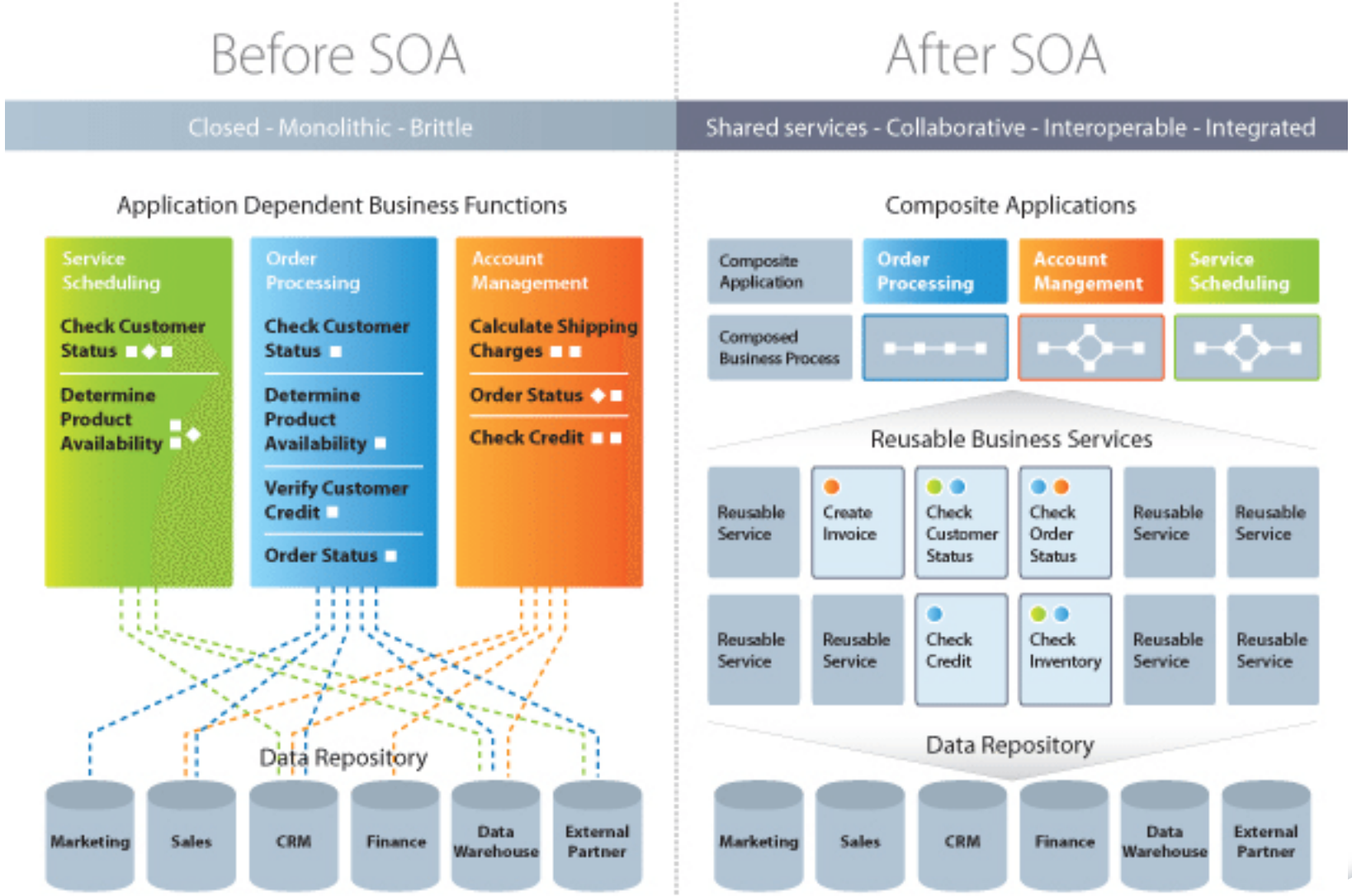
DIT345 Fundamentals of Software Architecture Styles Recap and Tactics

Rebekka Wohlrab

SOA Example

- Move from a set of monolith servers to service-oriented architecture (SOA)
- What are the pros and cons?



https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec

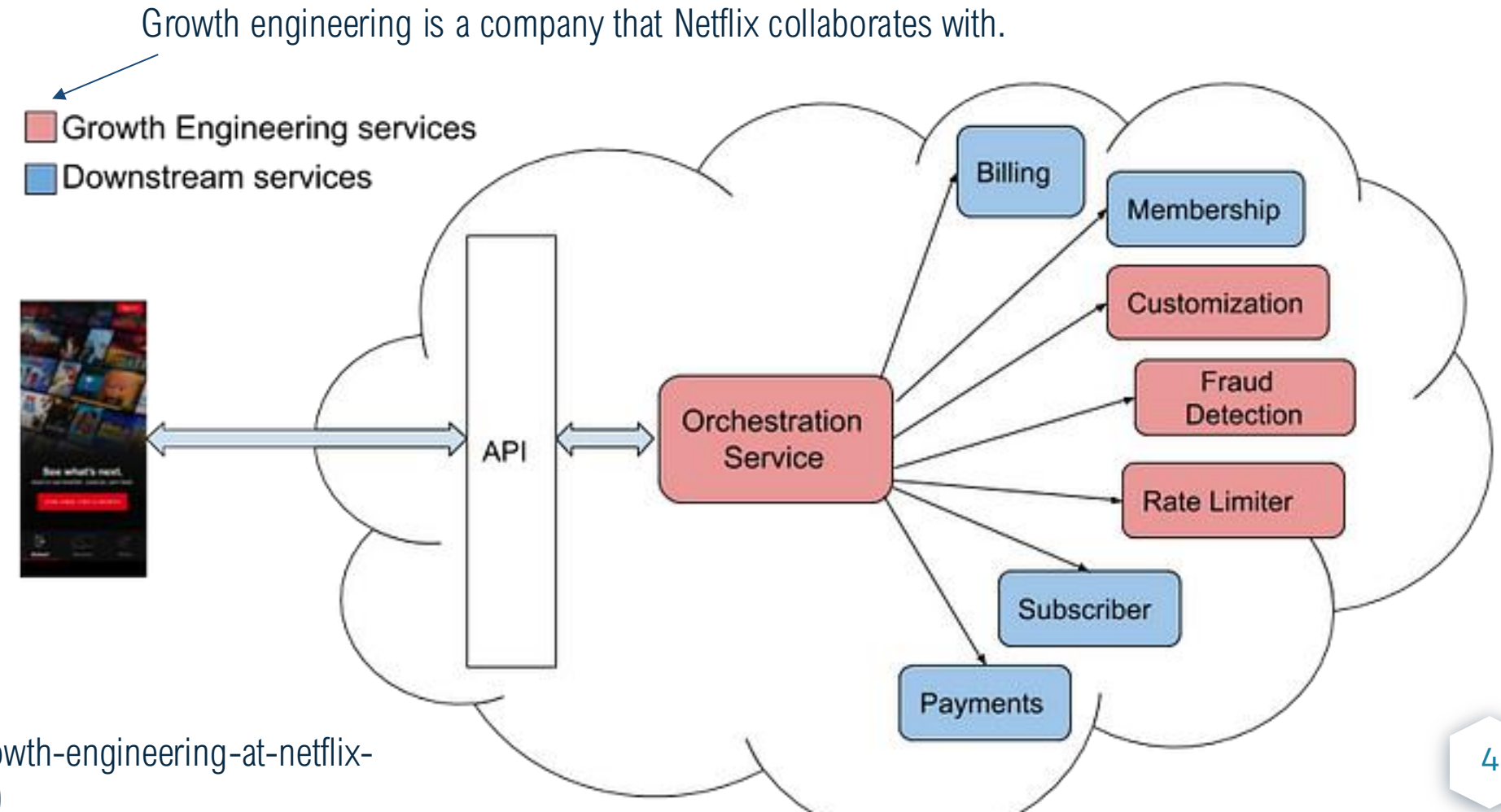
How to design a microservices system?

