RESPONSIBILITY ALLOCATION

DECOMPOSING SYSTEMS

- Where do we put the data?
- Where do we put the features?
- What should the interfaces look like?
- How do we weave everything back together?

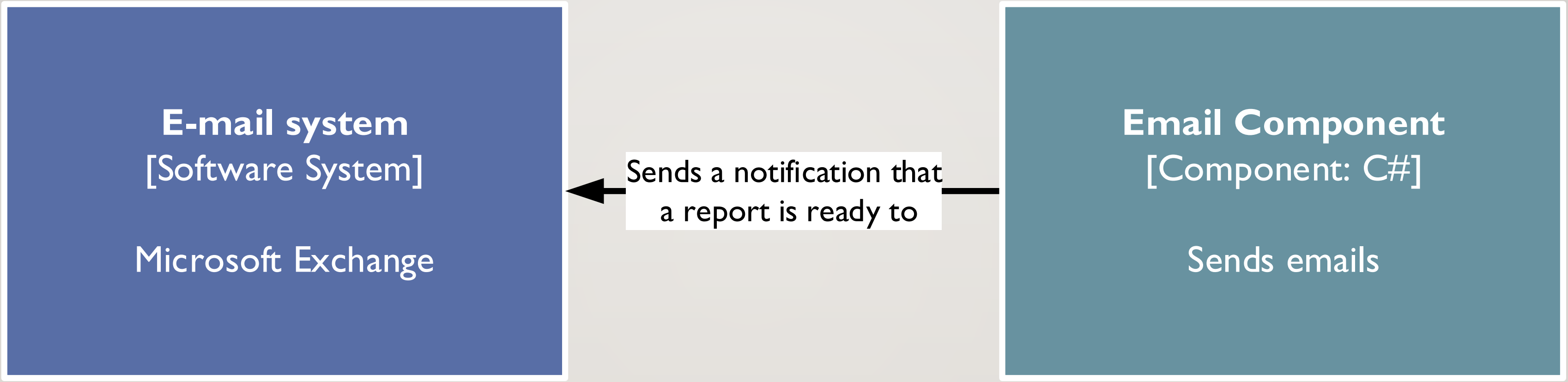
COUPLING & COHESION

COUPLING

Type of Coupling	Effect
Runtime / operational	Consumer cannot run without the provider
Development	Code changes in producer and consumer must be coordinated
Responsibility	Two things change together because of shared responsibility or concepts

Any or all can be present at the same time

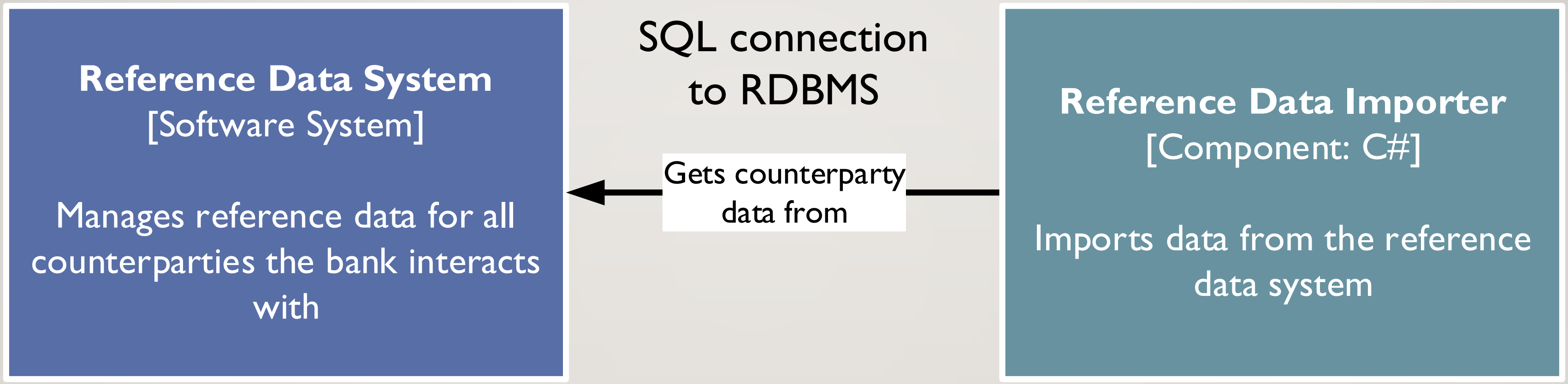
EXAMPLE OF ANALYZING COUPLING



Operational: Strong. SMTP is synchronous, connection-oriented, conversational

Development: Weak. SMTP is well-defined standard with history of interoperability

EXAMPLE OF ANALYZING COUPLING



Reference Data System
[Software System]

Manages reference data for all counterparties the bank interacts with

SQL connection to RDBMS

Gets counterparty data from

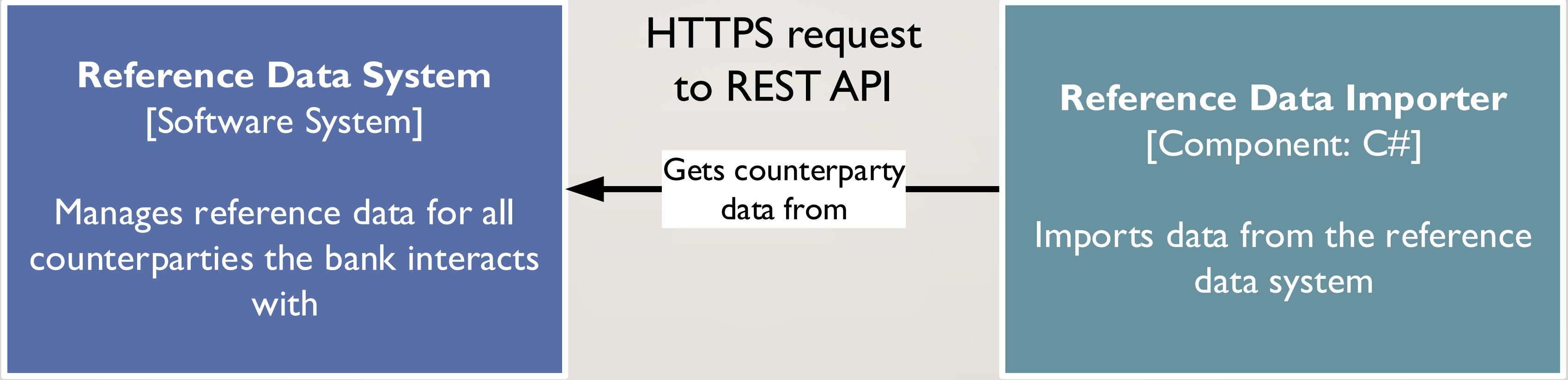
Reference Data Importer
[Component: C#]

Imports data from the reference data system

Operational: Very strong. Dependent on availability of server. Must be aware of topology and failover strategy

Development: Very strong. Dependent on schema, server version, protocol version.

EXAMPLE OF ANALYZING COUPLING



Reference Data System
[Software System]

Manages reference data for all counterparties the bank interacts with

HTTPS request
to REST API

Gets counterparty data from

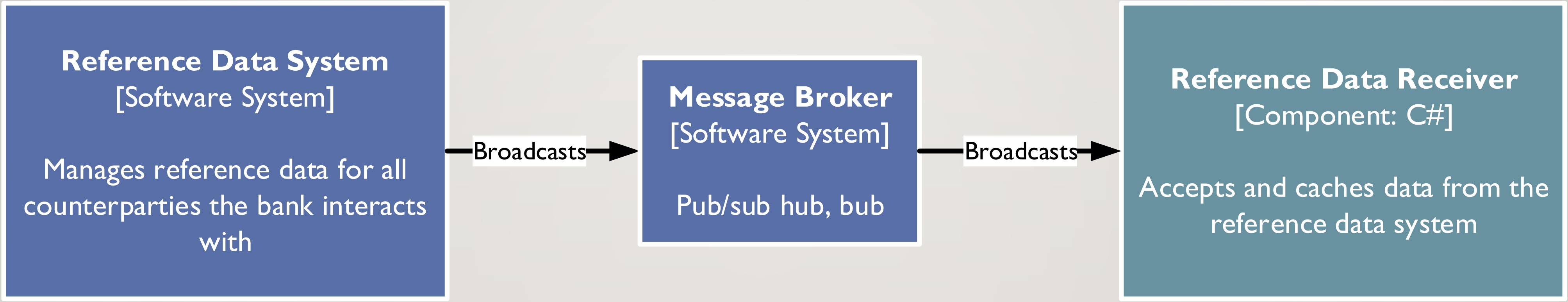
Reference Data Importer
[Component: C#]

Imports data from the reference data system

Operational: Strong, but less than before. Dependent on availability of server.

Development: Strong, but less. Insulated from data format changes. Open encoding can further reduce coupling

EXAMPLE OF ANALYZING COUPLING



Reference Data System
[Software System]

Manages reference data for all counterparties the bank interacts with

Message Broker
[Software System]

Pub/sub hub, bub

Reference Data Receiver
[Component: C#]

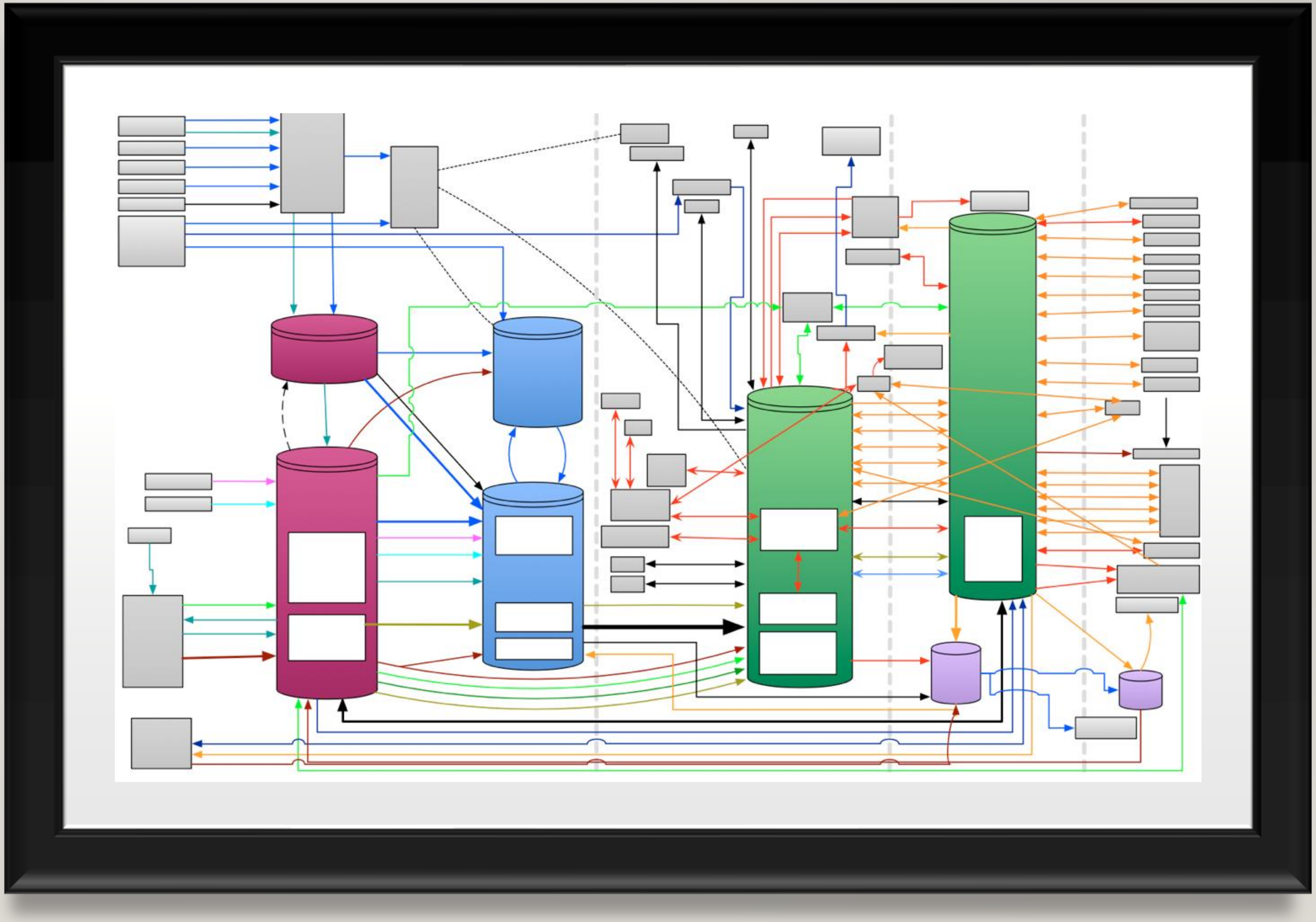
Accepts and caches data from the reference data system

Operational: Very weak. Receiver can run with stale data when either broker or upstream are broken.

Development: Weak. Insulated from schema changes.

“LONG CHAIN” INTERFACES

A SIMPLE ARROW CAN HIDE A GREAT DEAL



EACH "INTERFACE" WAS REALLY A CHAIN

1. Extract tables to files
2. Push files across network
3. Load tables into "LZ"
4. Process into "cold" DB
5. Swap hot & cold DBs (hours later)

1. Send message to queue
2. Take message from queue, unwrap, inspect, and dispatch to 1-of-N other queues.
3. Drain queue to file
4. Batch job wakes up 2 times a day, does FTP to remote end
5. Another batch job pulls a reconciliation file, drops file into file system
6. Parser reads the file, shreds it into messages, puts them on another queue

ARCHITECTURE QUALITIES IN LONG CHAINS

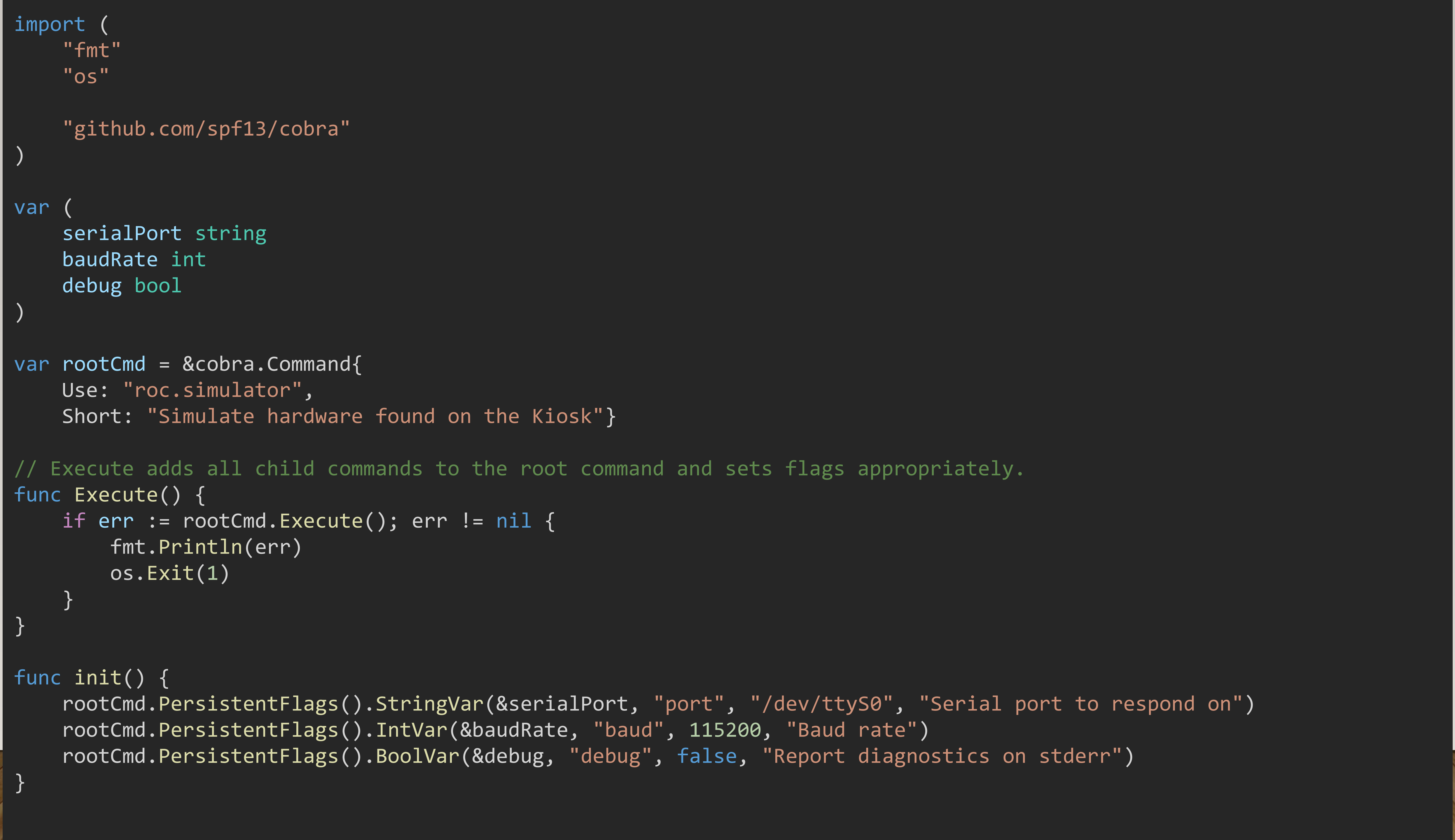
Losses accumulate:

- Latency strictly worse than the slowest link in the chain.
- Availability strictly worse than the least available link.
- Throughput strictly worse than the throughput of the worst bottleneck
- Security strictly worse than the security of the weakest link

COHESION

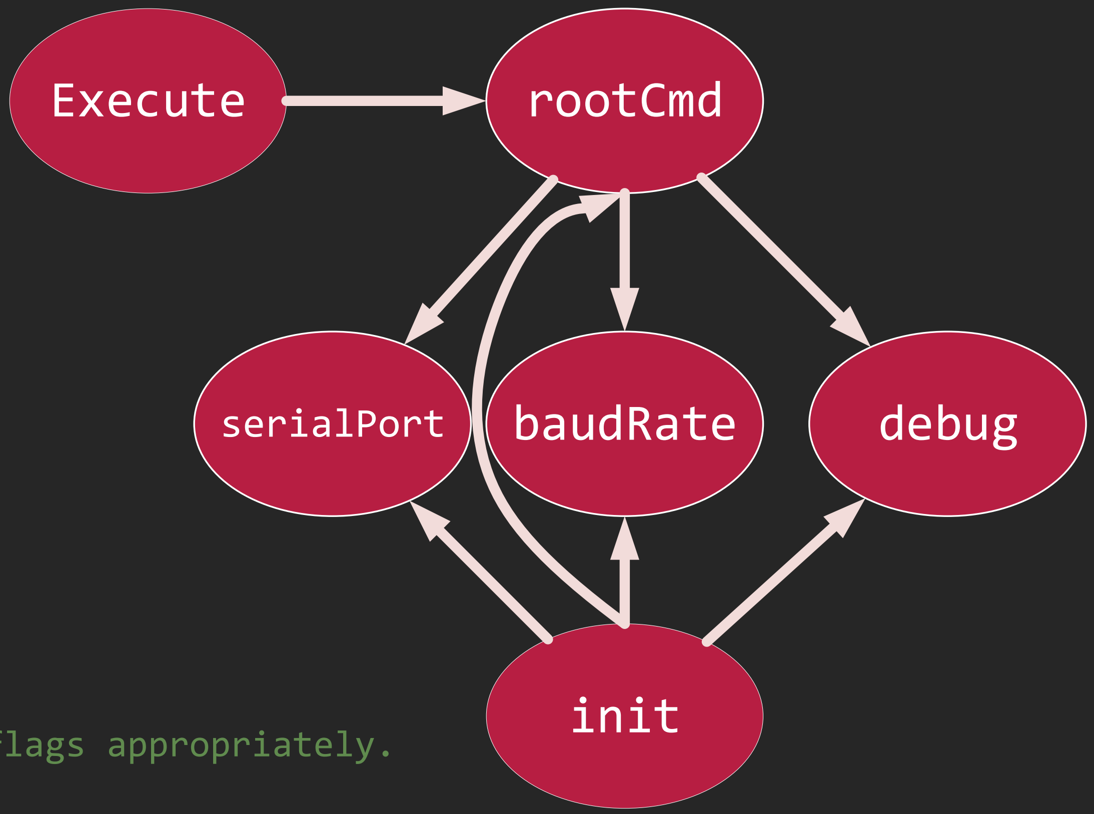
COHESION

- Does the module "fit" together as a logical unit?
- Look at references between functions and variables
- Are they fully connected? Or partitioned?
- Much easier to see in the code than the early designs.
- Iterate and adjust the architecture!

