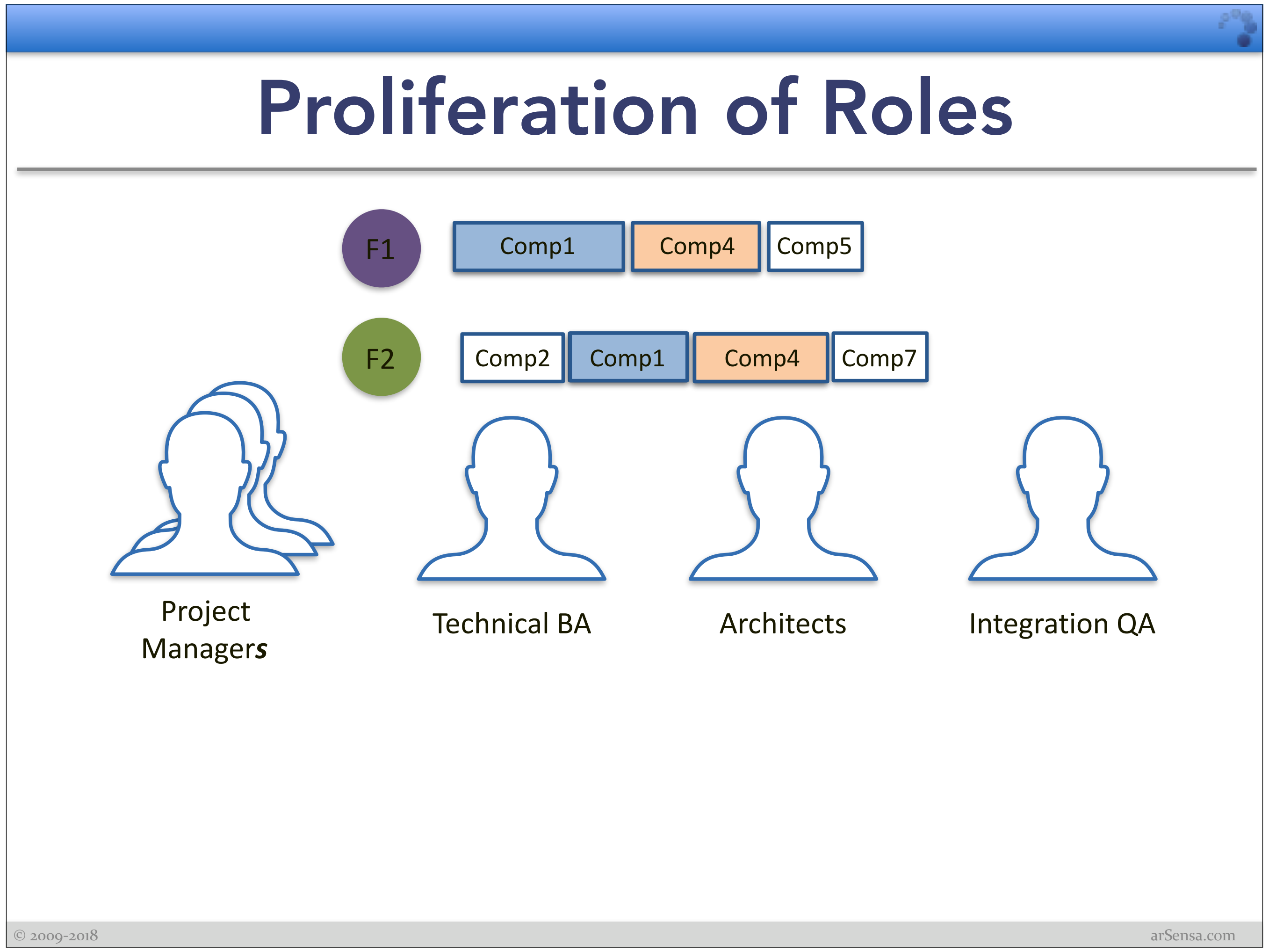
There are often large gaps of time when a feature might not be actively worked on due to unavailability of a component team. Further, integration testing can't be completed until all components are delivered. This introduces complexities in testing.

Proliferation of Roles



Gluing all the components together into a release, coordinating features together and managing the code now becomes a full-time job. The Release Engineer helps coordinate.

Of course, all these additional people means that we need managers to manage them all.