- **Domain-driven decomposition/design**:
  Business concerns and features drive the decomposition
- Each business concern or feature becomes a **microservice**

- Need to understand **what the main business goals/features** are!

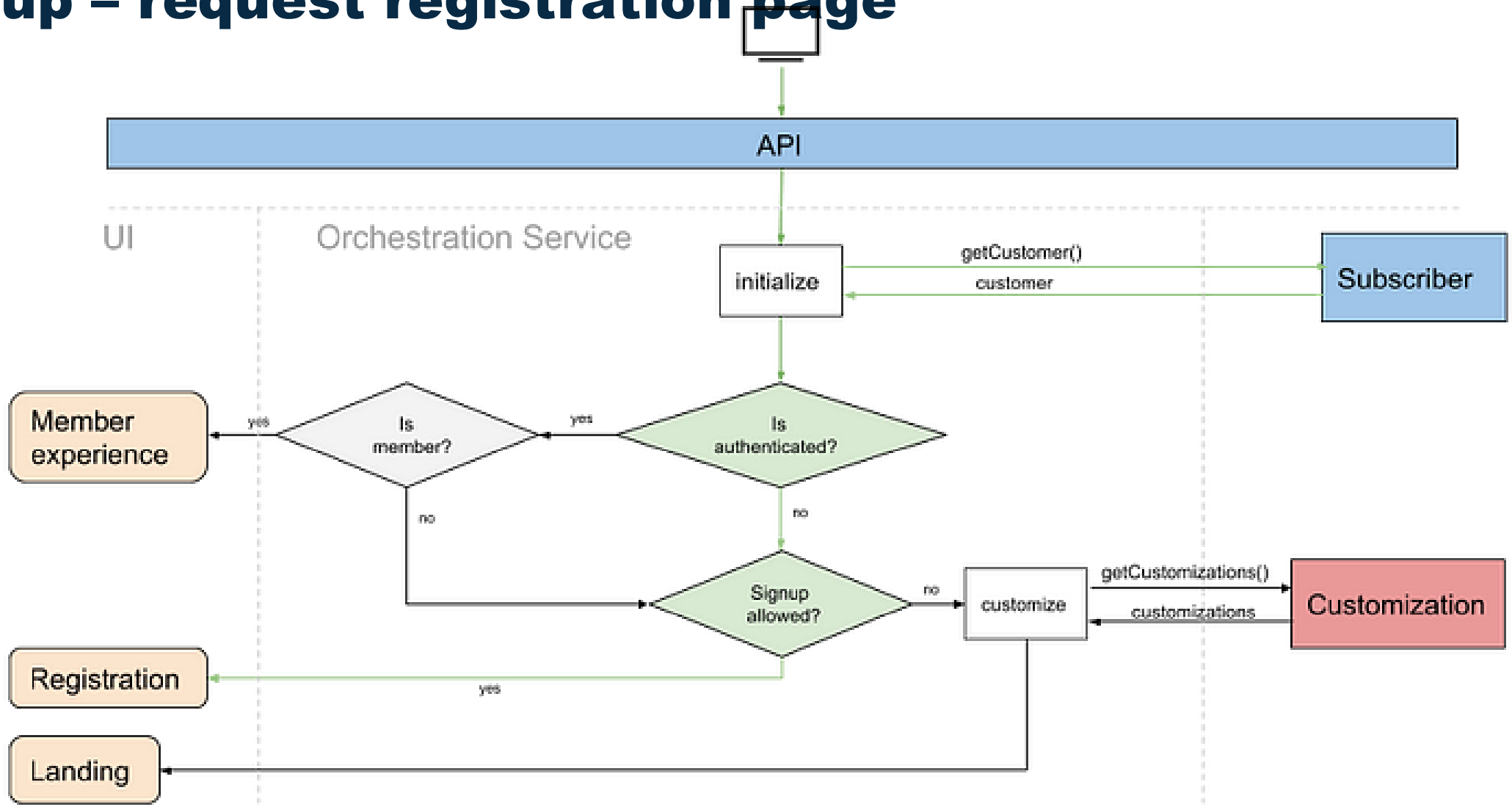
- Example: What do you think are good candidates of microservices for the Netflix signup system?