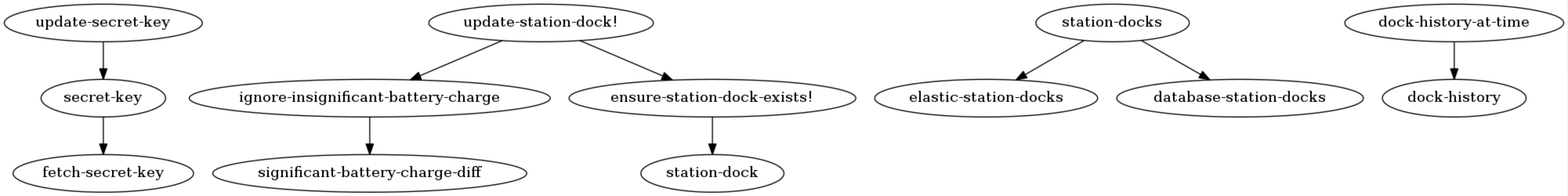


```go
import (
    "fmt"
    "os"

    "github.com/spf13/cobra"
)

var (
    serialPort string
    baudRate int
    debug bool
)

var rootCmd = &cobra.Command{
    Use: "roc.simulator",
    Short: "Simulate hardware found on the Kiosk"}

// Execute adds all child commands to the root command and sets flags appropriately.
func Execute() {
    if err := rootCmd.Execute(); err != nil {
        fmt.Println(err)
        os.Exit(1)
    }
}

func init() {
    rootCmd.PersistentFlags().StringVar(&serialPort, "port", "/dev/ttyS0", "Serial port to respond on")
    rootCmd.PersistentFlags().IntVar(&baudRate, "baud", 115200, "Baud rate")
    rootCmd.PersistentFlags().BoolVar(&debug, "debug", false, "Report diagnostics on stderr")
}
```

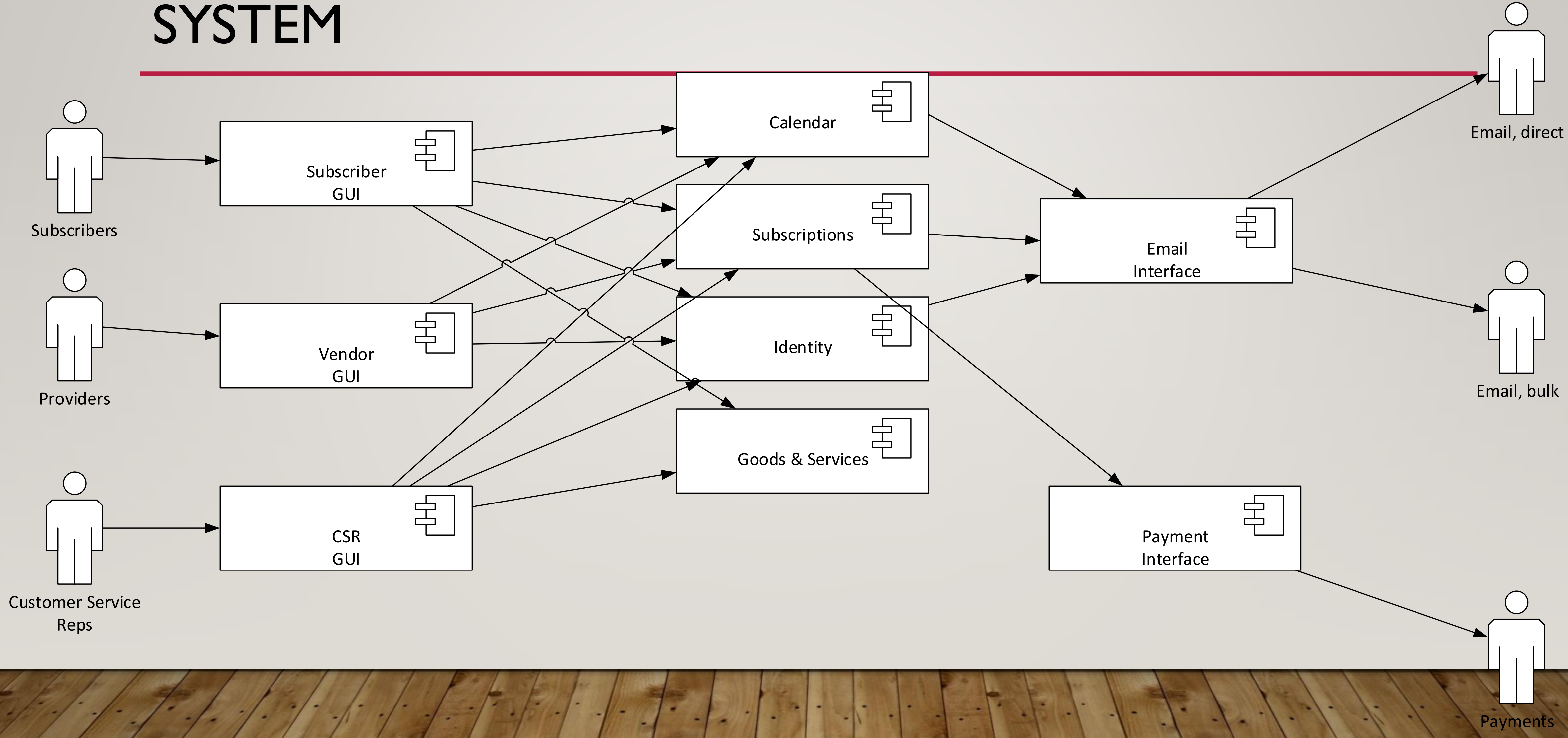
```go
import (
    "fmt"
    "os"

    "github.com/spf13/cobra"
)

var (
    serialPort string
    baudRate int
    debug bool
)

var rootCmd = &cobra.Command{
    Use: "roc.simulator",
    Short: "Simulate hardware found on the Kiosk"}

// Execute adds all child commands to the root command and sets flags appropriately.
func Execute() {
    if err := rootCmd.Execute(); err != nil {
        fmt.Println(err)
        os.Exit(1)
    }
}

func init() {
    rootCmd.PersistentFlags().StringVar(&serialPort, "port", "/dev/ttyS0", "Serial port to respond on")
    rootCmd.PersistentFlags().IntVar(&baudRate, "baud", 115200, "Baud rate")
    rootCmd.PersistentFlags().BoolVar(&debug, "debug", false, "Report diagnostics on stderr")
}
```



NOT VERY COHESIVE



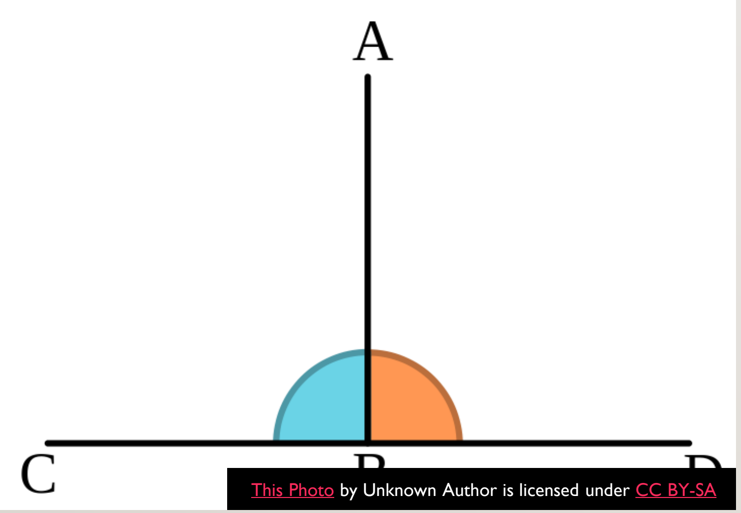
LOOKING BACK: OUR FIRST STAB AT THE SAMPLE SYSTEM



ORTHOGONAL

“You keep using that word…”

ORTHOGONAL: IN MATH



- Dot product of one vector onto the other is zero.
- Zero projection → Perpendicular
- Intersection, but no overlap

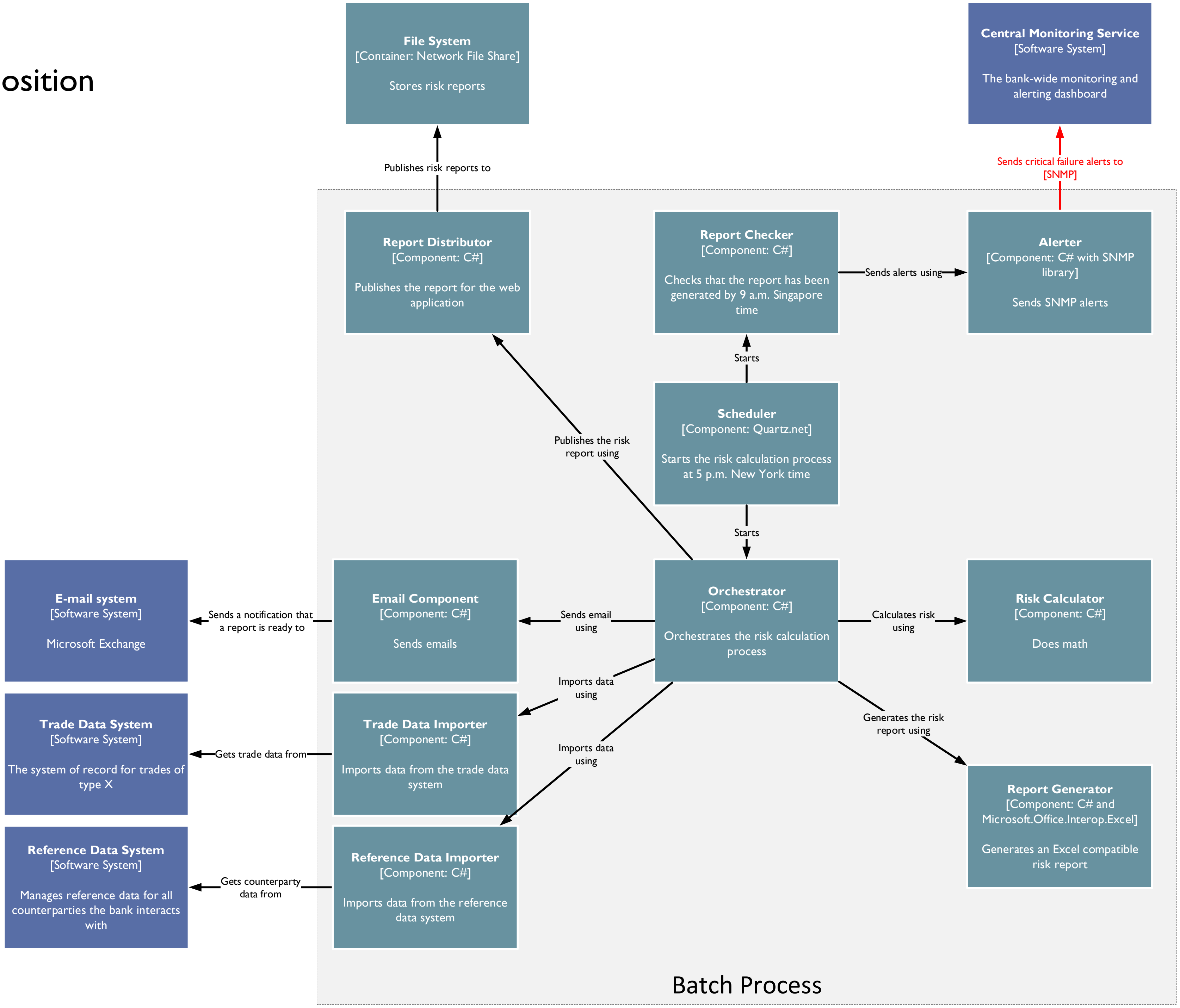
ORTHOGONAL: IN SOFTWARE

- Separation of concerns
- High cohesion within a module or component
- Low coupling between modules or components
- Little overlap in functionality between modules
- Information hiding / decision hiding

Activity: Let's Evaluate This Decomposition

Coupling between modules?

Not bad.


File System
[Container: Network File Share]
Stores risk reports
Central Monitoring Service
[Software System]
The bank-wide monitoring and alerting dashboard
Publishes risk reports to
Sends critical failure alerts to [SNMP]
Report Distributor
[Component: C#]
Publishes the report for the web application
Report Checker
[Component: C#]
Checks that the report has been generated by 9 a.m. Singapore time
Sends alerts using
Alerter
[Component: C# with SNMP library]
Sends SNMP alerts
Starts
Scheduler
[Component: Quartz.net]
Starts the risk calculation process at 5 p.m. New York time
Publishes the risk report using
Starts
E-mail system
[Software System]
Microsoft Exchange
Sends a notification that a report is ready to
Email Component
[Component: C#]
Sends emails
Sends email using
Orchestrator
[Component: C#]
Orchestrates the risk calculation process
Calculates risk using
Risk Calculator
[Component: C#]
Does math
Imports data using
Trade Data System
[Software System]
The system of record for trades of type X
Gets trade data from
Trade Data Importer
[Component: C#]
Imports data from the trade data system
Imports data using
Generates the risk report using
Report Generator
[Component: C# and Microsoft.Office.Interop.Excel]
Generates an Excel compatible risk report
Reference Data System
[Software System]
Manages reference data for all counterparties the bank interacts with
Gets counterparty data from
Reference Data Importer
[Component: C#]
Imports data from the reference data system
Batch Process

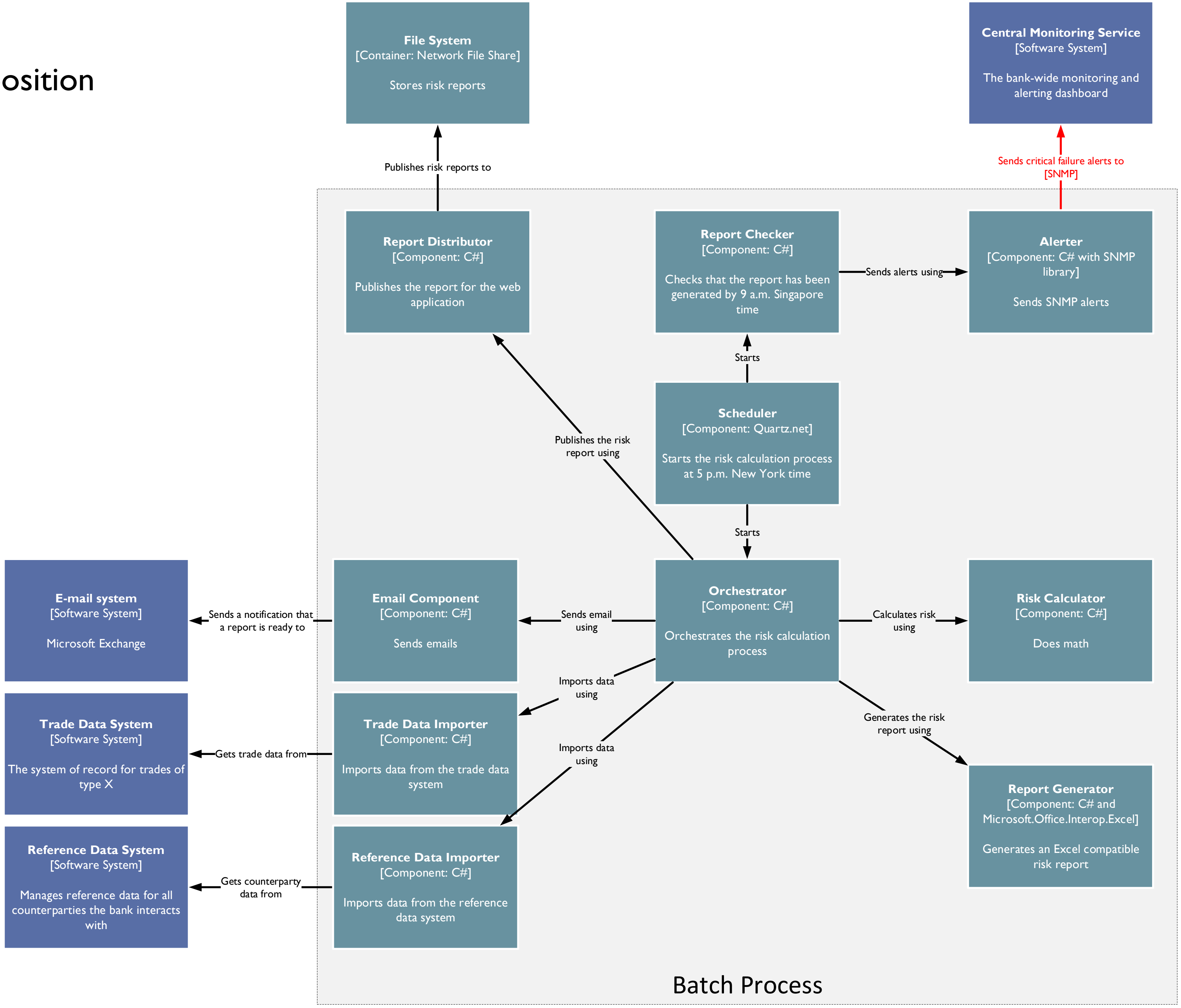
Activity: Let's Evaluate This Decomposition

Coupling between modules?