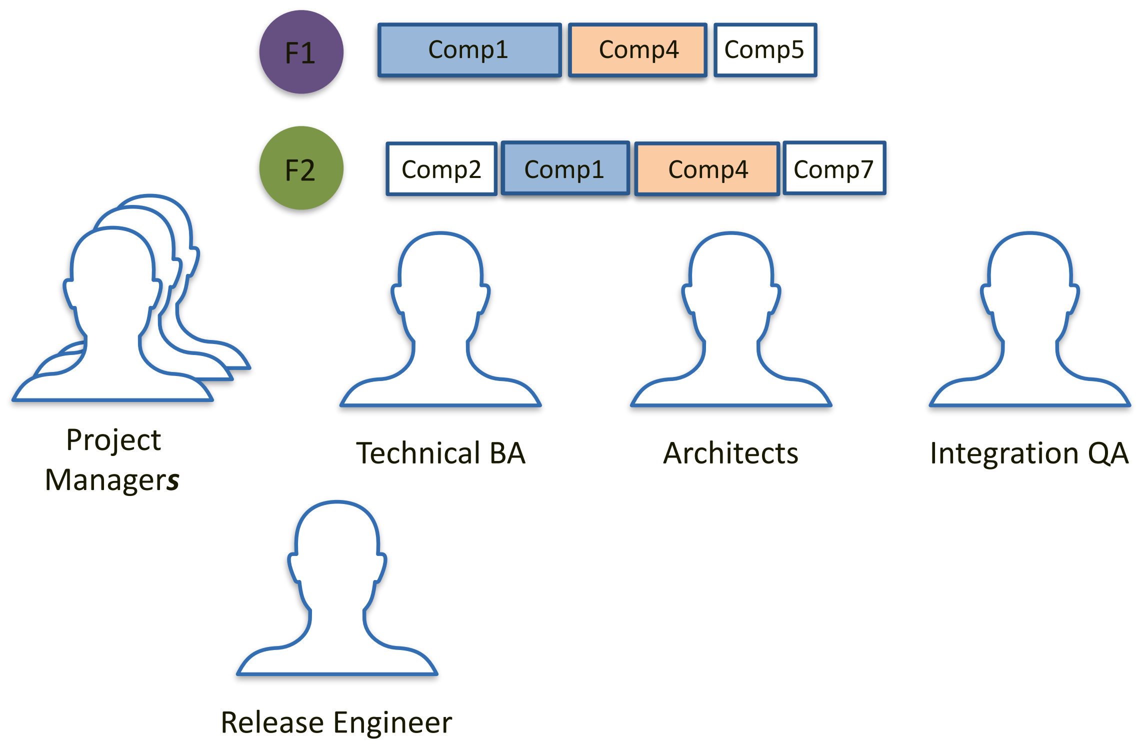


Proliferation of Roles

Gluing all the components together into a release, coordinating features together and managing the code now becomes a full-time job. The Release Engineer helps coordinate.

Of course, all these additional people means that we need managers to manage them all.