Signing up at Netflix



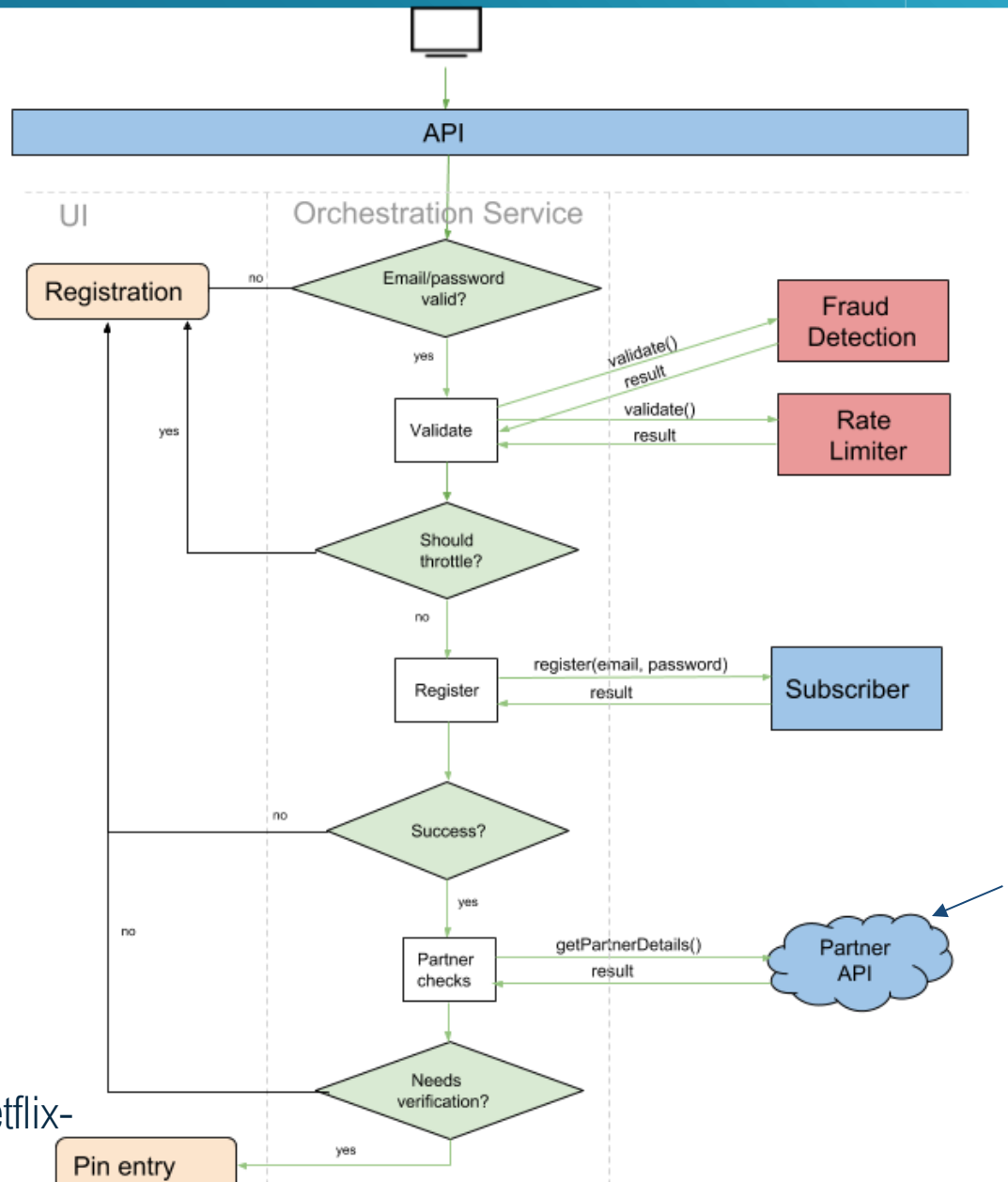
Source: https://netflixtechblog.com/growth-engineering-at-netflix-accelerating-innovation-90eb8e70ce59

Sign up – request registration page



Source: https://netflixtechblog.com/growth-engineering-at-netflix-accelerating-innovation-90eb8e70ce59

Sign up

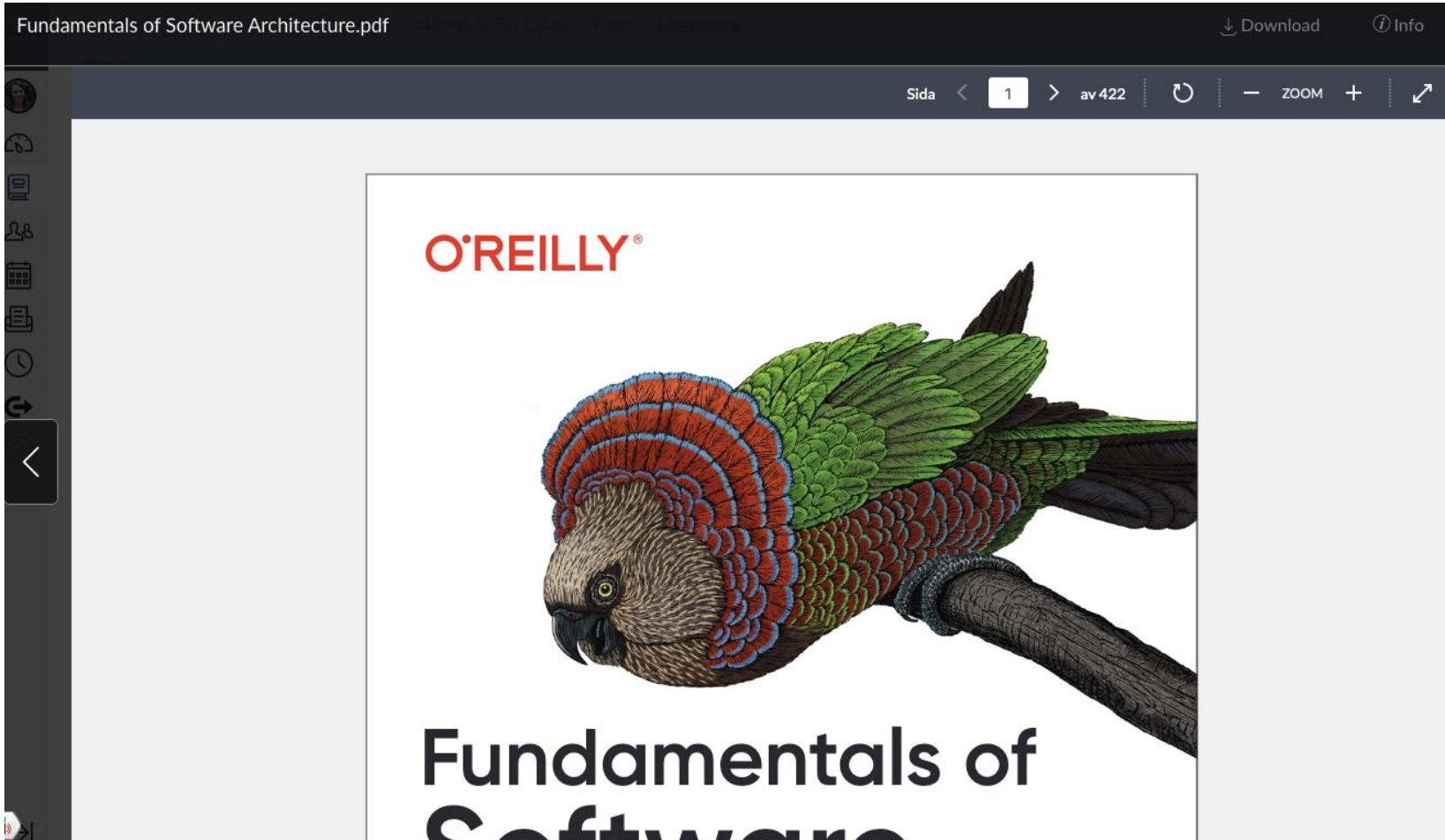


Can be iOS, smart TVs, ...