Not bad.

Cohesion within components?

We can't tell from this level.


File System
[Container: Network File Share]
Stores risk reports
Central Monitoring Service
[Software System]
The bank-wide monitoring and alerting dashboard
Publishes risk reports to
Sends critical failure alerts to [SNMP]
Report Distributor
[Component: C#]
Publishes the report for the web application
Report Checker
[Component: C#]
Checks that the report has been generated by 9 a.m. Singapore time
Sends alerts using
Alerter
[Component: C# with SNMP library]
Sends SNMP alerts
Starts
Scheduler
[Component: Quartz.net]
Starts the risk calculation process at 5 p.m. New York time
Publishes the risk report using
Starts
E-mail system
[Software System]
Microsoft Exchange
Sends a notification that a report is ready to
Email Component
[Component: C#]
Sends emails
Sends email using
Orchestrator
[Component: C#]
Orchestrates the risk calculation process
Calculates risk using
Risk Calculator
[Component: C#]
Does math
Imports data using
Trade Data System
[Software System]
The system of record for trades of type X
Gets trade data from
Trade Data Importer
[Component: C#]
Imports data from the trade data system
Imports data using
Generates the risk report using
Report Generator
[Component: C# and Microsoft.Office.Interop.Excel]
Generates an Excel compatible risk report
Reference Data System
[Software System]
Manages reference data for all counterparties the bank interacts with
Gets counterparty data from
Reference Data Importer
[Component: C#]
Imports data from the reference data system
Batch Process

Activity: Let’s Evaluate This Decomposition

Coupling between modules?

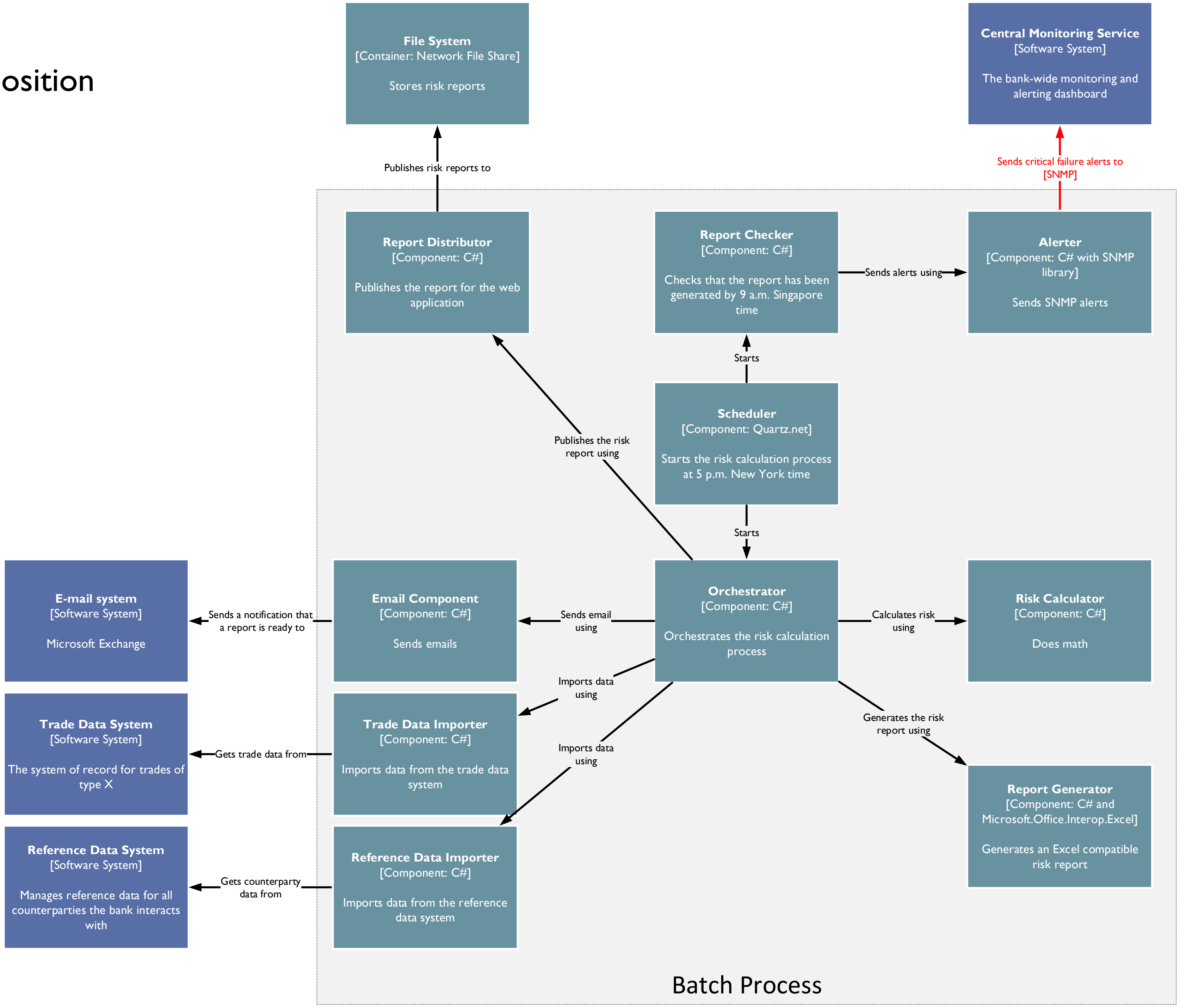
Not bad.

Cohesion within components?

We can’t tell from this level.

Overlapping functionality?

Some, in the importers


File System
[Container: Network File Share]
Stores risk reports
Central Monitoring Service
[Software System]
The bank-wide monitoring and alerting dashboard
Publishes risk reports to
Sends critical failure alerts to [SNMP]
Report Distributor
[Component: C#]
Publishes the report for the web application
Report Checker
[Component: C#]
Checks that the report has been generated by 9 a.m. Singapore time
Sends alerts using
Alerter
[Component: C# with SNMP library]
Sends SNMP alerts
Starts
Scheduler
[Component: Quartz.net]
Starts the risk calculation process at 5 p.m. New York time
Publishes the risk report using
Starts
E-mail system
[Software System]
Microsoft Exchange
Sends a notification that a report is ready to
Email Component
[Component: C#]
Sends emails
Sends email using
Orchestrator
[Component: C#]
Orchestrates the risk calculation process
Calculates risk using
Risk Calculator
[Component: C#]
Does math
Imports data using
Trade Data System
[Software System]
The system of record for trades of type X
Gets trade data from
Trade Data Importer
[Component: C#]
Imports data from the trade data system
Imports data using
Generates the risk report using
Report Generator
[Component: C# and Microsoft.Office.Interop.Excel]
Generates an Excel compatible risk report
Reference Data System
[Software System]
Manages reference data for all counterparties the bank interacts with
Gets counterparty data from
Reference Data Importer
[Component: C#]
Imports data from the reference data system
Batch Process

Activity: Let’s Evaluate This Decomposition

Coupling between modules?

Not bad.

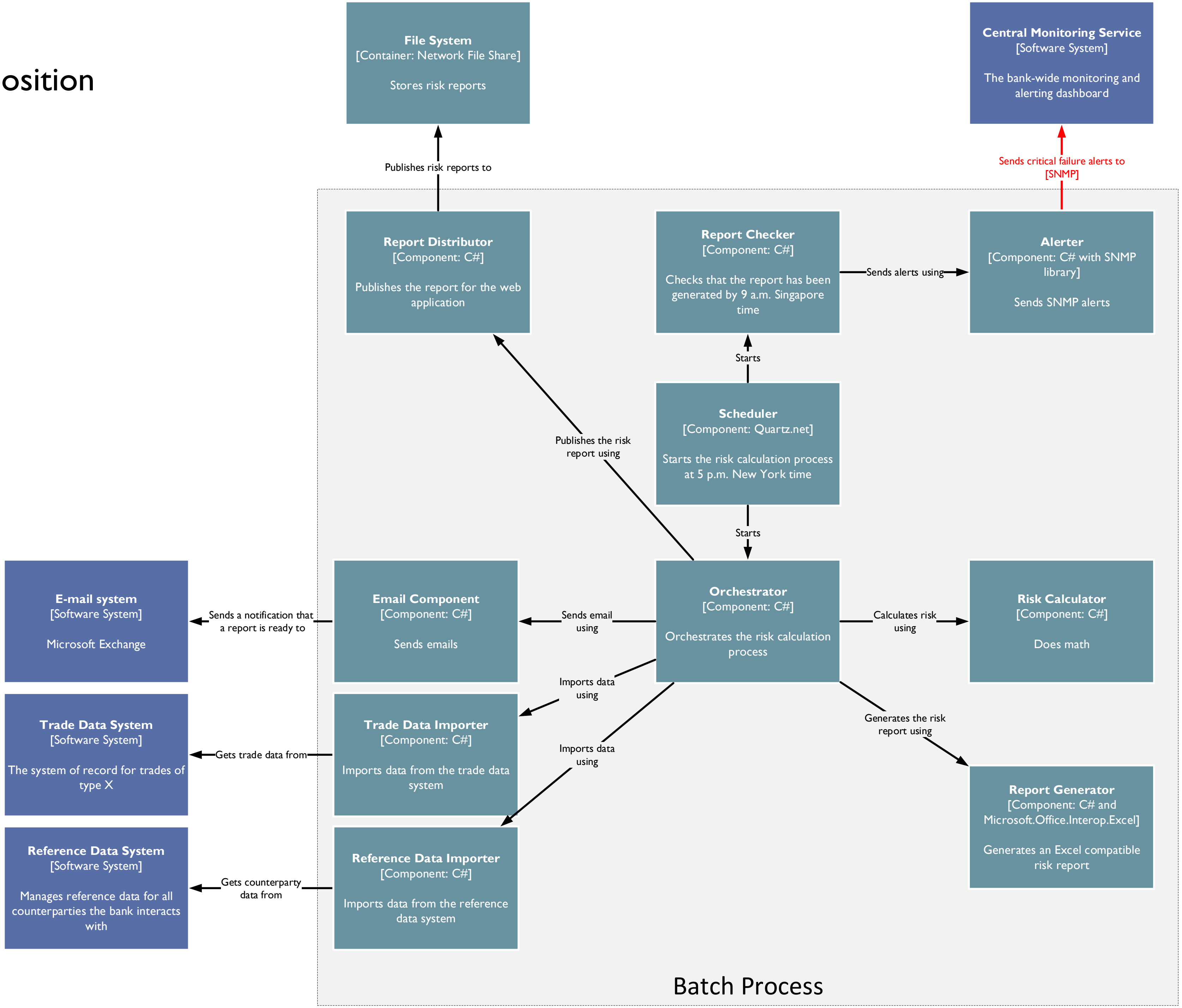
Cohesion within components?

We can’t tell from this level.

Overlapping functionality?

Some, in the importers

As in the Parnas paper, much depends on the API design.



File System
[Container: Network File Share]
Stores risk reports

Central Monitoring Service
[Software System]
The bank-wide monitoring and alerting dashboard

Publishes risk reports to

Sends critical failure alerts to [SNMP]

Report Distributor
[Component: C#]
Publishes the report for the web application

Report Checker
[Component: C#]
Checks that the report has been generated by 9 a.m. Singapore time

Sends alerts using

Alerter
[Component: C# with SNMP library]
Sends SNMP alerts

Starts

Scheduler
[Component: Quartz.net]
Starts the risk calculation process at 5 p.m. New York time

Publishes the risk report using

Starts

E-mail system
[Software System]
Microsoft Exchange

Sends a notification that a report is ready to

Email Component
[Component: C#]
Sends emails

Sends email using

Orchestrator
[Component: C#]
Orchestrates the risk calculation process

Calculates risk using

Risk Calculator
[Component: C#]
Does math

Imports data using

Trade Data System
[Software System]
The system of record for trades of type X

Gets trade data from

Trade Data Importer
[Component: C#]
Imports data from the trade data system

Imports data using

Generates the risk report using

Report Generator
[Component: C# and Microsoft.Office.Interop.Excel]
Generates an Excel compatible risk report

Reference Data System
[Software System]
Manages reference data for all counterparties the bank interacts with

Gets counterparty data from

Reference Data Importer
[Component: C#]
Imports data from the reference data system

Batch Process

Activity: Let’s Evaluate This Decomposition

Coupling between modules?

Not bad.

Cohesion within components?

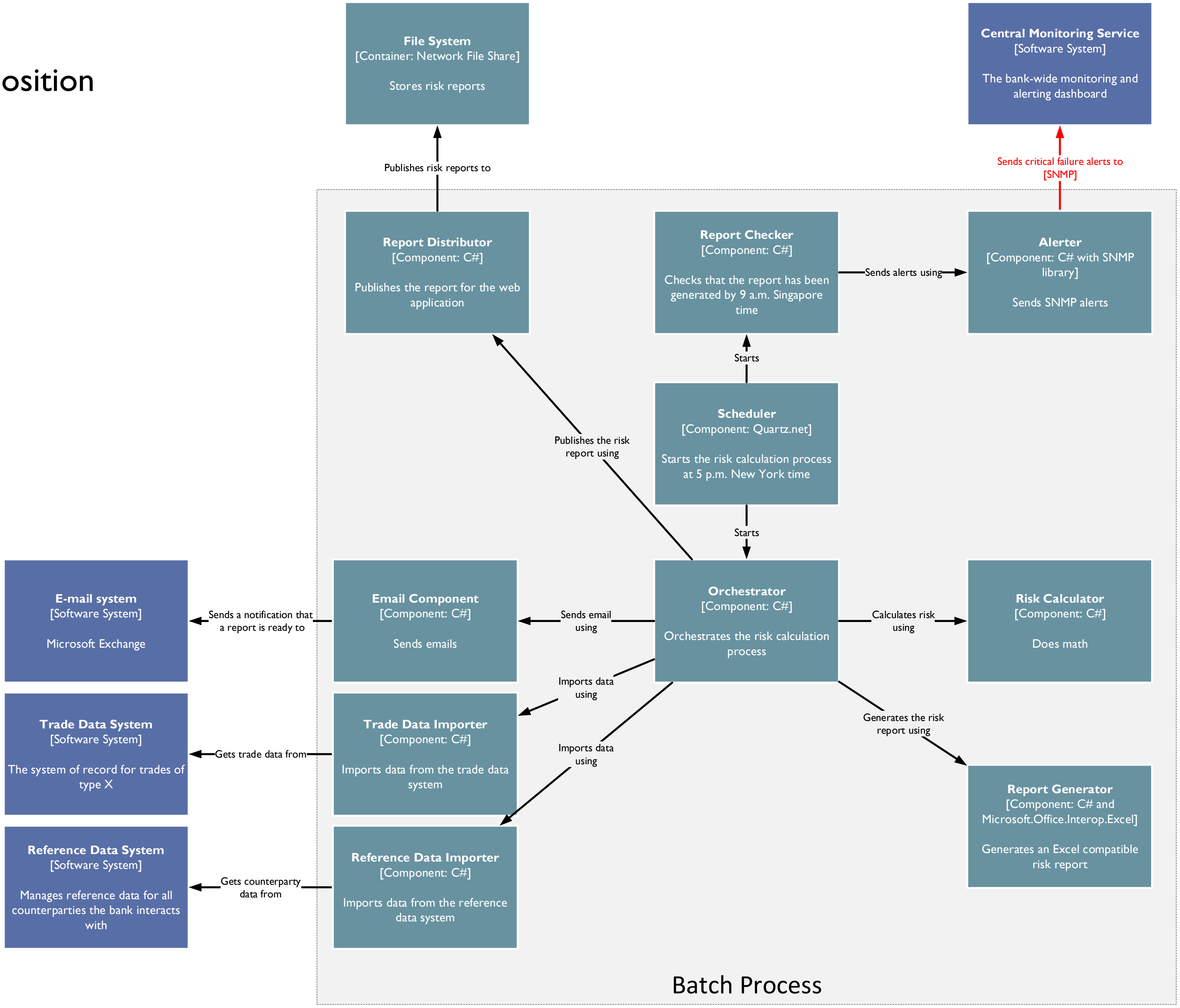
We can’t tell from this level.

Overlapping functionality?

Some, in the importers

As in the Parnas paper, much depends on the API design.