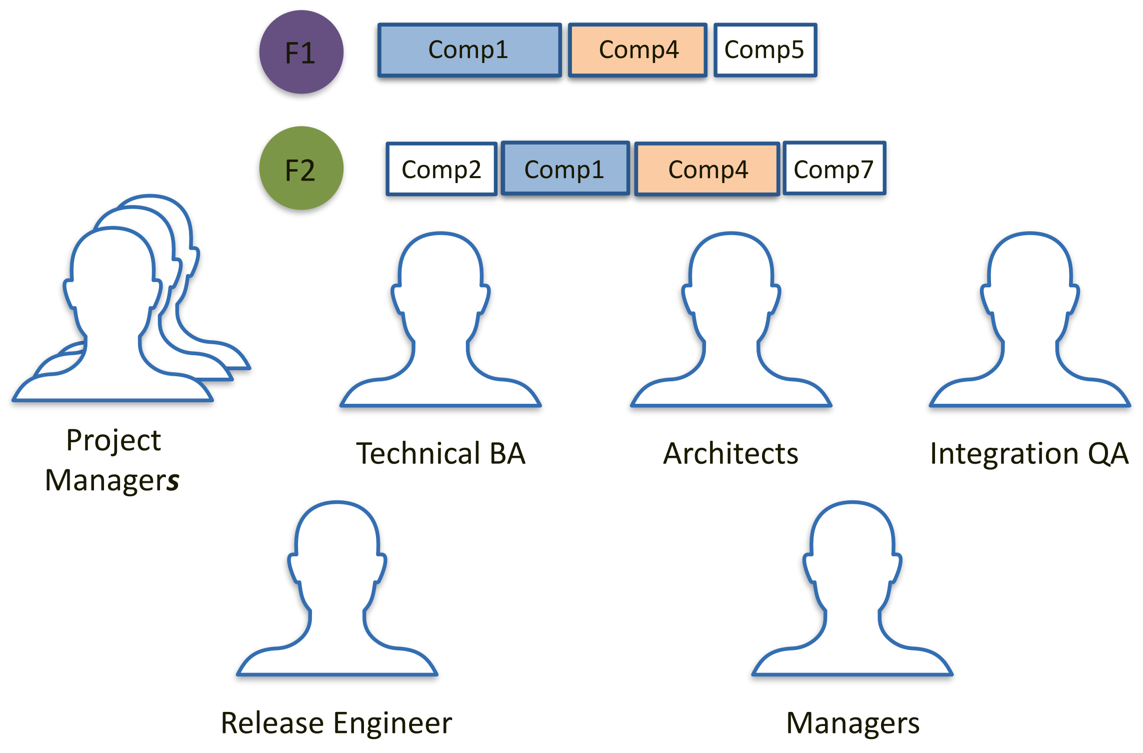
Proliferation of Roles



Gluing all the components together into a release, coordinating features together and managing the code now becomes a full-time job. The Release Engineer helps coordinate.

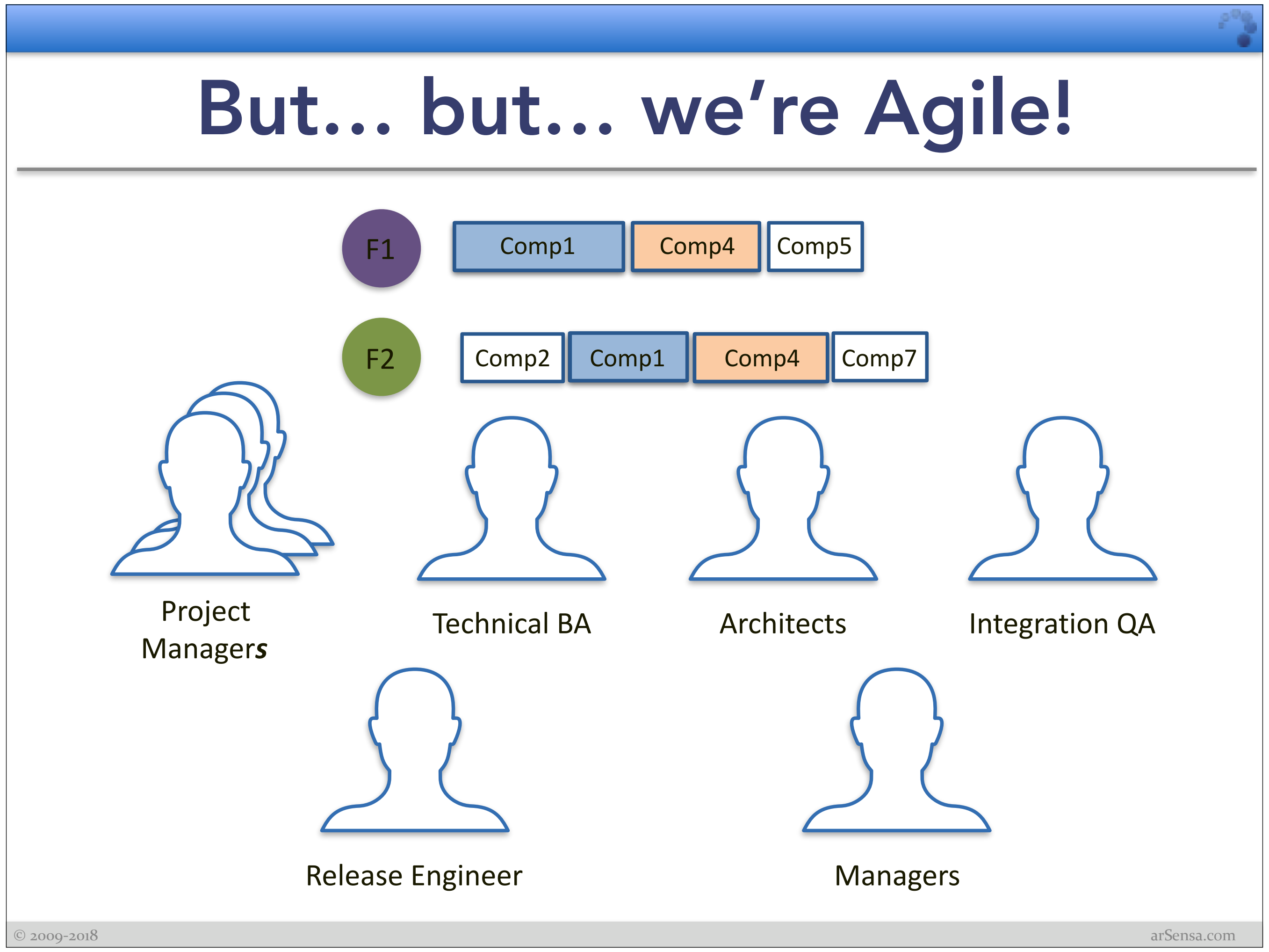
Of course, all these additional people means that we need managers to manage them all.