Source: https://netflixtechblog.com/growth-engineering-at-netflix-accelerating-innovation-90eb8e70ce59

Book Advice: Fundamentals of Software Architecture

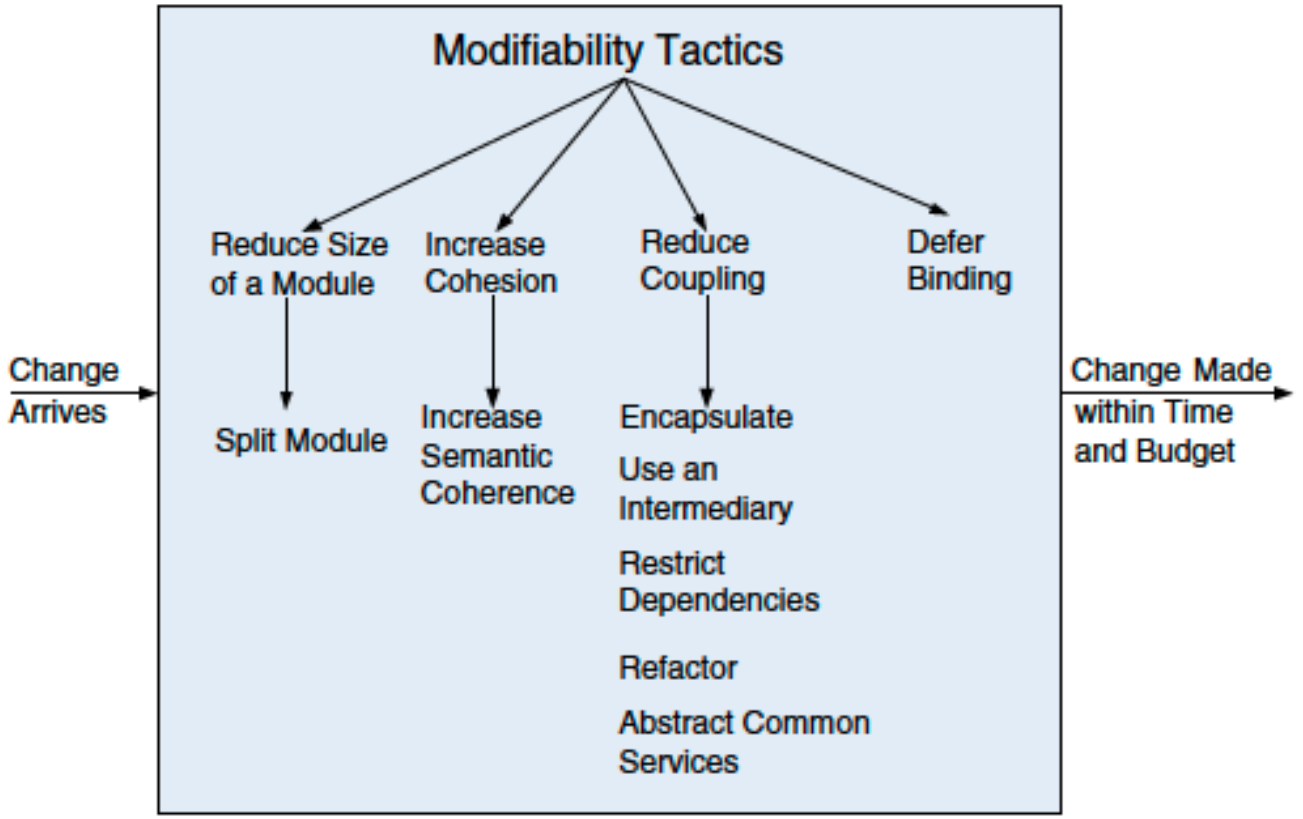
- Describes most of the styles we looked at, together with their pros and cons



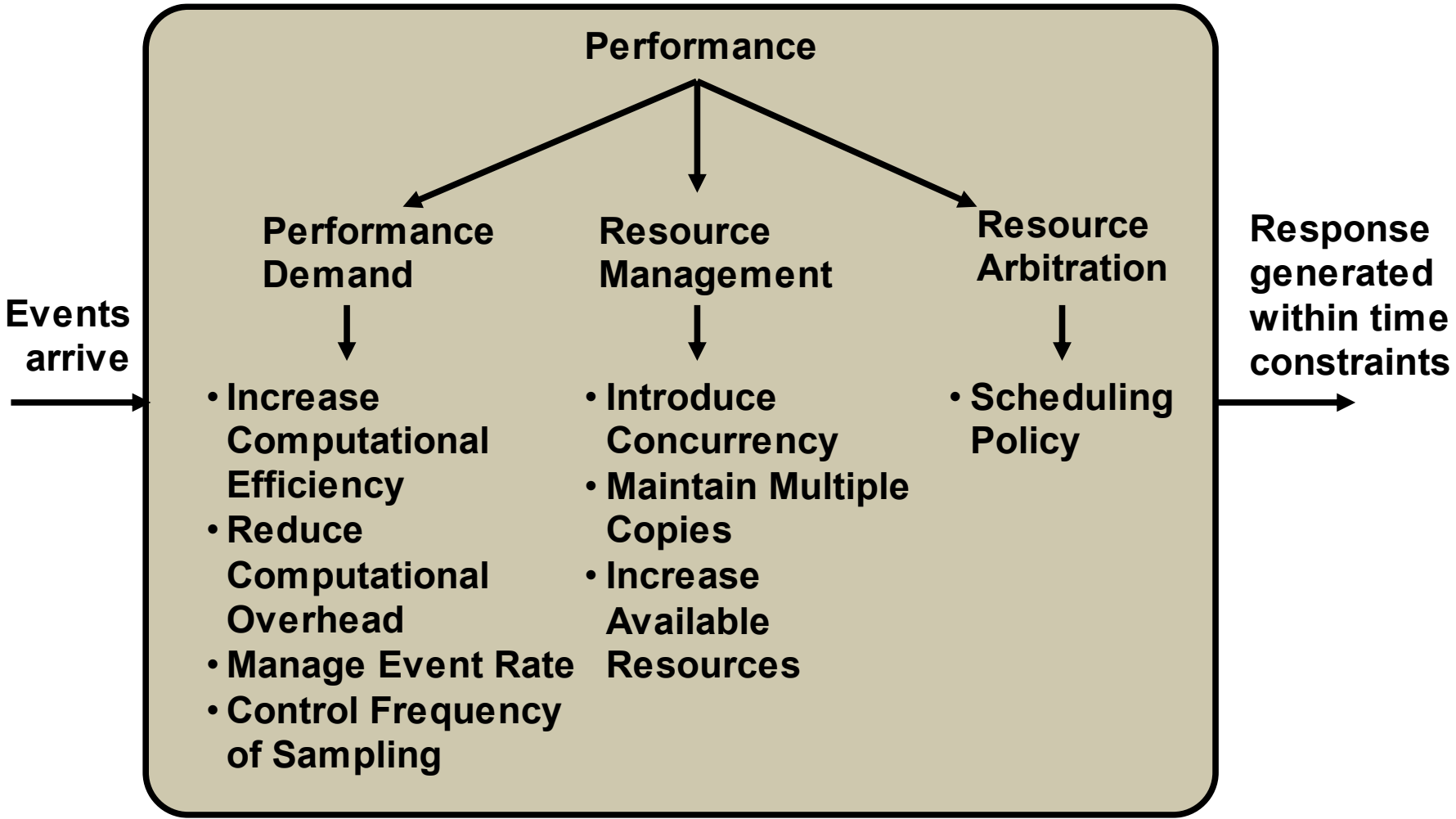
So... What if I have a problem with one (or more) of the QAs?