Here are some places that are likely to present trouble



File System
[Container: Network File Share]
Stores risk reports

Central Monitoring Service
[Software System]
The bank-wide monitoring and alerting dashboard

Publishes risk reports to

Sends critical failure alerts to [SNMP]

Report Distributor
[Component: C#]
Publishes the report for the web application

Report Checker
[Component: C#]
Checks that the report has been generated by 9 a.m. Singapore time

Sends alerts using

Alerter
[Component: C# with SNMP library]
Sends SNMP alerts

Starts

Publishes the risk report using

Scheduler
[Component: Quartz.net]
Starts the risk calculation process at 5 p.m. New York time

Starts

E-mail system
[Software System]
Microsoft Exchange

Sends a notification that a report is ready to

Email Component
[Component: C#]
Sends emails

Sends email using

Orchestrator
[Component: C#]
Orchestrates the risk calculation process

Calculates risk using

Risk Calculator
[Component: C#]
Does math

Imports data using

Trade Data System
[Software System]
The system of record for trades of type X

Gets trade data from

Trade Data Importer
[Component: C#]
Imports data from the trade data system

Imports data using

Generates the risk report using

Report Generator
[Component: C# and Microsoft.Office.Interop.Excel]
Generates an Excel compatible risk report

Reference Data System
[Software System]
Manages reference data for all counterparties the bank interacts with

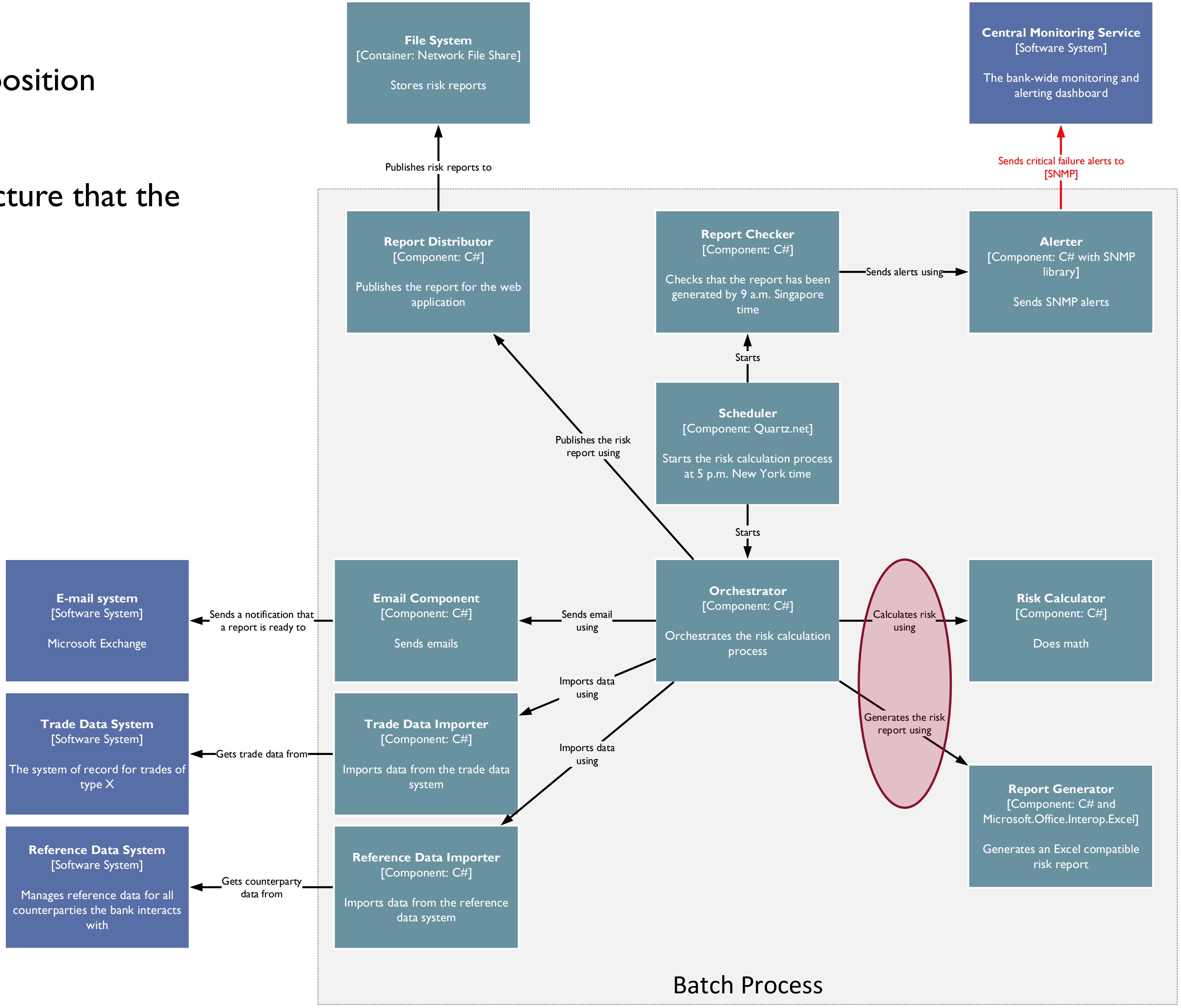
Gets counterparty data from

Reference Data Importer
[Component: C#]
Imports data from the reference data system

Batch Process

Activity: Let's Evaluate This Decomposition

Risk calculator produces a data structure that the report generator must consume.

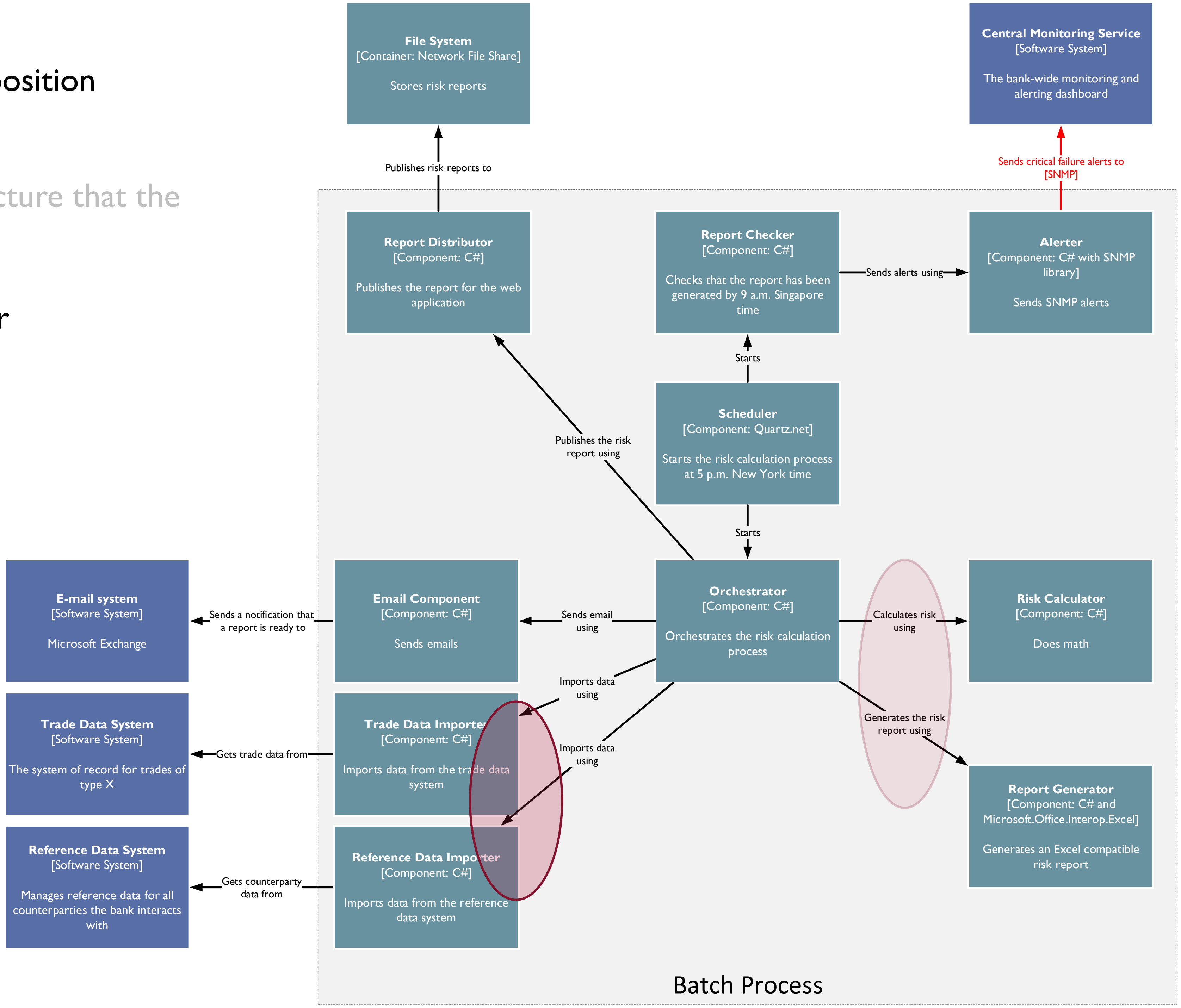


File System
[Container: Network File Share]
Stores risk reports
Central Monitoring Service
[Software System]
The bank-wide monitoring and alerting dashboard
Publishes risk reports to
Sends critical failure alerts to [SNMP]
Report Distributor
[Component: C#]
Publishes the report for the web application
Report Checker
[Component: C#]
Checks that the report has been generated by 9 a.m. Singapore time
Sends alerts using
Alerter
[Component: C# with SNMP library]
Sends SNMP alerts
Starts
Scheduler
[Component: Quartz.net]
Starts the risk calculation process at 5 p.m. New York time
Publishes the risk report using
Starts
E-mail system
[Software System]
Microsoft Exchange
Sends a notification that a report is ready to
Email Component
[Component: C#]
Sends emails
Sends email using
Orchestrator
[Component: C#]
Orchestrates the risk calculation process
Calculates risk using
Risk Calculator
[Component: C#]
Does math
Imports data using
Trade Data System
[Software System]
The system of record for trades of type X
Gets trade data from
Trade Data Importer
[Component: C#]
Imports data from the trade data system
Imports data using
Generates the risk report using
Report Generator
[Component: C# and Microsoft.Office.Interop.Excel]
Generates an Excel compatible risk report
Reference Data System
[Software System]
Manages reference data for all counterparties the bank interacts with
Gets counterparty data from
Reference Data Importer
[Component: C#]
Imports data from the reference data system
Batch Process

Activity: Let's Evaluate This Decomposition

Risk calculator produces a data structure that the report generator must consume.

Data importers probably have similar implementation needs



File System
[Container: Network File Share]

Stores risk reports

Central Monitoring Service
[Software System]

The bank-wide monitoring and alerting dashboard

Publishes risk reports to

Sends critical failure alerts to [SNMP]

Report Distributor
[Component: C#]

Publishes the report for the web application

Report Checker
[Component: C#]

Checks that the report has been generated by 9 a.m. Singapore time

Sends alerts using

Alerter
[Component: C# with SNMP library]

Sends SNMP alerts

Starts

Scheduler
[Component: Quartz.net]

Starts the risk calculation process at 5 p.m. New York time

Publishes the risk report using

Starts

E-mail system
[Software System]

Microsoft Exchange

Sends a notification that a report is ready to

Email Component
[Component: C#]

Sends emails

Sends email using

Orchestrator
[Component: C#]

Orchestrates the risk calculation process

Calculates risk using

Risk Calculator
[Component: C#]

Does math

Imports data using

Imports data using

Generates the risk report using

Trade Data System
[Software System]

The system of record for trades of type X

Gets trade data from

Trade Data Importer
[Component: C#]

Imports data from the trade data system

Report Generator
[Component: C# and Microsoft.Office.Interop.Excel]

Generates an Excel compatible risk report

Reference Data System
[Software System]

Manages reference data for all counterparties the bank interacts with

Gets counterparty data from

Reference Data Importer
[Component: C#]

Imports data from the reference data system

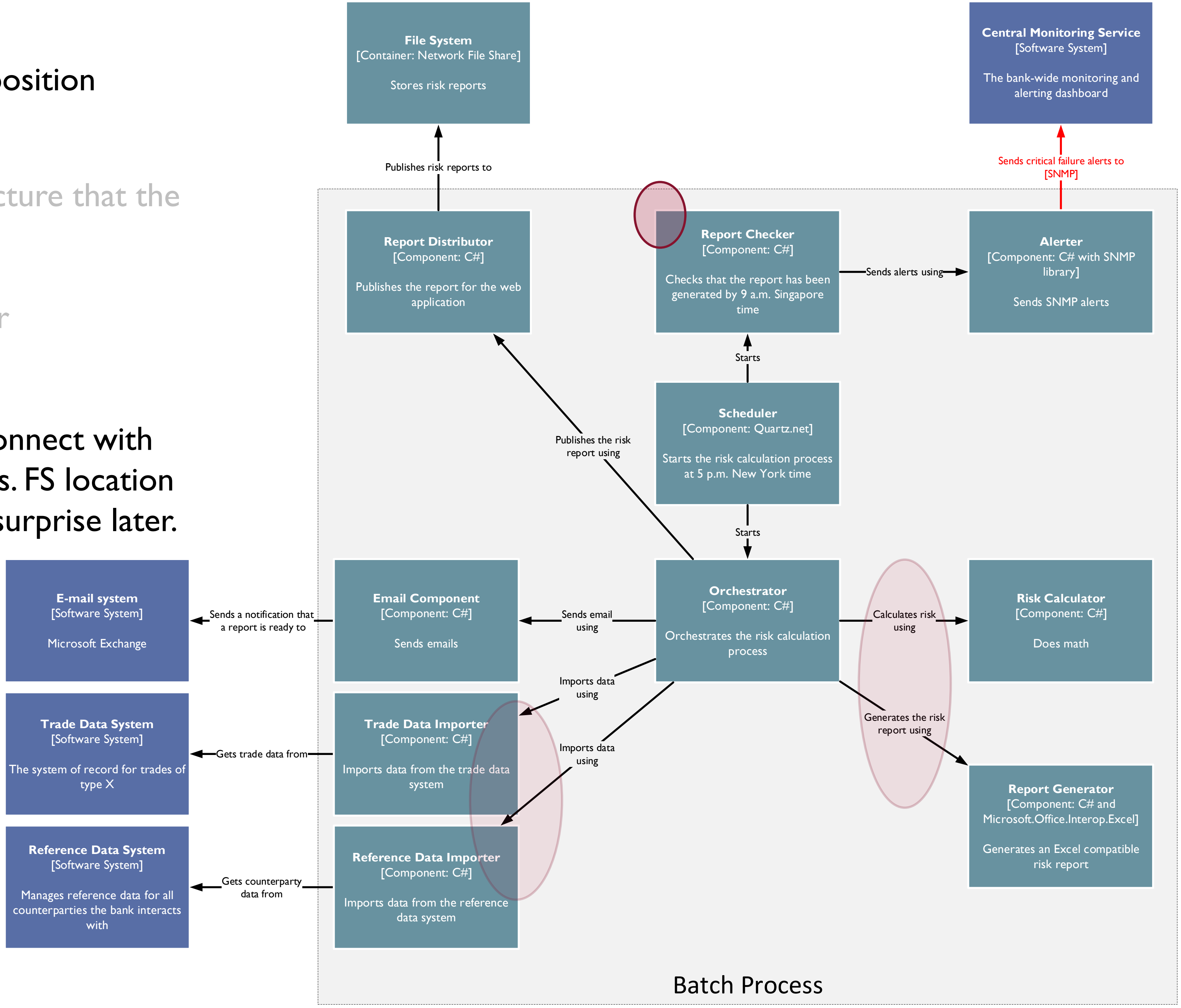
Batch Process

Activity: Let's Evaluate This Decomposition

Risk calculator produces a data structure that the report generator must consume.

Data importers probably have similar implementation needs

Report checker doesn't appear to connect with the file system that holds the reports. FS location is *latent coupling* that will be a nasty surprise later.


File System
[Container: Network File Share]
Stores risk reports
Central Monitoring Service
[Software System]
The bank-wide monitoring and alerting dashboard
Publishes risk reports to
Sends critical failure alerts to [SNMP]
Report Distributor
[Component: C#]
Publishes the report for the web application
Report Checker
[Component: C#]
Checks that the report has been generated by 9 a.m. Singapore time
Sends alerts using
Alerter
[Component: C# with SNMP library]
Sends SNMP alerts
Starts
Scheduler
[Component: Quartz.net]
Starts the risk calculation process at 5 p.m. New York time
Publishes the risk report using
Starts
E-mail system
[Software System]
Microsoft Exchange
Sends a notification that a report is ready to
Email Component
[Component: C#]
Sends emails
Sends email using
Orchestrator
[Component: C#]
Orchestrates the risk calculation process
Calculates risk using
Risk Calculator
[Component: C#]
Does math
Imports data using
Trade Data System
[Software System]
The system of record for trades of type X
Gets trade data from
Trade Data Importer
[Component: C#]
Imports data from the trade data system
Imports data using
Generates the risk report using
Report Generator
[Component: C# and Microsoft.Office.Interop.Excel]
Generates an Excel compatible risk report
Reference Data System
[Software System]
Manages reference data for all counterparties the bank interacts with
Gets counterparty data from
Reference Data Importer
[Component: C#]
Imports data from the reference data system
Batch Process

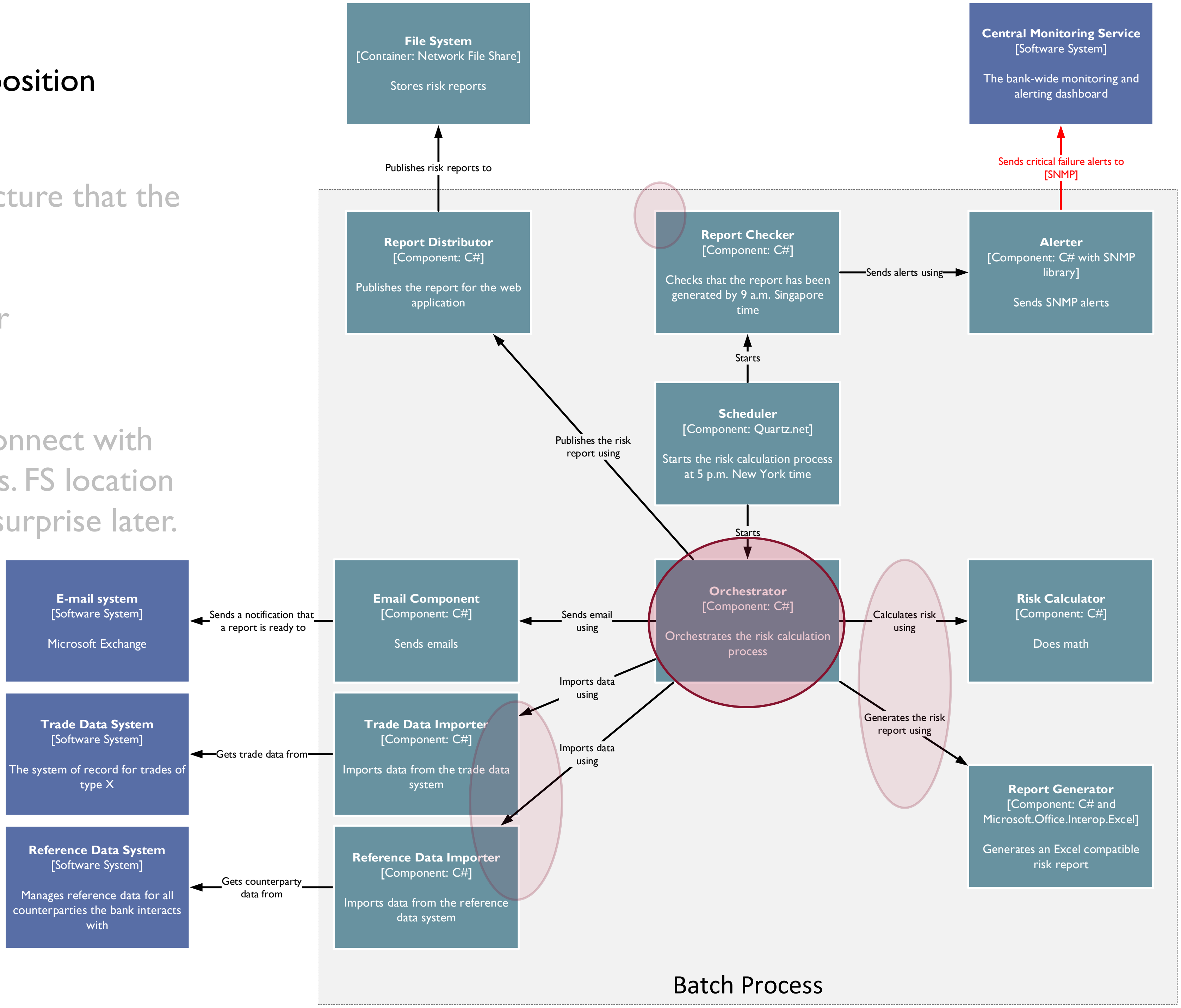
Activity: Let’s Evaluate This Decomposition

Risk calculator produces a data structure that the report generator must consume.

Data importers probably have similar implementation needs

Report checker doesn’t appear to connect with the file system that holds the reports. FS location is *latent coupling* that will be a nasty surprise later.