Proliferation of Roles



Gluing all the components together into a release, coordinating features together and managing the code now becomes a full-time job. The Release Engineer helps coordinate.

Of course, all these additional people means that we need managers to manage them all.

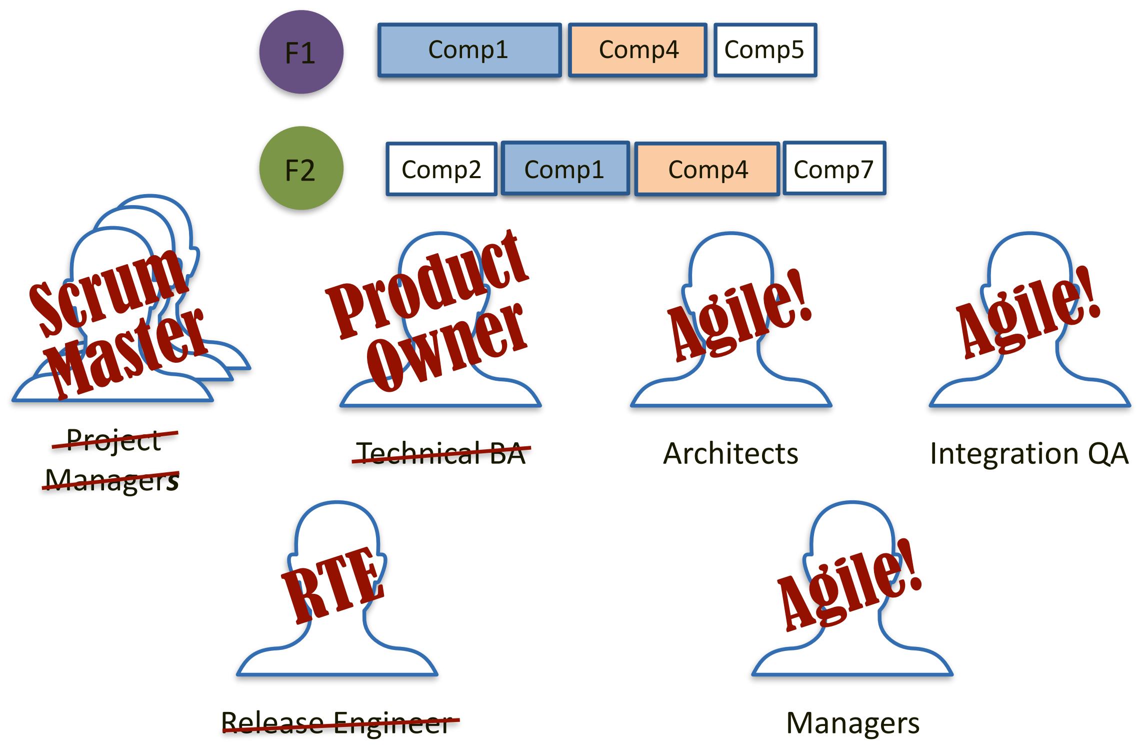


But… but… we're Agile!