- Each quality attributes comes with a set of **tactics**.
- An architectural tactic is a design decision that affects a quality attribute response.

- Usually useful to improve a specific quality attribute.

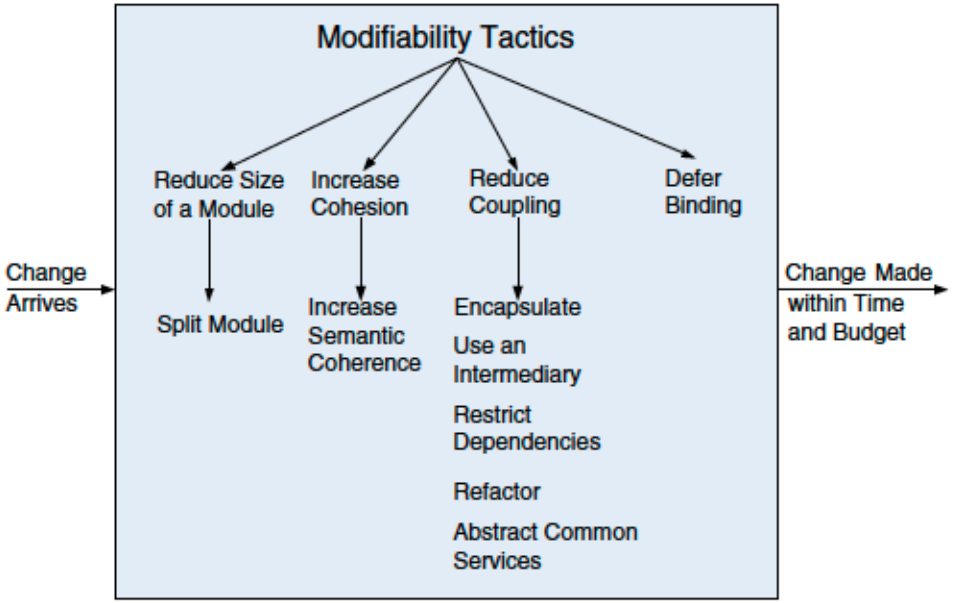


Performance Tactics

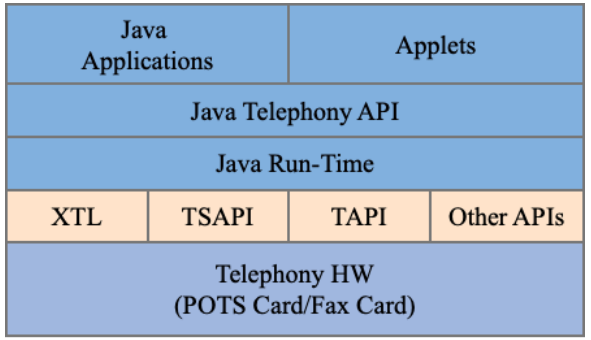


Architectural Tactics and Styles

- You can find a lot of tactics for each quality attribute in the book.
- Tactics are the “building blocks” of design from which architectural styles are created.
- Tactics are atoms and styles (or patterns) are molecules.



- Example: Layered style
- Uses the modifiability tactic ”Restrict dependencies”.
  Layers define an ordering and only allow a layer to use the services of its adjacent lower layer.
- Less dependencies, so it becomes much easier to limit the side effects of replacing a layer.



How can you document architectural decisions?

- Using Architecture Decision Records:
- "An architecture decision record is a short text file [...] that describes a set of forces and a single decision in response to those forces."
- Include: Context, Design Decision, Consequences

Documenting Architecture Decisions by Michael Nygard
http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions

ADR 1: Use the pipe-and-filter library

This ADR is closely related to ADR 2.

In the past while writing agents, we decided it was a good idea to use the pipe and filter architectural pattern. See this ADR for context (if it's still around).

We also found that when the engineers who like pipes and filters took their eyes off of a project, changes would creep in that violated the pattern. Eventually this defeated the purpose of trying to structure the code in an architecturally evident way.

Joe has written a pipe and filter library that will soon be open sourced by the team. Using a library like this enforces the architectural style on the project: there are few ways to code around the framework short of ditching it altogether. This stops engineers from accidentally violating key assumptions on accident.

Decision

We will use the pipe-and-filter library to organize all application logic for the training agents.

Status

Accepted

Consequences

All training agents must fit within the context of pipe and filter. For the moment, we think that this is the case. There may be some pipelines that are very small, but will still work.

All filters must be completely decoupled from each other.

Concurrency can be controlled on a per-filter basis. This means we can write faster programs, but we also must be careful to write thread-safe filters if we want to run them concurrently.

Reacting to errors or expected failure modes will be much easier, and can be implemented with the message subscriber feature of the pipe and filter library.

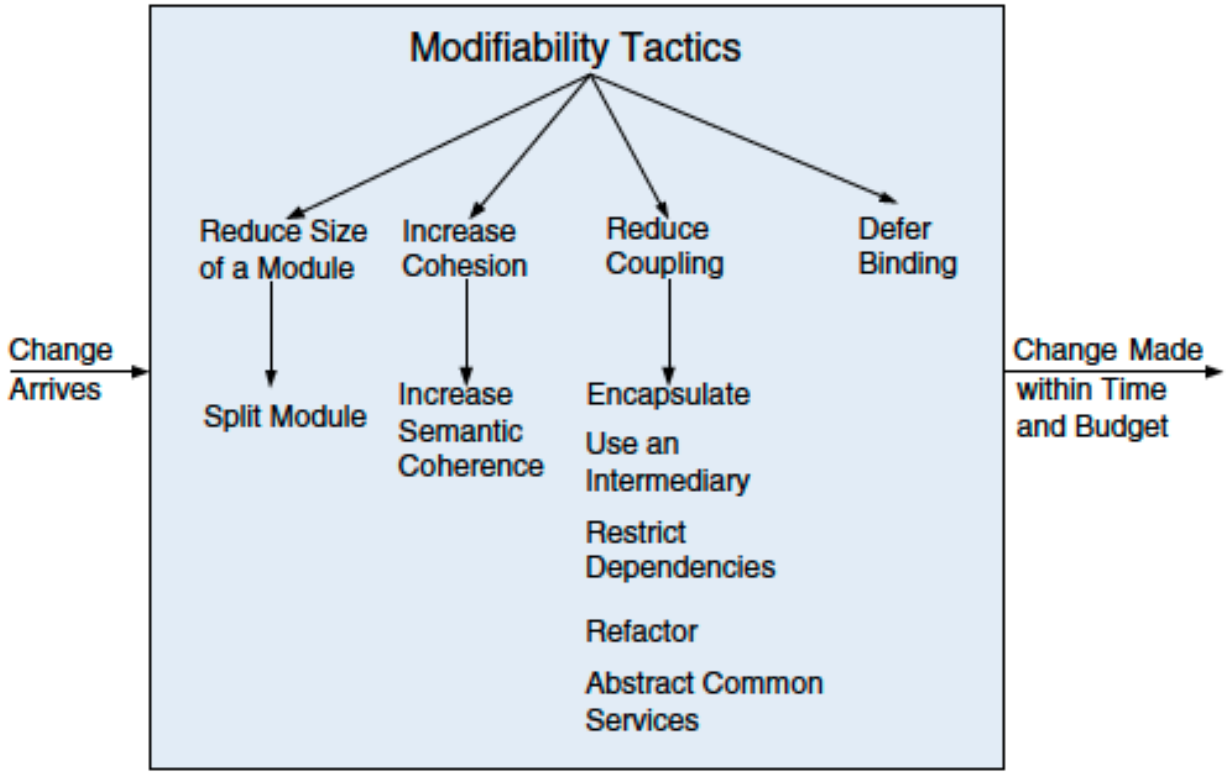
There will be stricter controls on logging and metrics- they must be implemented as message subscribers. This decouples them from the filters entirely, limiting the possibility of mistakes and duplication of logging/metrics code for common error cases, but also means that more work is required to carefully pass good error responses around so that well-formed messages can be constructed and enqueued by the filters.