Orchestrator might end need to do lots of data transformation to bridge interfaces.



File System
[Container: Network File Share]
Stores risk reports
Central Monitoring Service
[Software System]
The bank-wide monitoring and alerting dashboard
Publishes risk reports to
Sends critical failure alerts to [SNMP]
Report Distributor
[Component: C#]
Publishes the report for the web application
Report Checker
[Component: C#]
Checks that the report has been generated by 9 a.m. Singapore time
Sends alerts using
Alerter
[Component: C# with SNMP library]
Sends SNMP alerts
Starts
Scheduler
[Component: Quartz.net]
Starts the risk calculation process at 5 p.m. New York time
Publishes the risk report using
Starts
E-mail system
[Software System]
Microsoft Exchange
Sends a notification that a report is ready to
Email Component
[Component: C#]
Sends emails
Sends email using
Orchestrator
[Component: C#]
Orchestrates the risk calculation process
Calculates risk using
Risk Calculator
[Component: C#]
Does math
Imports data using
Trade Data System
[Software System]
The system of record for trades of type X
Gets trade data from
Trade Data Importer
[Component: C#]
Imports data from the trade data system
Imports data using
Generates the risk report using
Report Generator
[Component: C# and Microsoft.Office.Interop.Excel]
Generates an Excel compatible risk report
Reference Data System
[Software System]
Manages reference data for all counterparties the bank interacts with
Gets counterparty data from
Reference Data Importer
[Component: C#]
Imports data from the reference data system
Batch Process

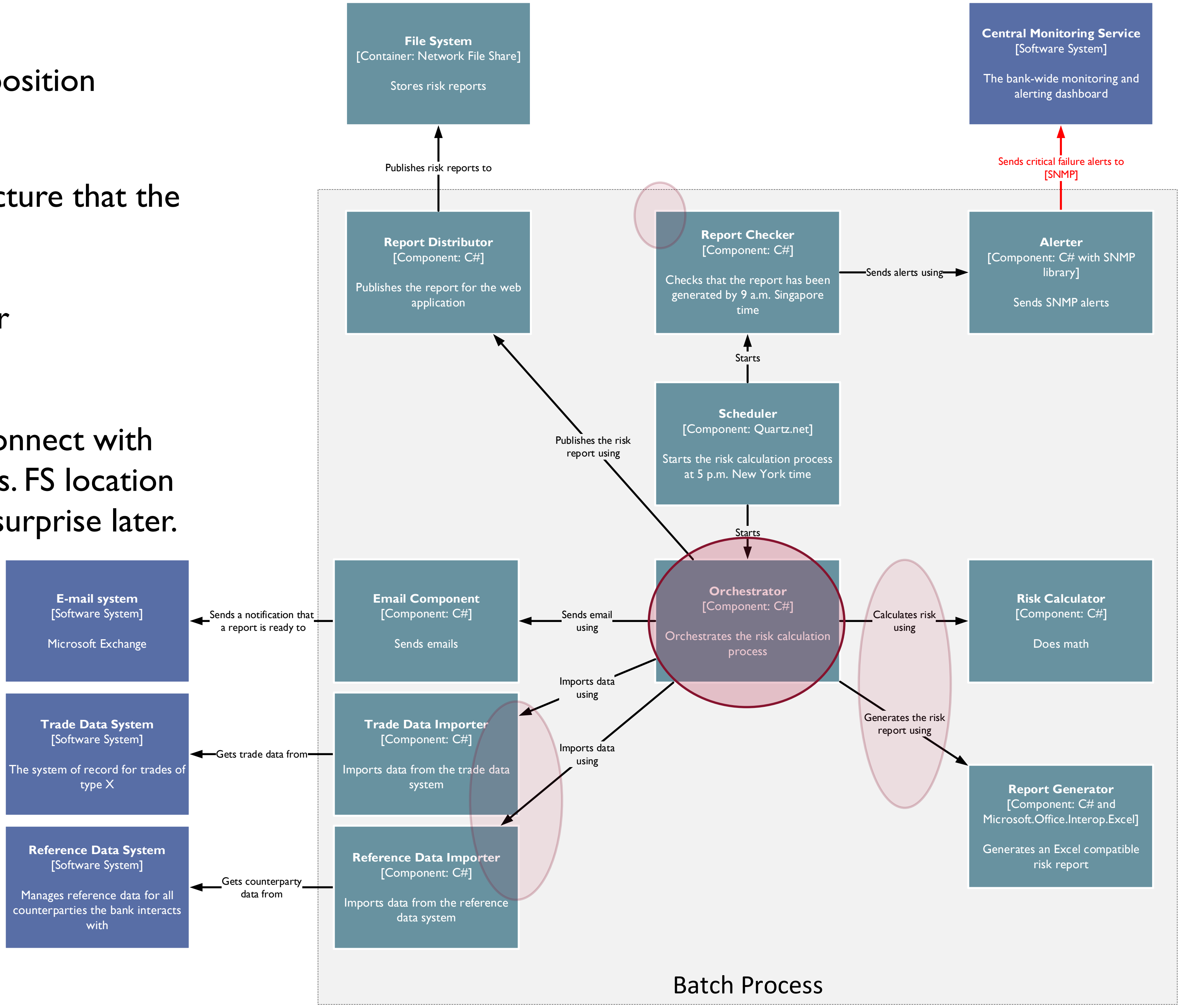
Activity: Let’s Evaluate This Decomposition

Risk calculator produces a data structure that the report generator must consume.

Data importers probably have similar implementation needs

Report checker doesn’t appear to connect with the file system that holds the reports. FS location is *latent coupling* that will be a nasty surprise later.

Orchestrator might end need to do lots of data transformation to bridge interfaces.



File System
[Container: Network File Share]
Stores risk reports

Central Monitoring Service
[Software System]
The bank-wide monitoring and alerting dashboard

Publishes risk reports to

Sends critical failure alerts to
[SNMP]

Report Distributor
[Component: C#]
Publishes the report for the web application

Report Checker
[Component: C#]
Checks that the report has been generated by 9 a.m. Singapore time

Sends alerts using

Alerter
[Component: C# with SNMP library]
Sends SNMP alerts

Starts

Scheduler
[Component: Quartz.net]
Starts the risk calculation process at 5 p.m. New York time

Publishes the risk report using

Starts

E-mail system
[Software System]
Microsoft Exchange

Sends a notification that a report is ready to

Email Component
[Component: C#]
Sends emails

Sends email using

Orchestrator
[Component: C#]
Orchestrates the risk calculation process

Calculates risk using

Risk Calculator
[Component: C#]
Does math

Imports data using

Trade Data System
[Software System]
The system of record for trades of type X

Gets trade data from

Trade Data Importer
[Component: C#]
Imports data from the trade data system

Imports data using

Generates the risk report using

Report Generator
[Component: C# and Microsoft.Office.Interop.Excel]
Generates an Excel compatible risk report

Reference Data System
[Software System]
Manages reference data for all counterparties the bank interacts with

Gets counterparty data from

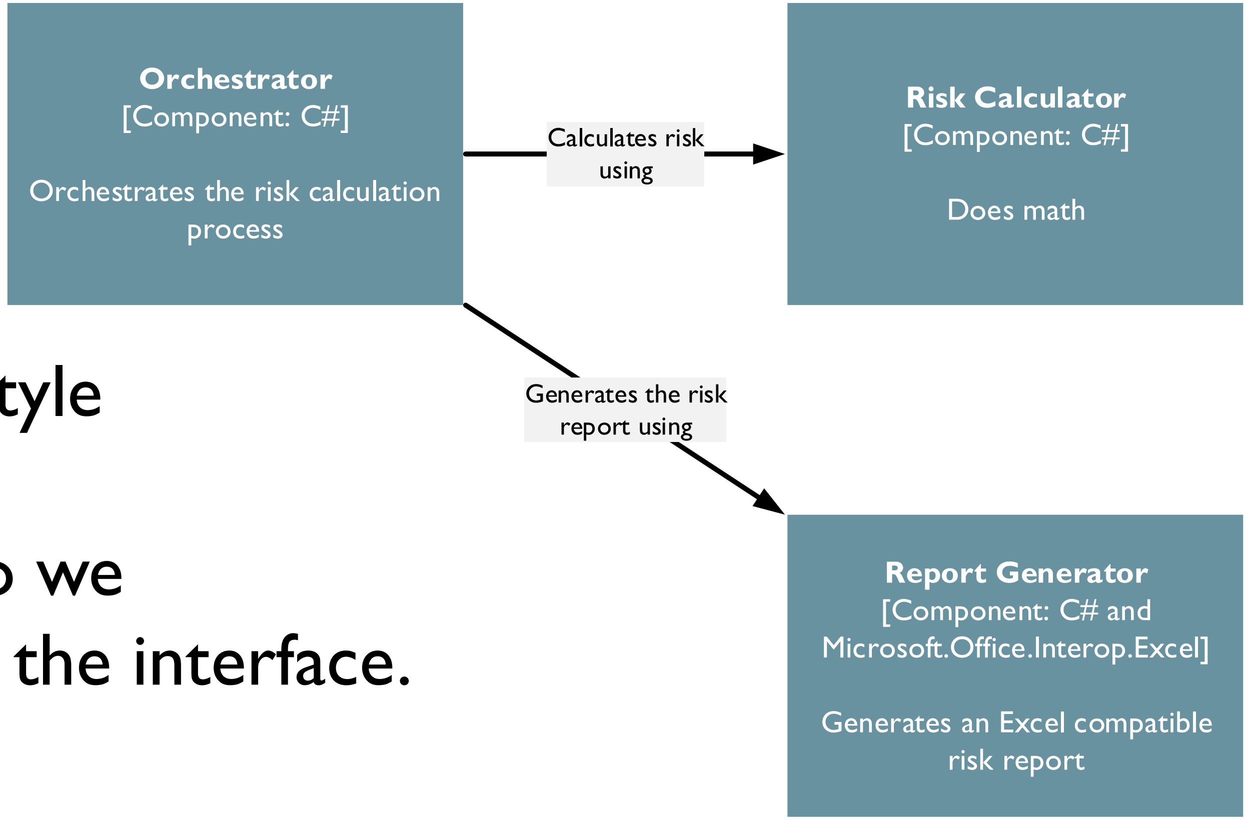
Reference Data Importer
[Component: C#]
Imports data from the reference data system

Batch Process

Problem: Risk calculator produces a data structure that the report generator must consume.

Solutions depend on architectural style

Here we're in a Windows service so we might use a shared library to define the interface.



Orchestrator
[Component: C#]

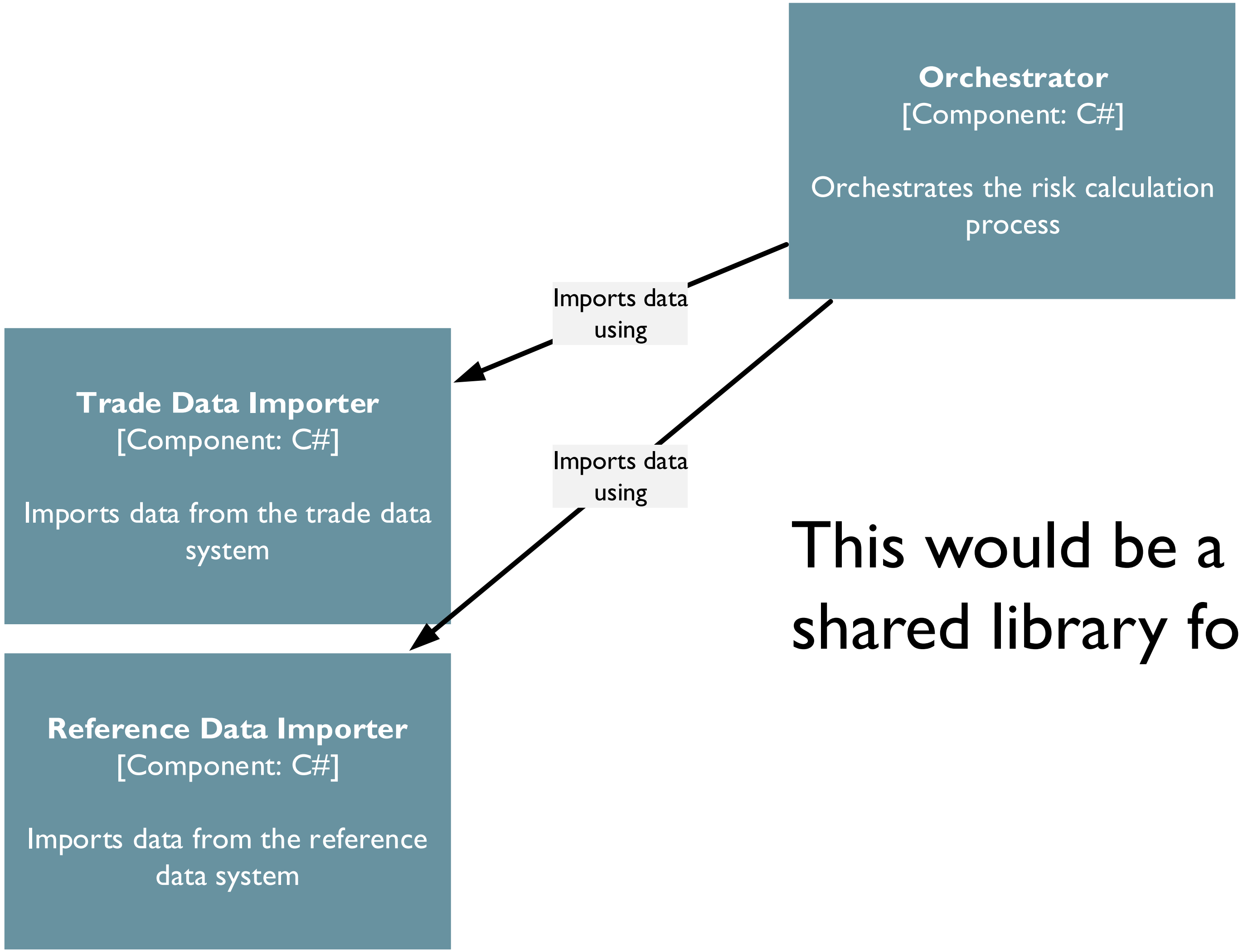
Orchestrates the risk calculation process

Risk Calculator
[Component: C#]

Does math

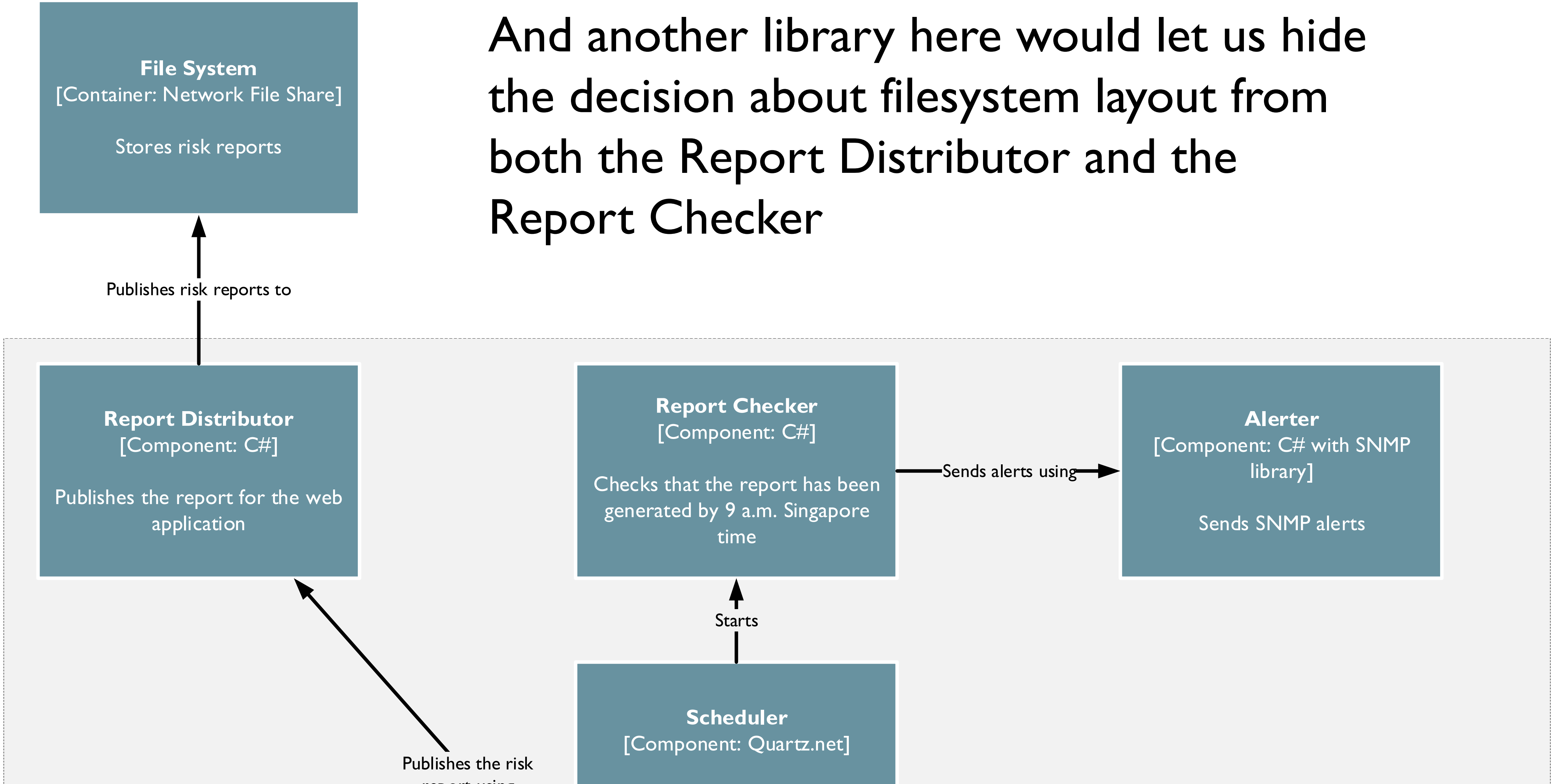
Report Generator
[Component: C# and Microsoft.Office.Interop.Excel]

Generates an Excel compatible risk report


Orchestrator
[Component: C#]
Orchestrates the risk calculation process
Imports data using
Imports data using
Trade Data Importer
[Component: C#]
Imports data from the trade data system
Reference Data Importer
[Component: C#]
Imports data from the reference data system

This would be a good place to use a shared library for common implementation.