Many organizations that go through an Agile Transformation build scrum teams around the teams already in place. This minimizes the disruption to the teams and requires the least amount of effort.

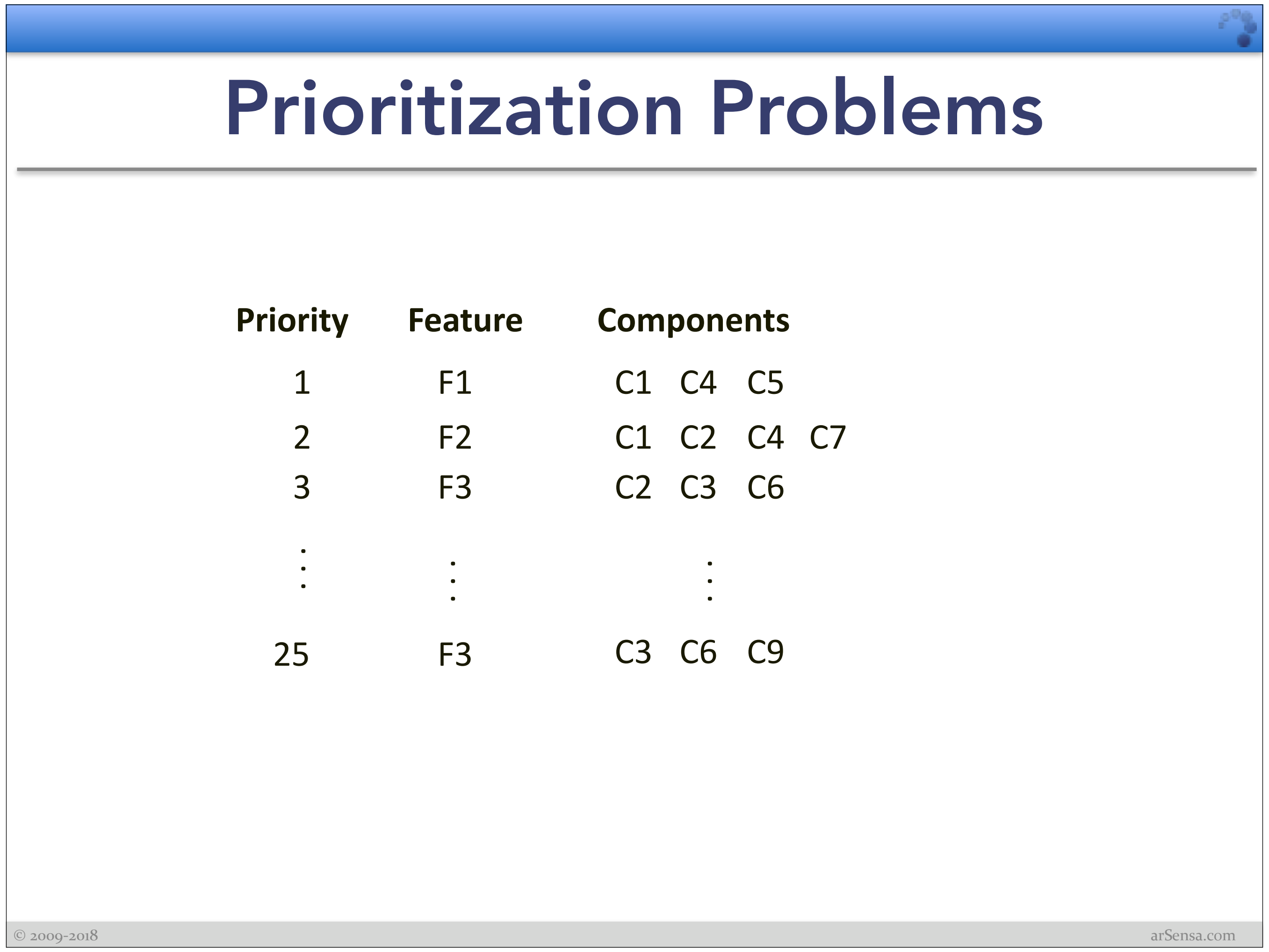
The problem caused by the component teams is thus not fixed by going Agile. Some increase in productivity may be achieved, but the fundamental assumptions of the original organization are still the primary bottlenecks.

But… but… we're Agile!



Many organizations that go through an Agile Transformation build scrum teams around the teams already in place. This minimizes the disruption to the teams and requires the least amount of effort.