Uncaught race conditions or errors in the pipeline library may cause failures in production.

So... What if I have a problem with one of the QAs?

- Each quality attributes comes with a set of tactics.
- An architectural tactic is a design decision that affects a quality attribute response.

- Usually useful to improve a specific quality attribute.



Tactics

- A tactic is a design decision that refines an architecture carried out in a style and is influential in controlling a quality attribute response.
- Tactics complement and refine styles/patterns that make up the architecture.

A Brief Introduction to Tactics

- We will look at some tactics to promote:
  - availability
  - performance

What is Availability?

- There are many related terms used in practice.
  - Availability
  - Fault-Tolerance
  - Reliability
  - Resilience
  - Robustness

- Availability informally refers to the degree to which a system is in an operable state.
- More precisely, it refers to the ability of a system to function according to its specification.
  - Including its quality attributes

Terminology: Fault vs Error vs Failure

- Fault: a defect in the software or hardware required to execute the software.
  - May or may not result in a failure
- Error: manifestation of a fault that results in incorrect output or behavior.
- Failure: deviation of a system from its specified behavior.

Fault ⟶ Error ⟶ Failure

Faults and Failures

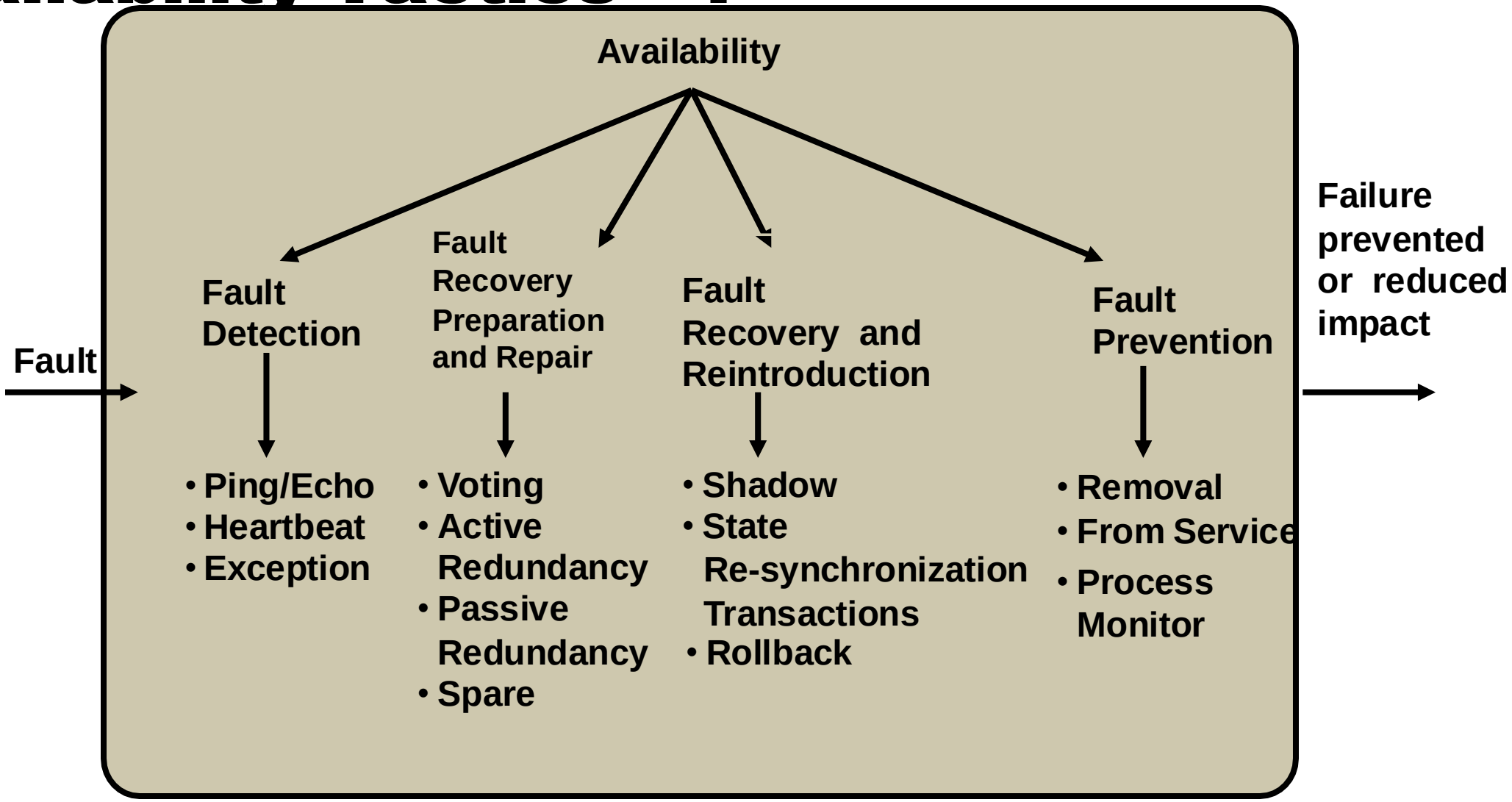
- Many different faults can give rise to the same symptoms or failures.
- Consider:
  - Hung process
  - Processor failure
  - Network outage

- All of these could be experienced in the same way as a reduction in the quality of service (performance, availability, etc.)

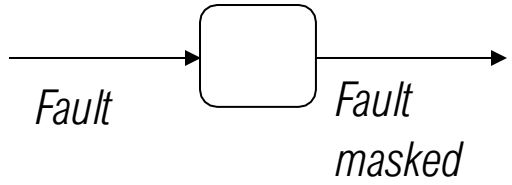
Why Does this Distinction Matter?

- When we talk about the availability of the system, we are referring to failures.
  - For example, we might want the system to be **free from failures** 99.9% of the time.
- When we make design decisions, however, we are addressing how to handle faults.
- We need to translate the failures that we want to avoid into the faults that can cause them.