And another library here would let us hide the decision about filesystem layout from both the Report Distributor and the Report Checker



File System
[Container: Network File Share]

Stores risk reports

Publishes risk reports to

Report Distributor
[Component: C#]

Publishes the report for the web application

Report Checker
[Component: C#]

Checks that the report has been generated by 9 a.m. Singapore time

Sends alerts using

Alerter
[Component: C# with SNMP library]

Sends SNMP alerts

Starts

Scheduler
[Component: Quartz.net]

Publishes the risk report using

USE ALL YOUR TOOLS

1. Module structure – layout of your code and libraries
2. Component structure – interactions between runtime components
3. Abstraction – Emphasize similar interfaces & data formats

Find solutions by rotating your perspective

When looking at components, think about modules

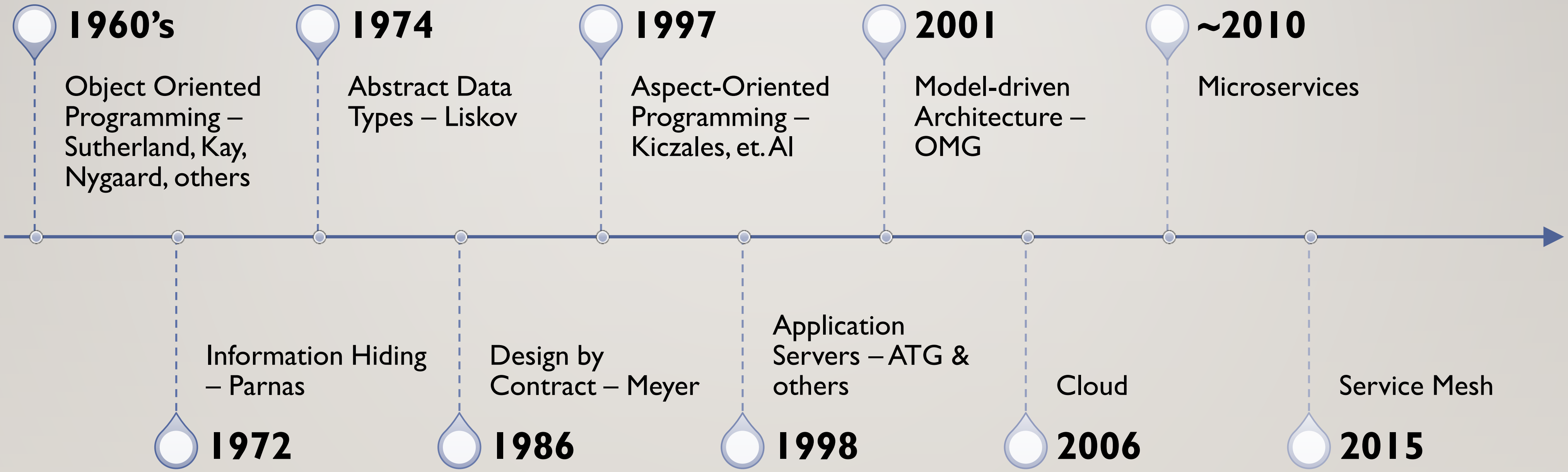
When looking at modules, think about components

When looking at data, think about code

When looking at code, think about data

SEPARATION OF CONCERNS

SoC: A PERENNIAL STRUGGLE



1960's
Object Oriented Programming – Sutherland, Kay, Nygaard, others

1972
Information Hiding – Parnas

1974
Abstract Data Types – Liskov

1986
Design by Contract – Meyer

1997
Aspect-Oriented Programming – Kiczales, et. Al

1998
Application Servers – ATG & others

2001
Model-driven Architecture – OMG

2006
Cloud

~2010
Microservices

2015
Service Mesh

COMMON TO EVERY SYSTEM

- Input/Output channels
- Initialization
- Configuration, credentials
- Configuration, performance
- Storage
- Query
- Consistency
- Encryption, authn, authz
- Deployment
- Failure and recovery

DOMAIN SPECIFIC — SUBSCRIPTIONS

- Bank interface
- Payment handling
- Customer service
- Refunds
- Fraud detection/mitigation

IDEAL SEPARATION

- One mechanism per concern (maybe even less than one per concern!)
- All perfectly orthogonal & composable

PRAGMATICALLY: PICK YOUR BATTLES

- Look at your architectural priorities, constraints, and ASRs.
- Solve for those first

DIMENSIONS TO WORK WITH

- Modules (e.g., Library)
- Components
- Processes
- Hosts
- Services
- Geographies

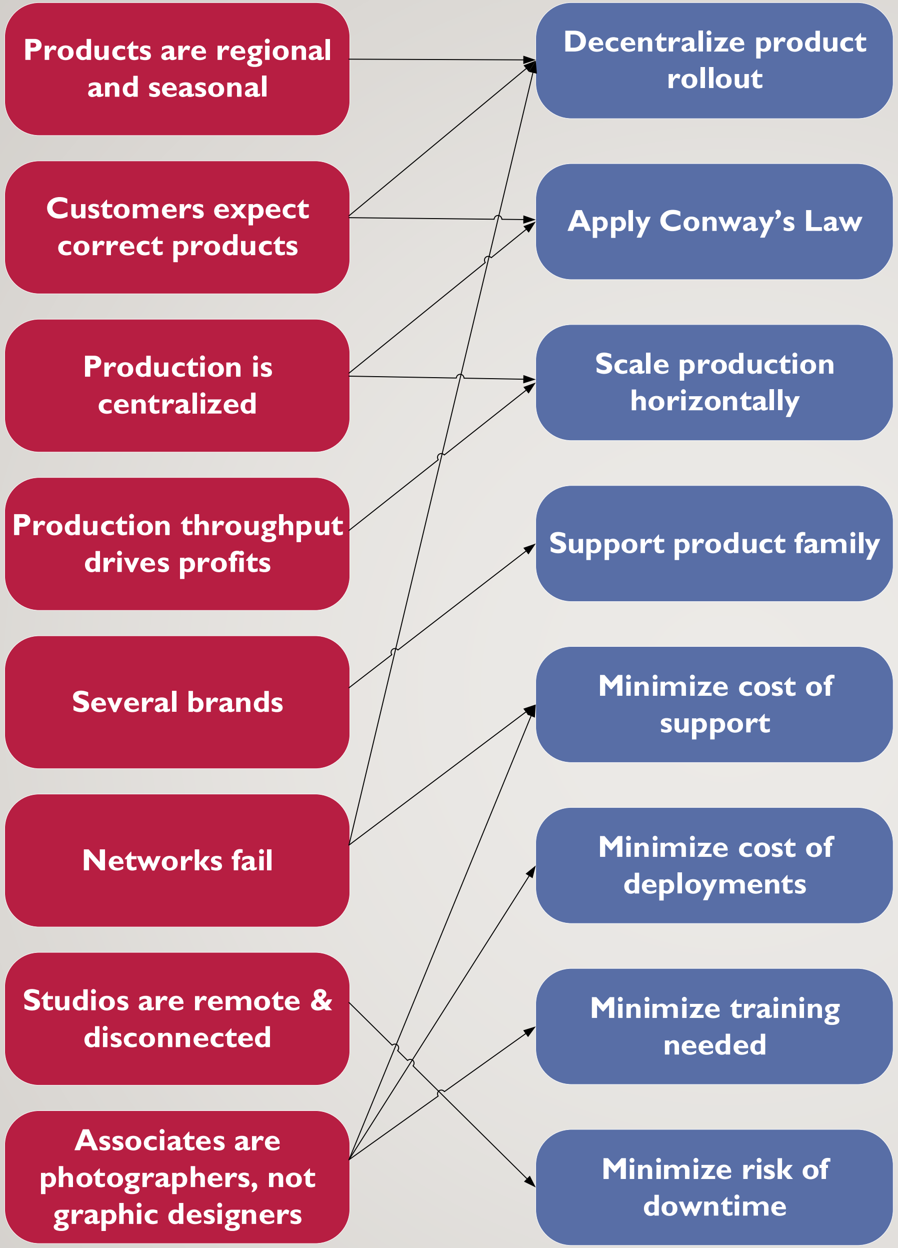
Beware target fixation

EXAMPLE: CREATION CENTER

LIFETOUCH PHOTO STUDIOS

- Embedded in other stores
- Multiple brands
- (At the time) not reliably connected
- No on-site support staff
- High turnover of associates w/seasonal hiring
- Centralized printing facility

- Products are regional and seasonal
- Customers expect correct products
- Production is centralized
- Production throughput drives profits
- Several brands
- Networks fail
- Studios are remote & disconnected
- Associates are photographers, not graphic designers