The problem caused by the component teams is thus not fixed by going Agile. Some increase in productivity may be achieved, but the fundamental assumptions of the original organization are still the primary bottlenecks.

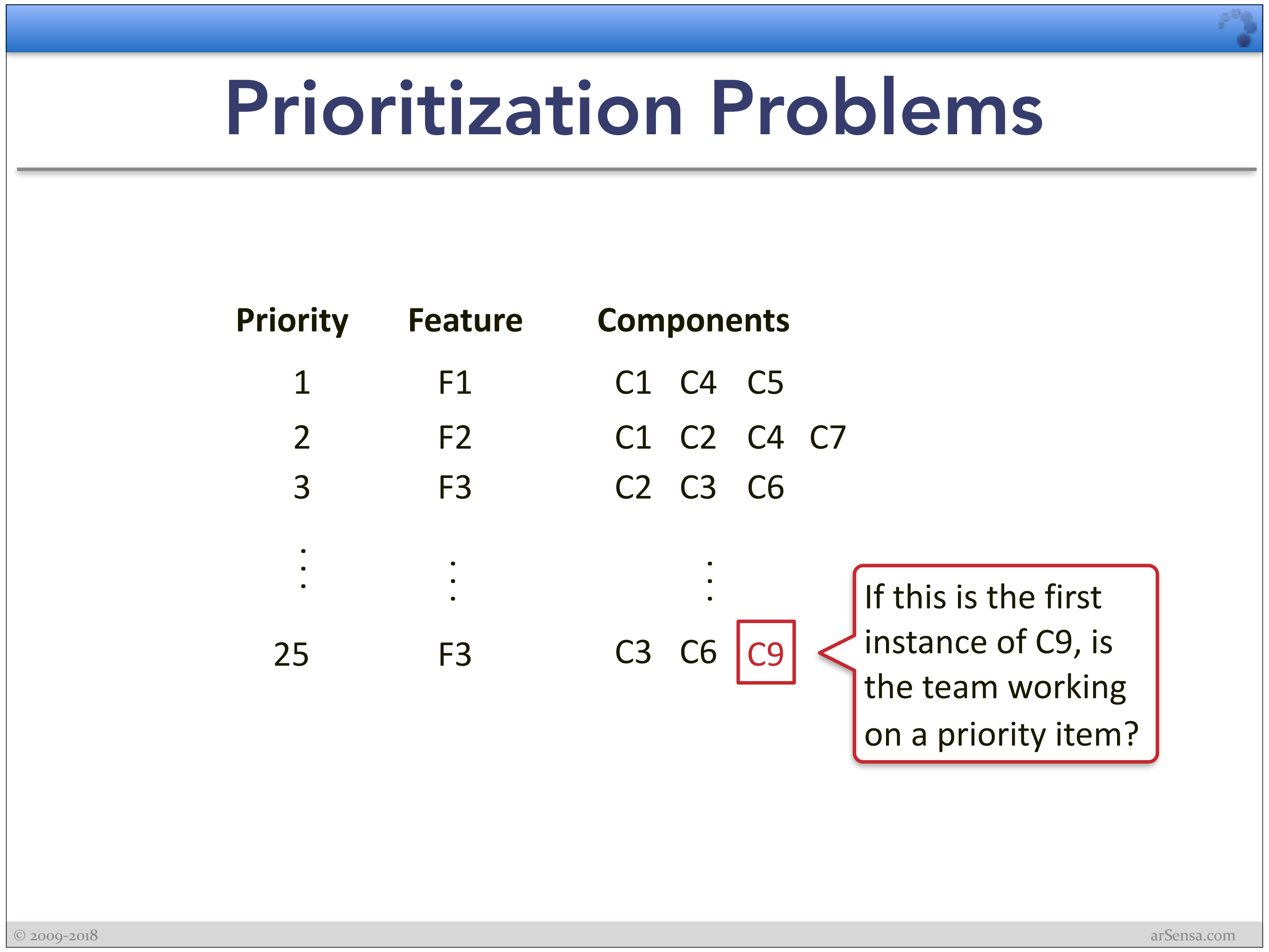


Prioritization Problems

Priority	Feature	Components
1	F1	C1 C4 C5
2	F2	C1 C2 C4 C7
3	F3	C2 C3 C6
⋮	⋮	⋮
25	F3	C3 C6 C9

Component based team structure is optimized for utilization of people. This is counter to being optimized for speed of delivery or simplicity of organization.