Availability Tactics - 1



Availability Tactics - 2



Fault detection tactics

ping/echo: one component issues a ping and expects to receive an echo within a predefined time from another component.

heartbeat: one component issues a message periodically while another listens for it.

exception: using exception mechanisms to indicate a fault that occurred at runtime when a component was needed.

What are considerations/tradeoffs in choosing one of these?

Availability Tactics - 3

Fault recovery

voting: when processes take equivalent input and compute output values that are sent to a voter

active redundancy: using redundant elements to respond to events in parallel

passive redundancy: a primary element responds to events and informs an idle element of state updates, so that it can take over the task if needed

spare: a standby computing platform is configured to replace failed components

Availability Tactics - 4

Fault recovery and reintroduction

shadow operation: a previously failed element is run in "shadow mode" before it is returned to service, so that it can be monitored for correctness.

state re-synchronization: saving state periodically and then using it to re-synchronize failed elements.

checkpoint/rollback: recording a consistent state that is created periodically or in response to specific events.

Availability Tactics - 5

Fault prevention

removal from service: removing a system element from operation to undergo some activities to prevent anticipated failures (e.g., resetting).

process monitor: processes are used to monitor critical elements, remove them from service, and re-instantiate new processes in their place.

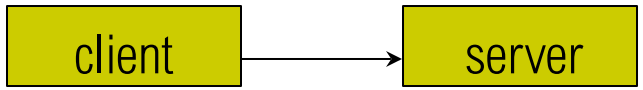
transactions: bundling of several sequential steps such that the entire bundle can be undone at once (like in Git)

prevents data from being affected if one step in a process fails

prevents collisions among several simultaneous threads from accessing the same data.

Working with styles and patterns in practice

Architectural patterns promote some quality attributes and inhibit others.
Once a pattern (or patterns) has (have) been selected,
many decisions still remain.



So what? What are the responsibilities of each element? How can I promote the other QAs that I care about?

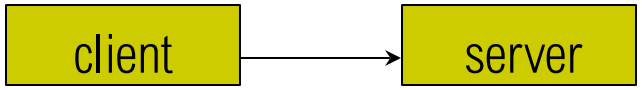
Pattern: *Client/Server*

Properties promoted:
scalability, separation of concerns

Properties inhibited:
performance, security, availability

Working with styles and patterns in practice

Next, we might select tactics to promote the quality attributes that are inhibited by the selection of client-server.



- Example: how might we achieve high availability?
  - tactic #1: Data redundancy
  - tactic #2: Redundant processors
- Similarly we could apply tactics to promote other quality attributes

A Brief Introduction to Tactics

- We will look at some tactics to promote:
  - availability
  - performance

- There are many others, but in the interests of time, we will focus on these.

Performance

- Performance = “It has to be fast.” Right?
- But what do we mean by “fast”? Is that all?
- Answer: “performance” can mean different things depending on the context.

Performance Defined

- Typically people mean one or more of the following:
  - Latency/response time
  - Throughput
  - Predictability (Jitter)
  - Utilization

Latency/Response time

- Latency is the period of delay between some input or action and some desired result.
- This is a concern for many systems.
  - The nature of the concern is different from one system to another, however.
  - For example: Is there value in completing the task after the “deadline” is passed?
  - Sometimes yes, sometimes no

Throughput

- Throughput is concerned with the number of tasks executed per unit time.
- Possible measures for throughput include:
  - Transactions per second
  - Page views per second
  - Streaming bits per second
  - …

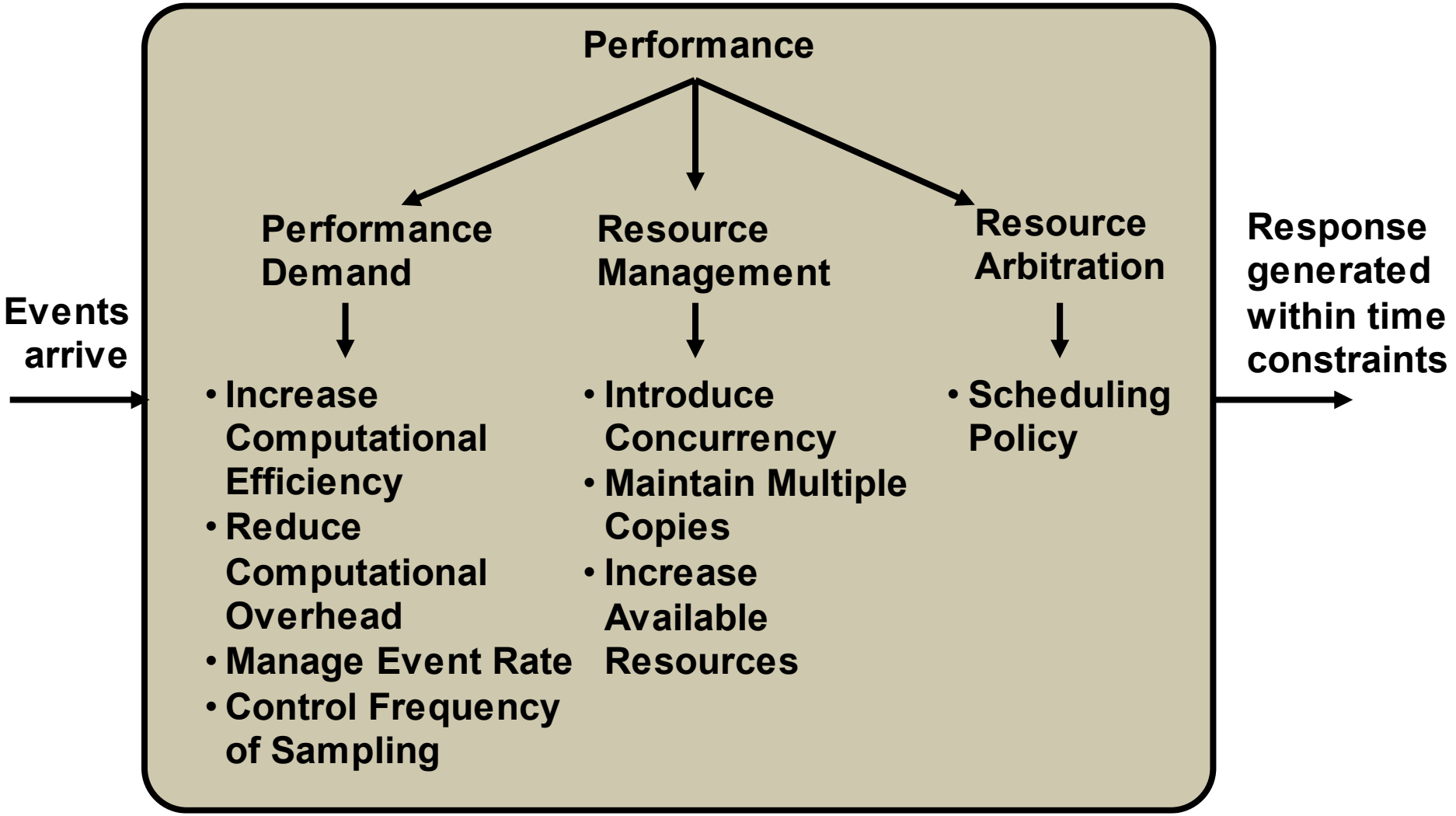
Predictability

- Predictability talks about the variation between executions (when it comes to the response time or execution time).
  - This is sometimes called “jitter”.
- Predictability refers to how deterministic the execution time is.
- This is important when trying to schedule tasks.
- In some domains this is a critical attribute of performance.
  - Example: real-time systems with hard deadlines.