Products are regional and seasonal

Customers expect correct products

Production is centralized

Production throughput drives profits

Several brands

Networks fail

Studios are remote & disconnected

Associates are photographers, not graphic designers

Decentralize product rollout

Apply Conway's Law

Scale production horizontally

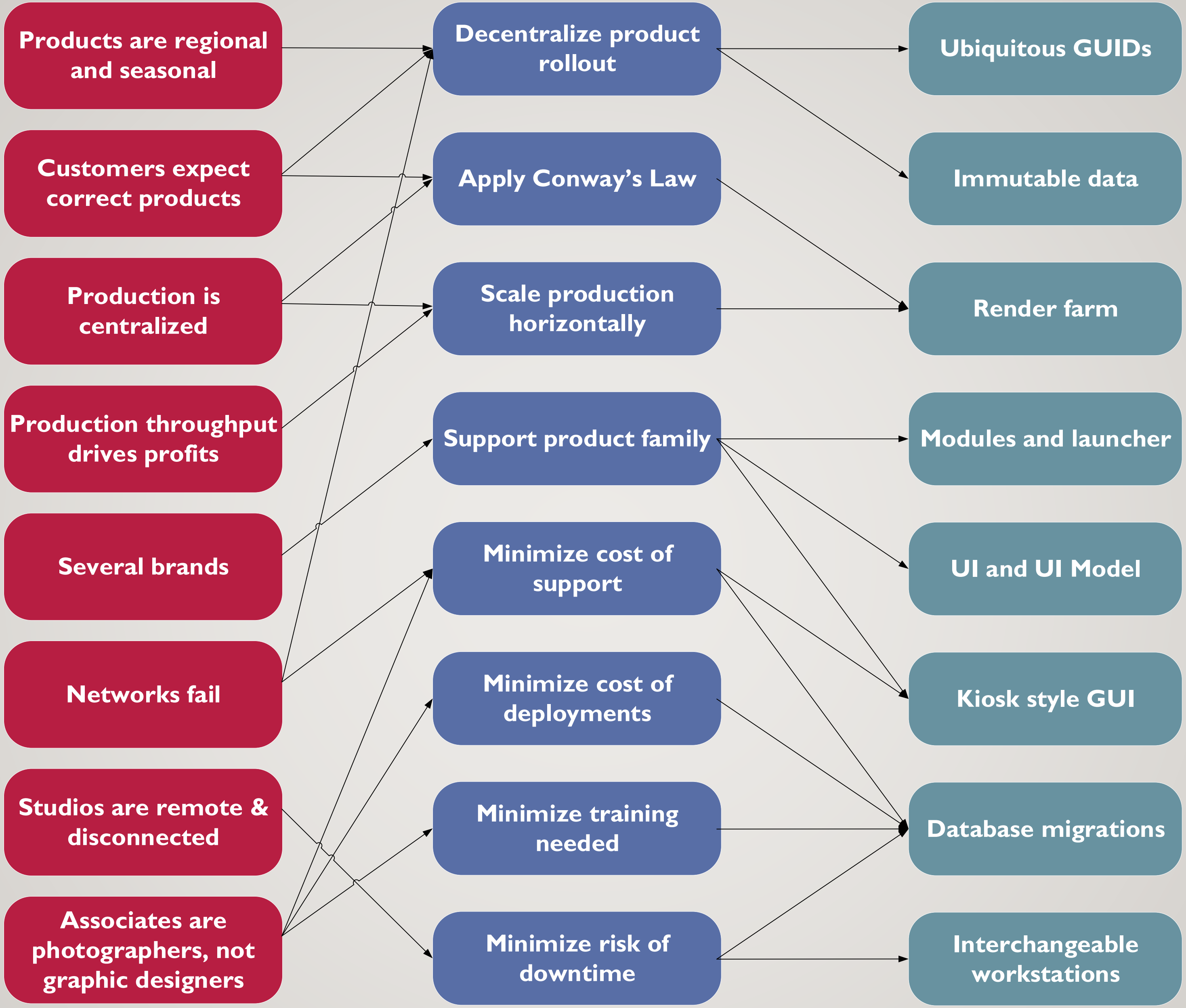
Support product family

Minimize cost of support

Minimize cost of deployments

Minimize training needed

Minimize risk of downtime



Products are regional and seasonal

Customers expect correct products

Production is centralized

Production throughput drives profits

Several brands

Networks fail

Studios are remote & disconnected

Associates are photographers, not graphic designers

Decentralize product rollout

Apply Conway's Law

Scale production horizontally

Support product family

Minimize cost of support

Minimize cost of deployments

Minimize training needed

Minimize risk of downtime

Ubiquitous GUIDs

Immutable data

Render farm

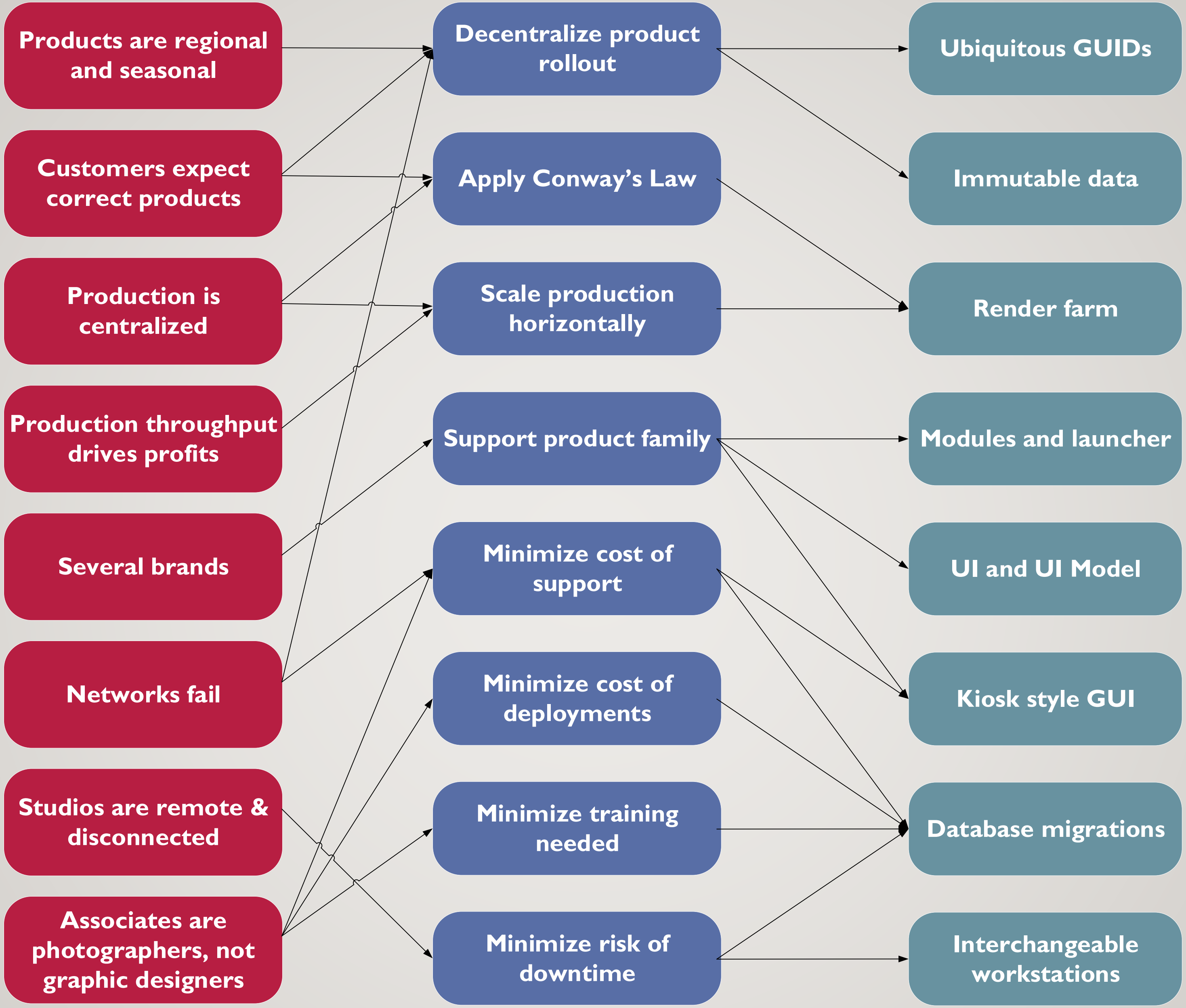
Modules and launcher

UI and UI Model

Kiosk style GUI

Database migrations

Interchangeable workstations



Products are regional and seasonal

Customers expect correct products

Production is centralized

Production throughput drives profits

Several brands

Networks fail

Studios are remote & disconnected

Associates are photographers, not graphic designers

Decentralize product rollout

Apply Conway's Law

Scale production horizontally

Support product family

Minimize cost of support

Minimize cost of deployments

Minimize training needed

Minimize risk of downtime

Ubiquitous GUIDs

Immutable data

Render farm

Modules and launcher

UI and UI Model

Kiosk style GUI

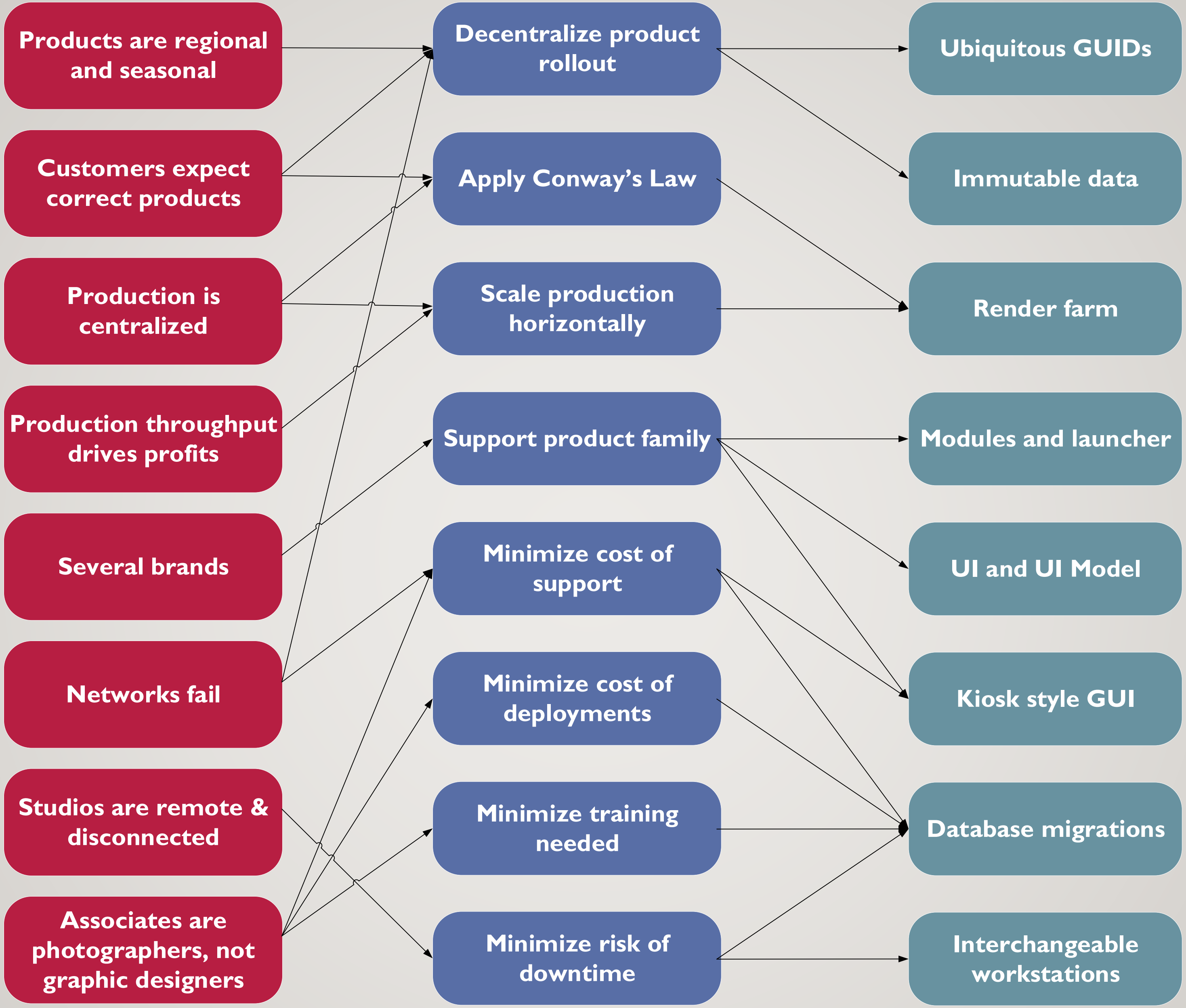
Database migrations

Interchangeable workstations

Facts

Forces

Facets



Products are regional and seasonal

Customers expect correct products

Production is centralized

Production throughput drives profits

Several brands

Networks fail

Studios are remote & disconnected

Associates are photographers, not graphic designers

Constraints

Decentralize product rollout

Apply Conway's Law

Scale production horizontally

Support product family

Minimize cost of support

Minimize cost of deployments

Minimize training needed

Minimize risk of downtime

ASRs

Ubiquitous GUIDs

Immutable data

Render farm

Modules and launcher

UI and UI Model

Kiosk style GUI

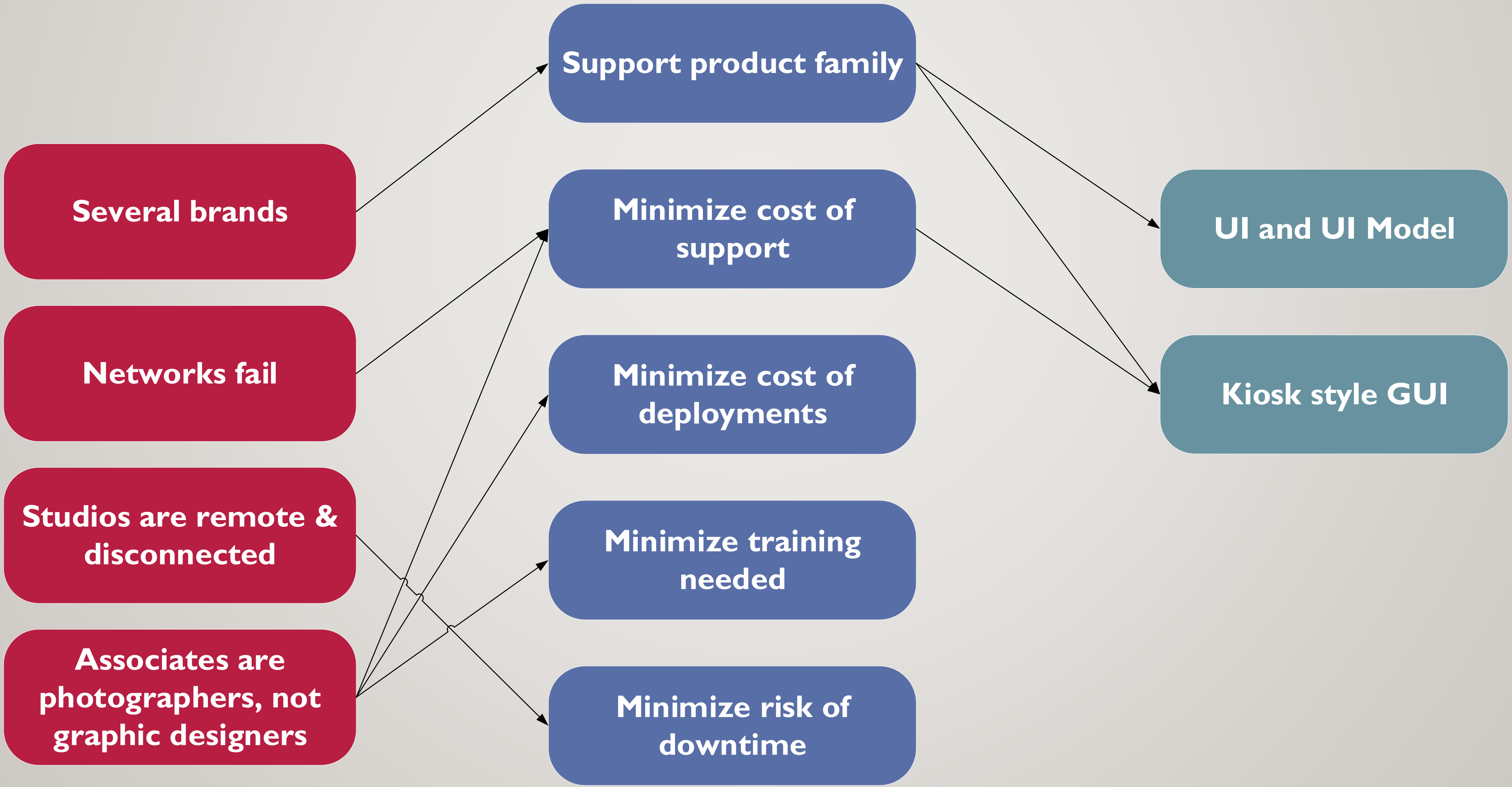
Database migrations

Interchangeable workstations

Concerns

Customers expect correct products

What we knew at the beginning.



Several brands

Networks fail

Studios are remote & disconnected

Associates are photographers, not graphic designers

Support product family

Minimize cost of support

Minimize cost of deployments

Minimize training needed

Minimize risk of downtime

UI and UI Model

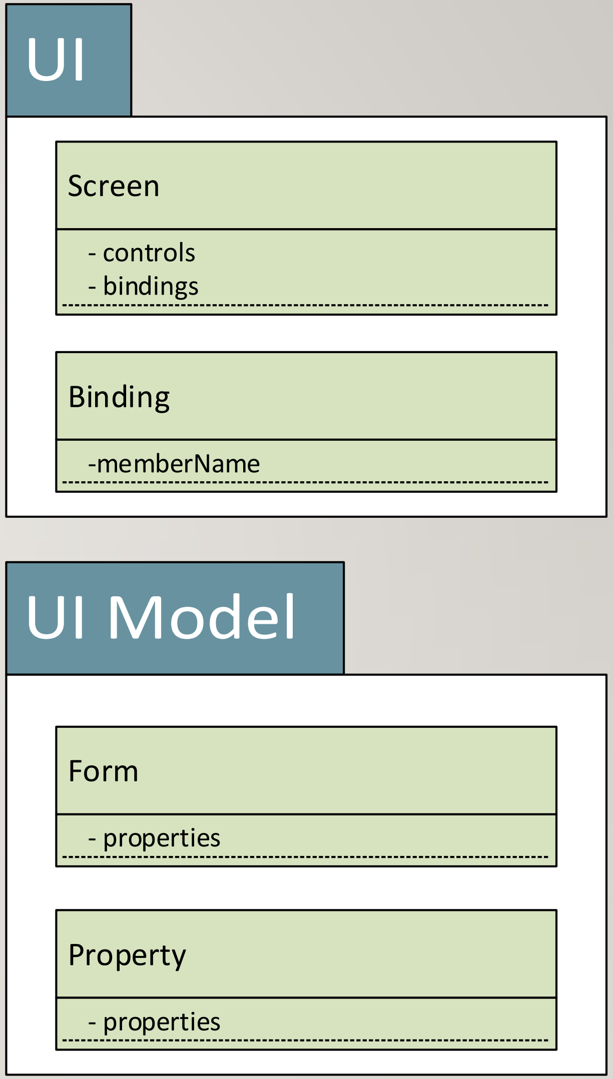
Kiosk style GUI

EXPLORE THE PROBLEM THROUGH SOLUTIONS

1. State the Problem
2. Find an Approach
3. Test a Solution
4. Find Gaps
5. Goto 1

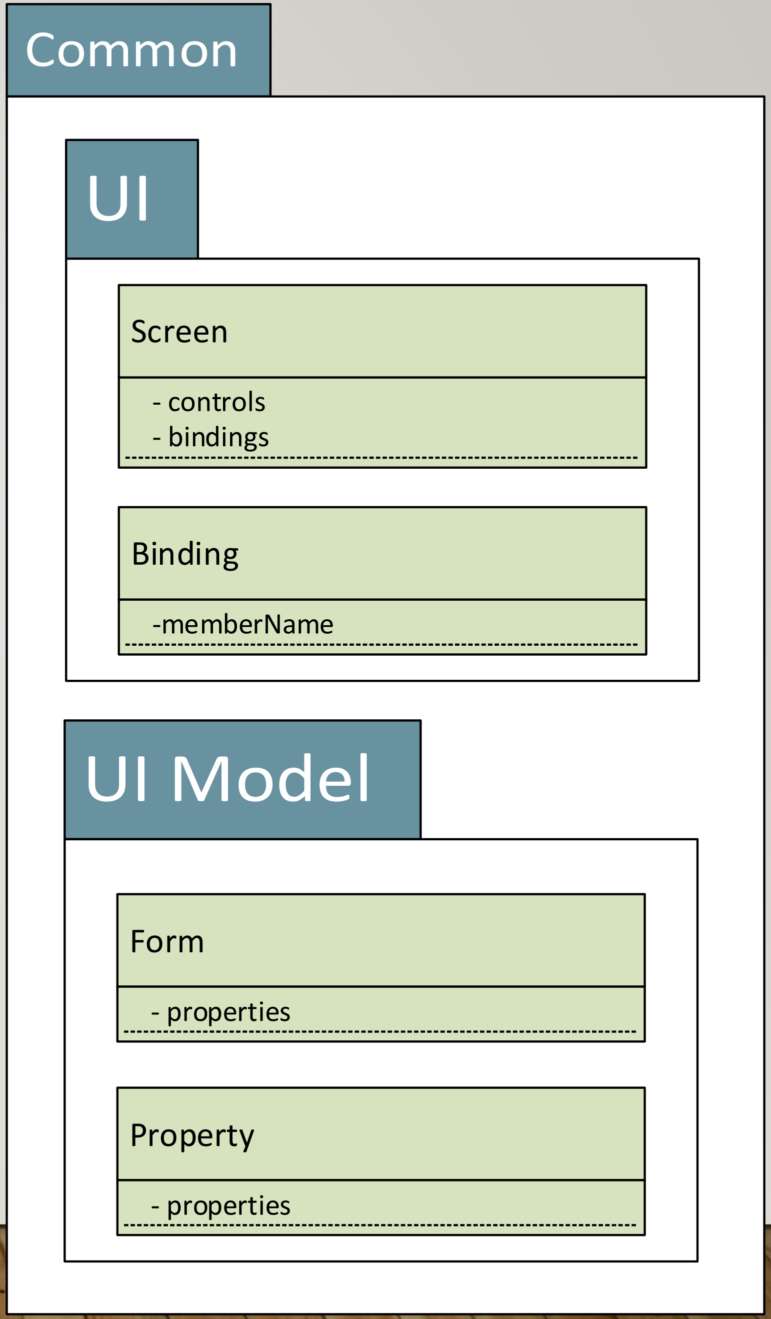
COMPOSITIONS THAT WORKED WELL

- Screen – visual. Populated with controls.
- Form – logical. Offers properties & coordinates their changes.
- Binding – mediator. Connects a property to one or more aspects of a control.



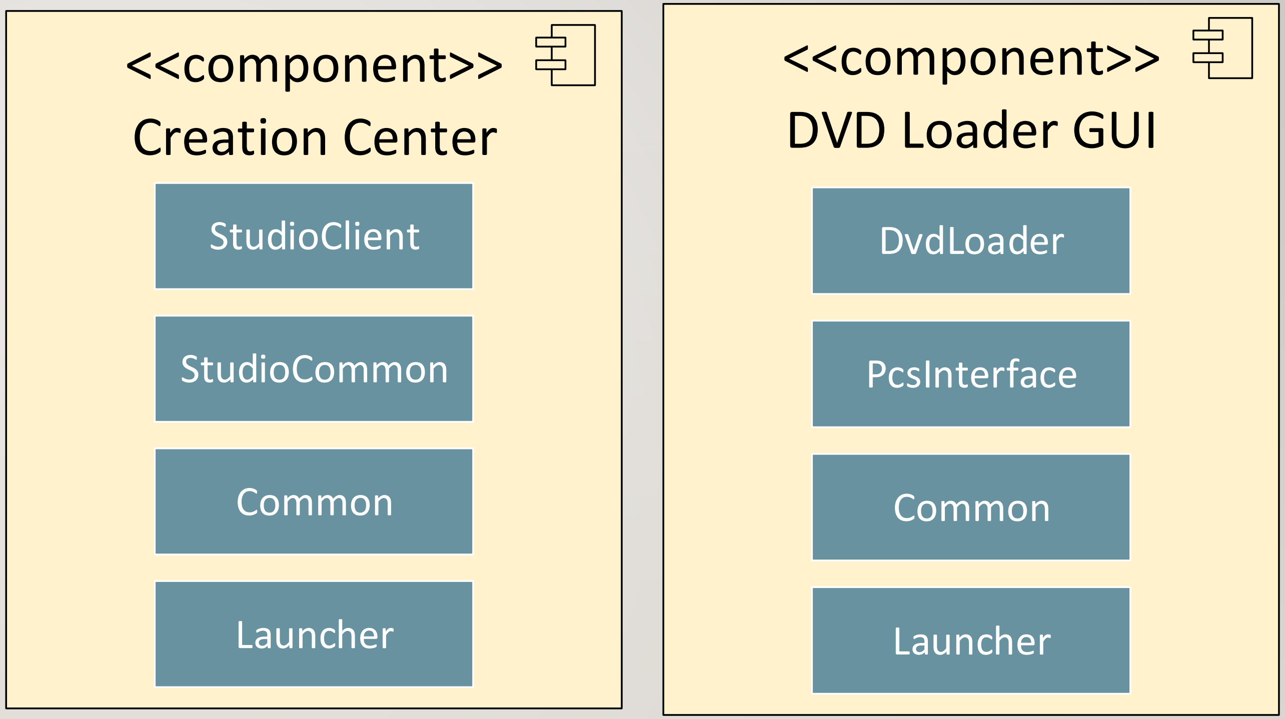
COMPOSITIONS THAT WORKED WELL

- Classes were packaged in the Common module.



COMPOSITIONS THAT WORKED WELL

- Modules bundled, run together
- `StudioClient`, `DvdLoader` and other GUI modules depend on `Common`
- DI files there create Form classes, but only instances of Property, Screen, Control, & Binding objects.



FROM MODULES TO COMPONENTS

We could combine modules into components. They didn't care what was in the component.

The UI machinery didn't care how it was packaged.

Deciding which GUI modules to use didn't impose any constraints on packaging.

Deciding on packaging didn't impose any constraint on the GUI.

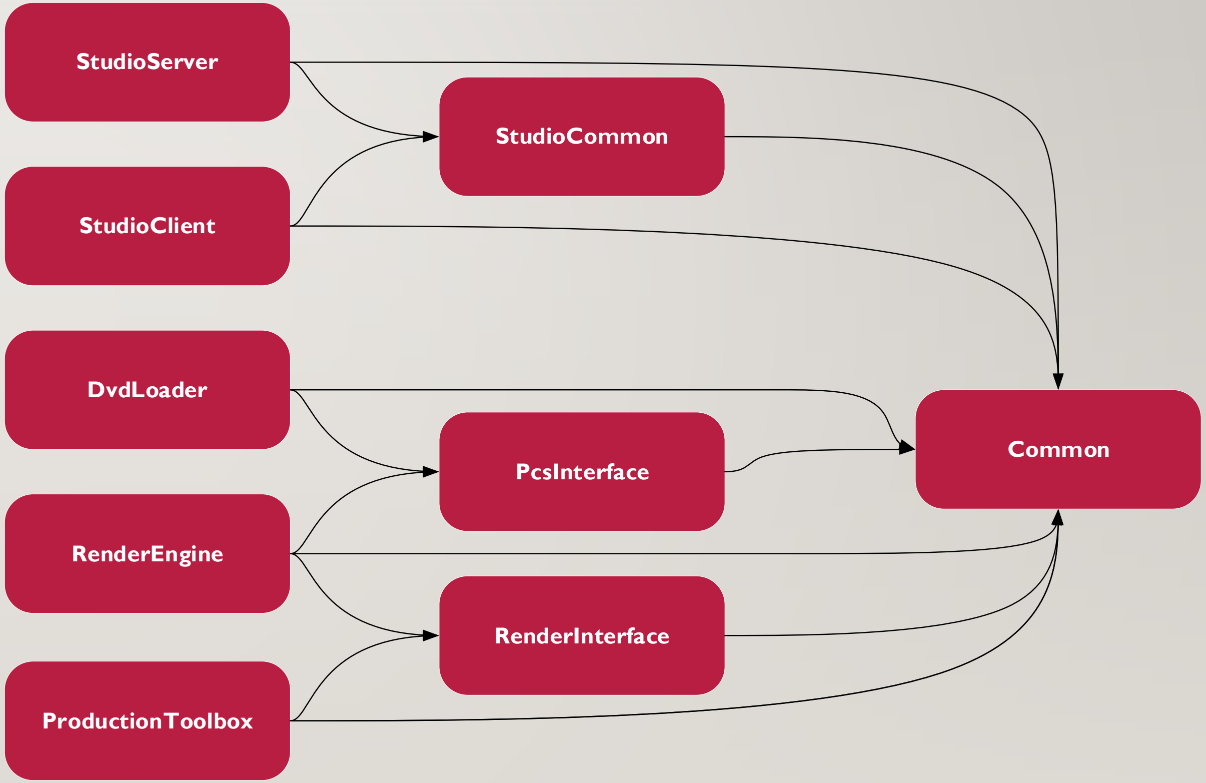
That's orthogonality.