It also is not optimized for prioritized delivery. Consider a list of priorities where the team for component 9 is not involved until near the bottom of the priority list. Either the team has significant slack time, or more typically they start work — work that has 24 things ahead of it that is more important.



Prioritization Problems

Priority	Feature	Components
1	F1	C1 C4 C5
2	F2	C1 C2 C4 C7
3	F3	C2 C3 C6
⋮	⋮	⋮
25	F3	C3 C6 $\boxed{C9}$

If this is the first instance of C9, is the team working on a priority item?

Component based team structure is optimized for utilization of people. This is counter to being optimized for speed of delivery or simplicity of organization.

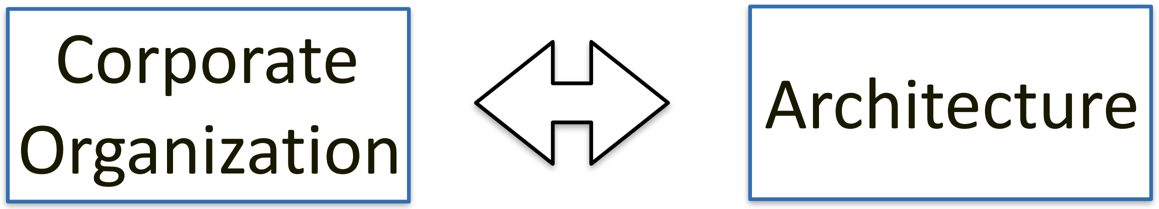
It also is not optimized for prioritized delivery. Consider a list of priorities where the team for component 9 is not involved until near the bottom of the priority list. Either the team has significant slack time, or more typically they start work — work that has 24 things ahead of it that is more important.

Conway's Law: Organizations that design systems are constrained to produce designs which are copies of the communication structures of these organizations.

Melvin Conway is a computer scientist who presented a paper on modular design to the 1968 Symposium on Modular Design. The participants at the symposium dubbed the core idea "Conway's Law"

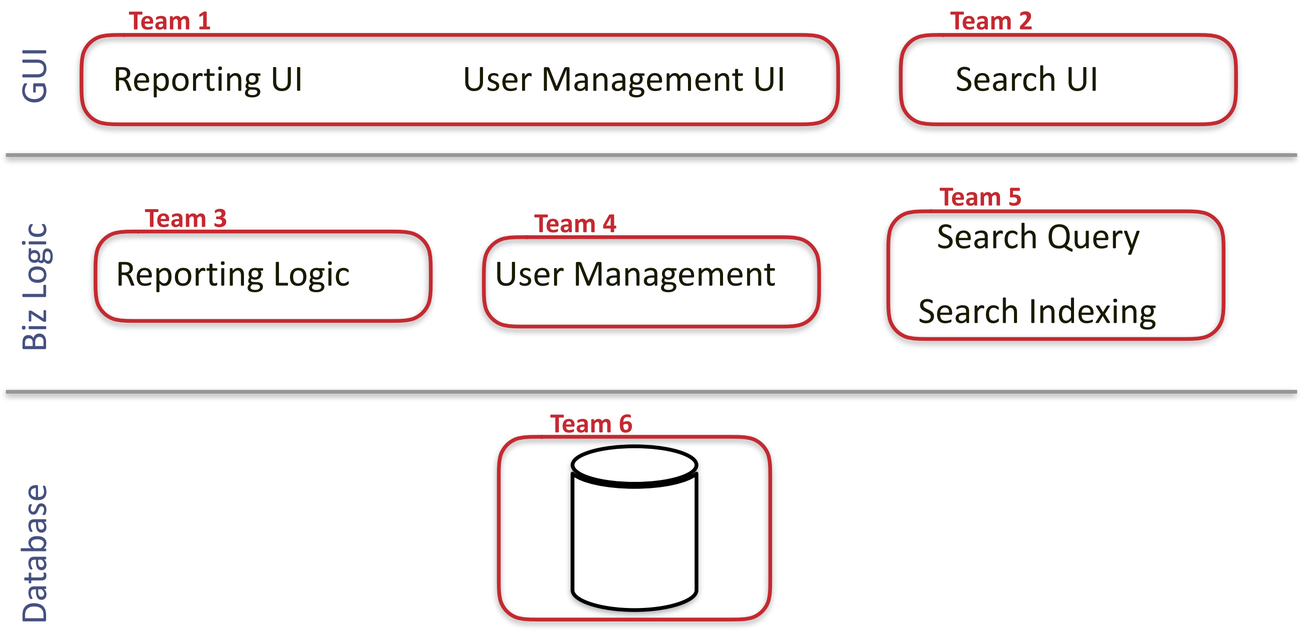
(Different Conway than the Conway's Game of Life… computer science has a lot of Conways)