Resource Utilization

- Another important consideration that sometimes comes under the performance heading is “resource utilization”.
- Utilization refers to the extent to which existing resources are used effectively.
- This could have to do with the network, CPU, power consumption, memory, or other resource.

Performance Tactics



Remember

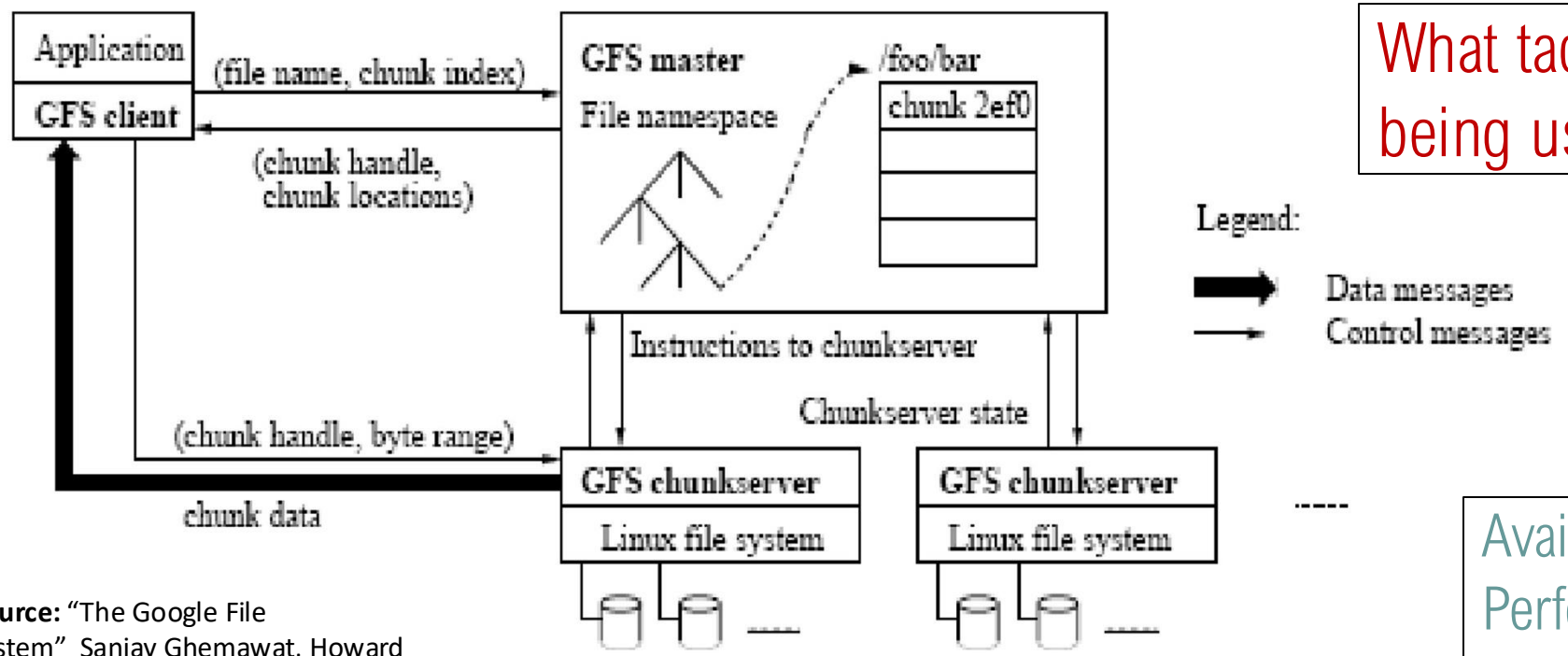


Figure 1: GFS Architecture

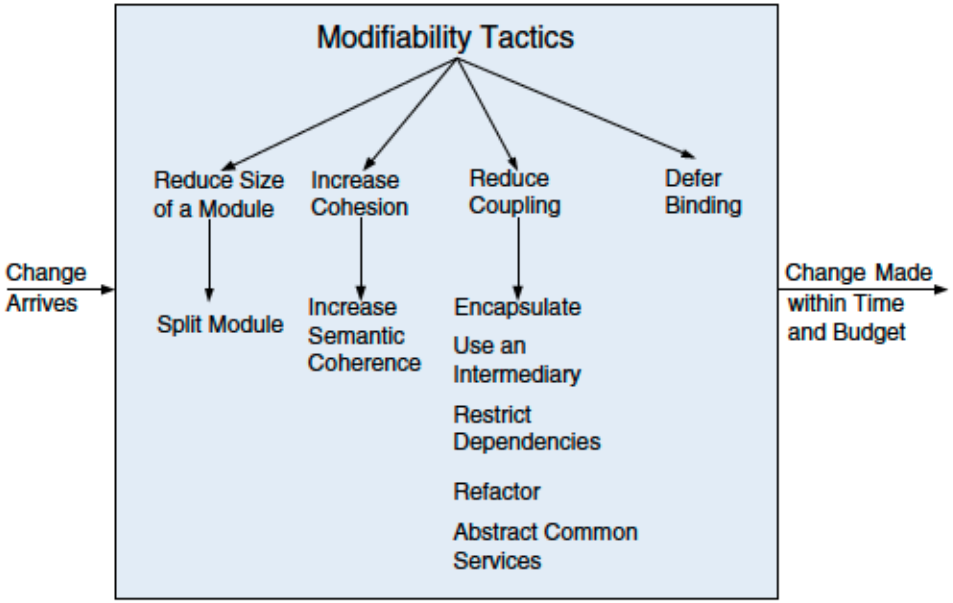
Source: "The Google File System" Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. SOSP 2003.

What tactics are being used here?

Availability
Performance
Scalability Cost

Architectural Tactics and Styles

- You can find a lot of tactics for each quality attribute in the book.
- Tactics are the “building blocks” of design from which architectural patterns are created.
- Tactics are atoms and patterns are molecules.



- Example: Layered pattern
- Uses the modifiability tactic ”Restrict dependencies”.

Layers define an ordering and only allow a layer to use the services of its adjacent lower layer. The possible communication paths are reduced to the number of layers minus one. This limitation has a great influence on the dependencies between the layers and makes it much easier to limit the side effects of replacing a layer.