In other words…



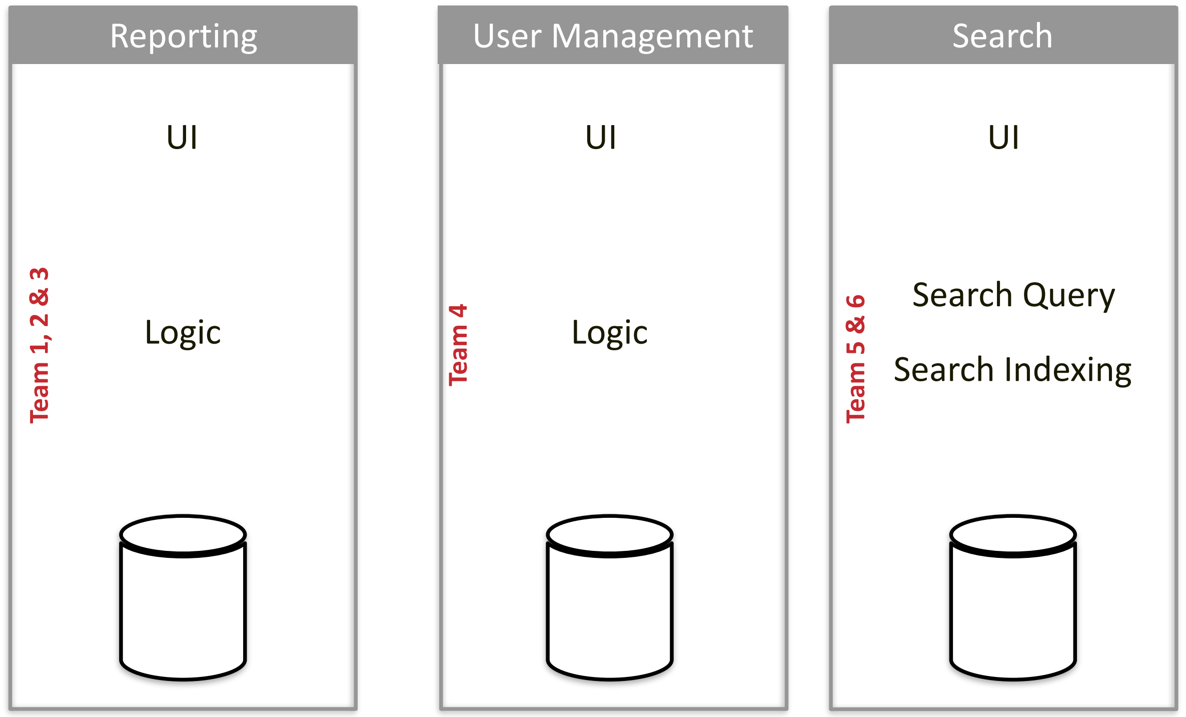
Organizations’ hierarchies accidentally influence their architecture. This is a side-effect of component teams designing their own pieces and naturally forming interaction points to other departments.

From this…



GUI
Team 1
Reporting UI
User Management UI
Team 2
Search UI
Biz Logic
Team 3
Reporting Logic
Team 4
User Management
Team 5
Search Query
Search Indexing
Database
Team 6

… to this



The modern architecture answer to this is Feature Based Teams.

Team structure should be based around the business need. Like components, a single team may be responsible for multiple areas. Multiple teams may be responsible for big areas, or multiple teams may be involved when a new large feature area is being built, with a smaller number of teams continuing to enhance it over the life of the product.

This team structure and architectural design concepts around it can be found in Software As A Service and Microservice style architectures. Monolithic systems can also benefit from this mechanism, but the feature boundaries need to be enforced more rigorously — the boundaries are more natural in SaaS and μServices.