THIS WAS ALSO INCREMENTAL ARCHITECTURE

1. Initial concept of layers
2. Property-binding architecture
3. Studio server vs Studio client
4. Use Spring for modules
5. Launcher builds classpath & configpath
6. Database migrations
7. Build setup.exe from CI, with test installation
8. Production interface, render farm, toolbox
9. Product creation GUI

BUT SOME CHALLENGES

- Common was the remains of our original monolithic project.
- *Everything* coupled to Common.



BUT SOME CHALLENGES

- Later, stores got connected.
- But the idea of a DVD was baked in hard
- That's what happens when ASRs and fundamental constraints change!

LOCALITY

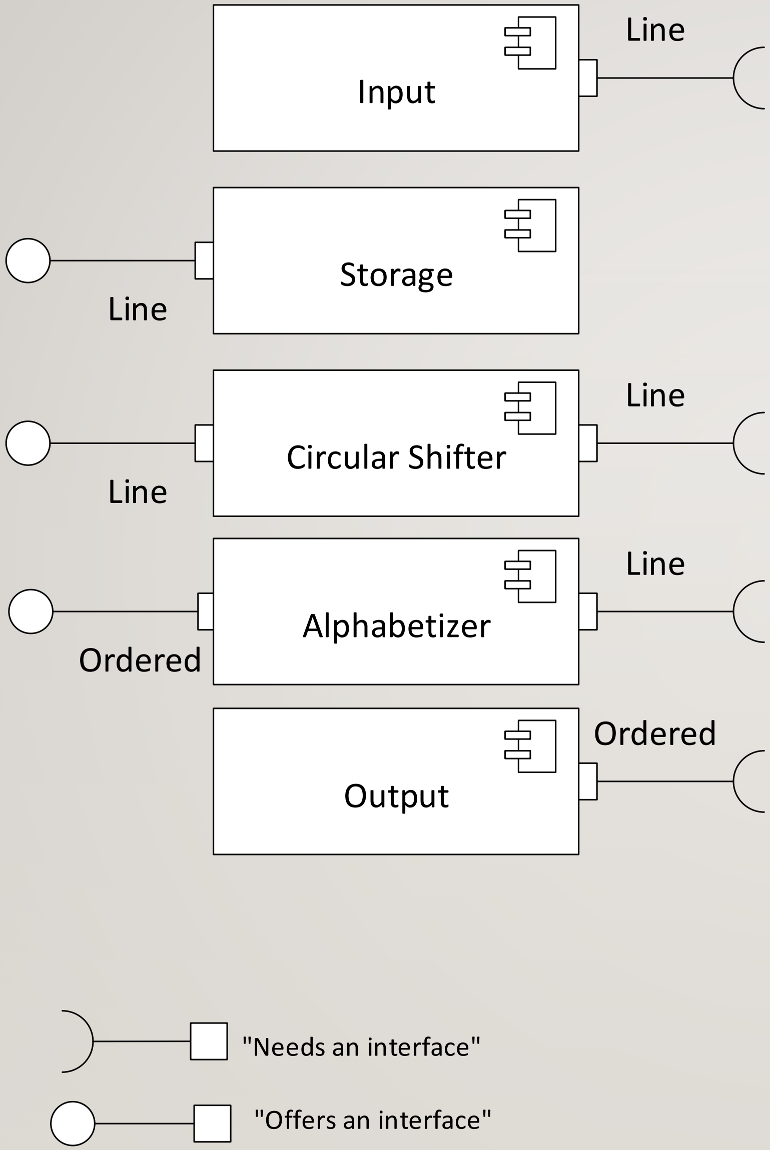
LOCALIZE DECISIONS; DON'T RYI

(Reveal Your Implementation)

- Don't let entity types proliferate through systems. Keep them local.
- Use common interfaces to avoid RYI
- Use common representations/media types/data formats to avoid RYI

RECALL THE KWIC INDEX

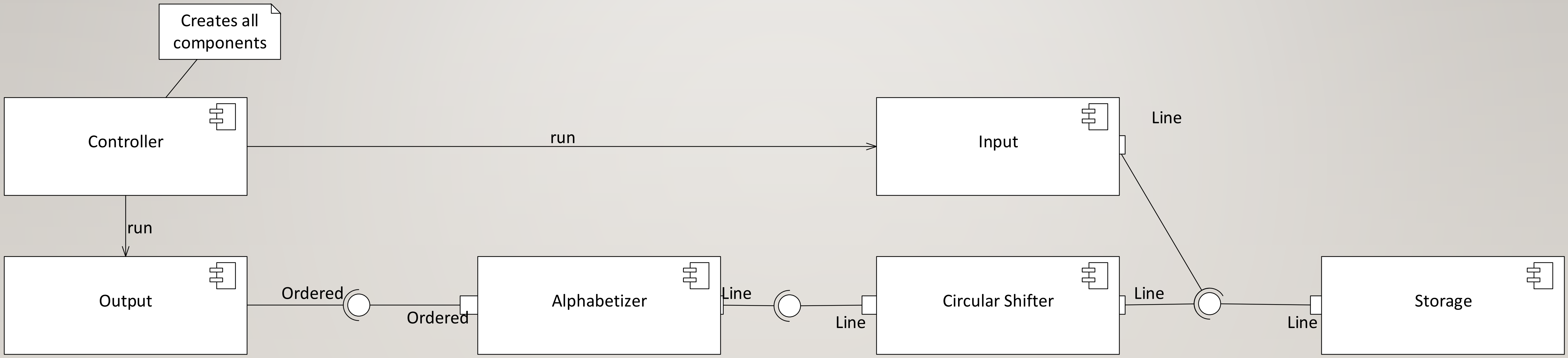
1. **Line Storage**
   Offers functional interface: SETCH, GETCH, GETW, DELW, DELLINE
2. **Input**
   Reads EBCDIC chars, calls line storage to put them into lines.
3. **Circular Shifter**
   Offers same interface as line storage. Makes it appear to have all shifts of all lines.
4. **Alphabetizer**
   Offers sort function INIT, and access function iTH that gets a line.
5. **Output**
   Repeatedly call iTH on alphabetizer, printing the line.
6. **Control**
   Similar to first approach, call each module in sequence.



WHY DID THE SECOND MODULARIZATION SURVIVE CHANGE BETTER?

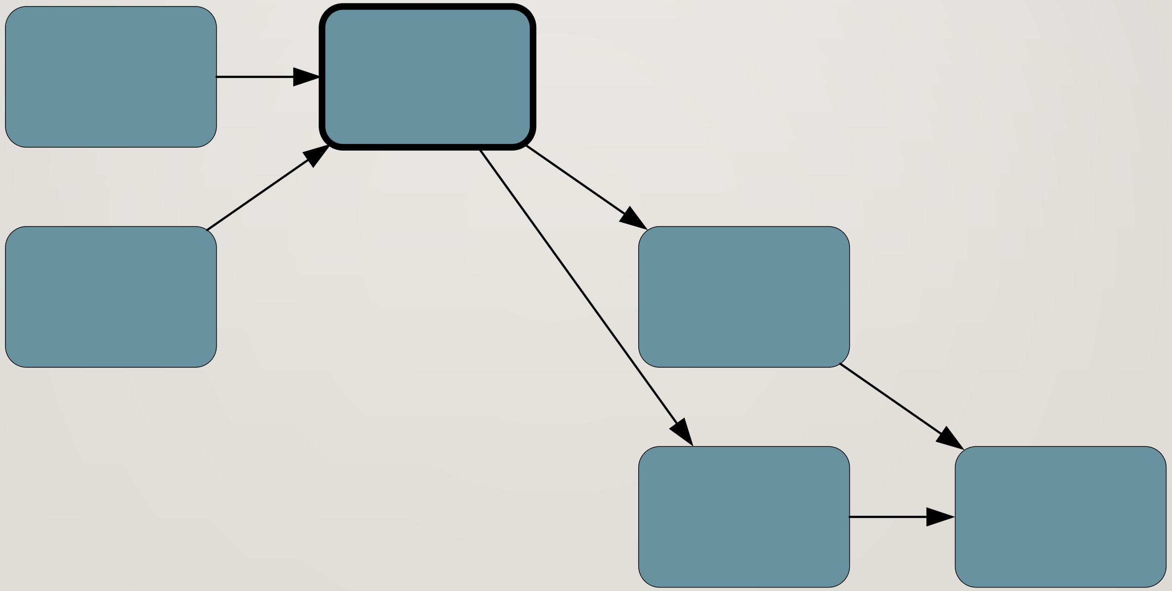
- Very few data types
- Small number of well defined interfaces
- Highly composable
- Limited RYI

WHY DID THE SECOND MODULARIZATION SURVIVE CHANGE BETTER?

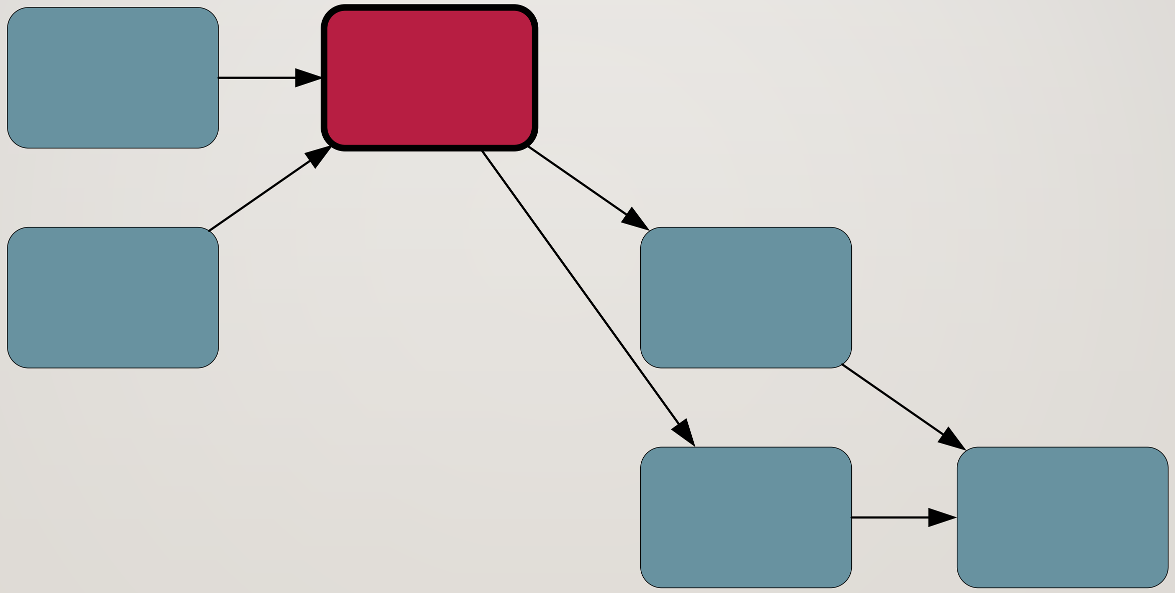


UPSTREAM AND DOWNSTREAM

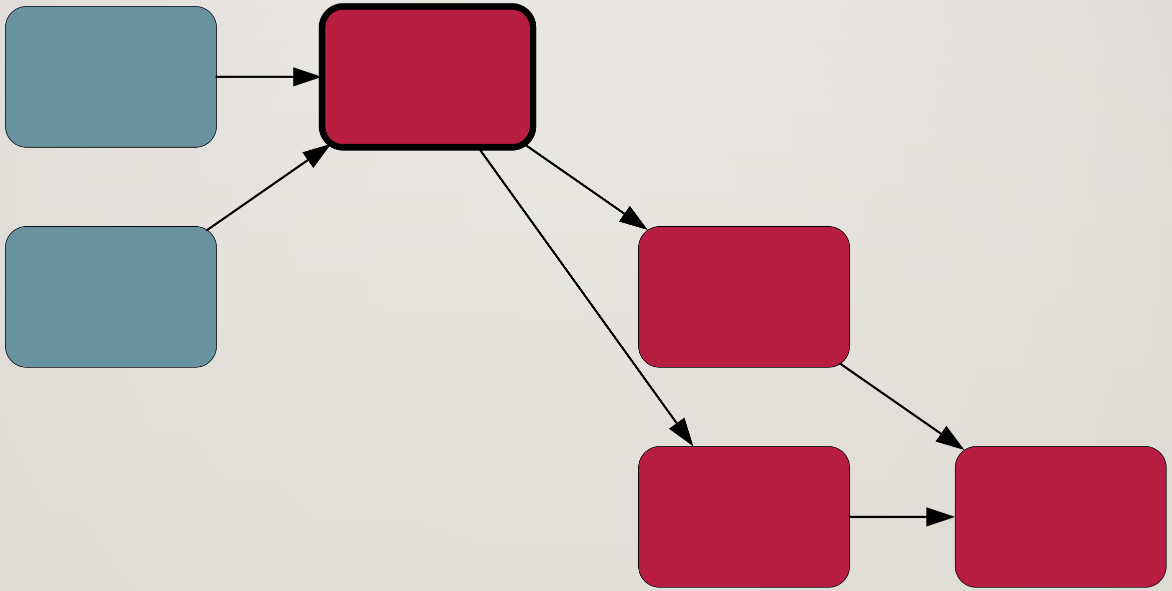
WE WORK ON ONE OR TWO COMPONENTS AT A TIME



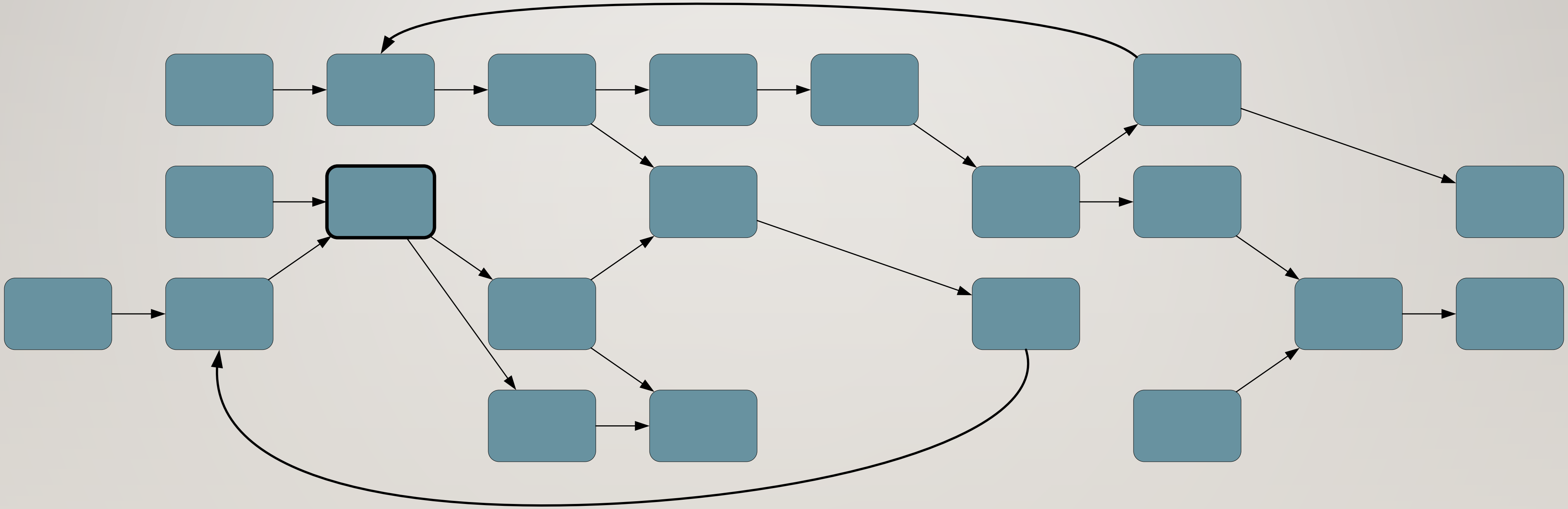
SO WE MAKE A CHANGE



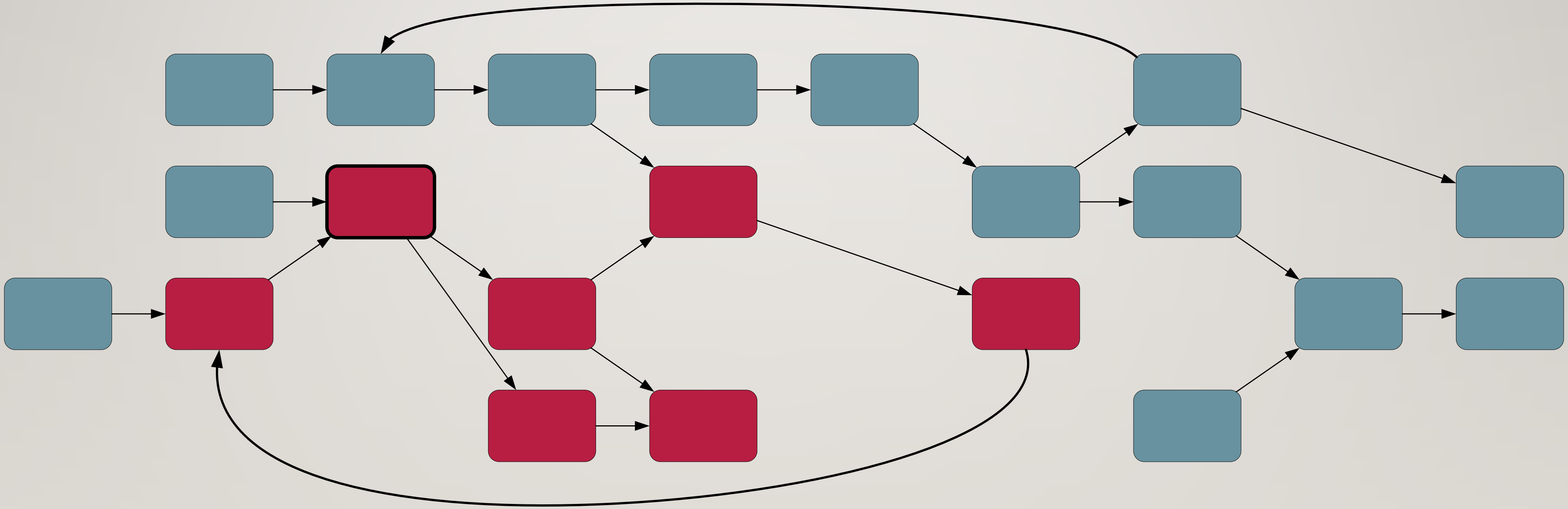
THAT HAS A RIPPLE EFFECT



BUT THE ENTERPRISE REALLY LOOKS LIKE THIS



AND OUR CHANGE HAS A BIG "SURFACE AREA"



REDUCING THE SURFACE AREA OF CHANGE

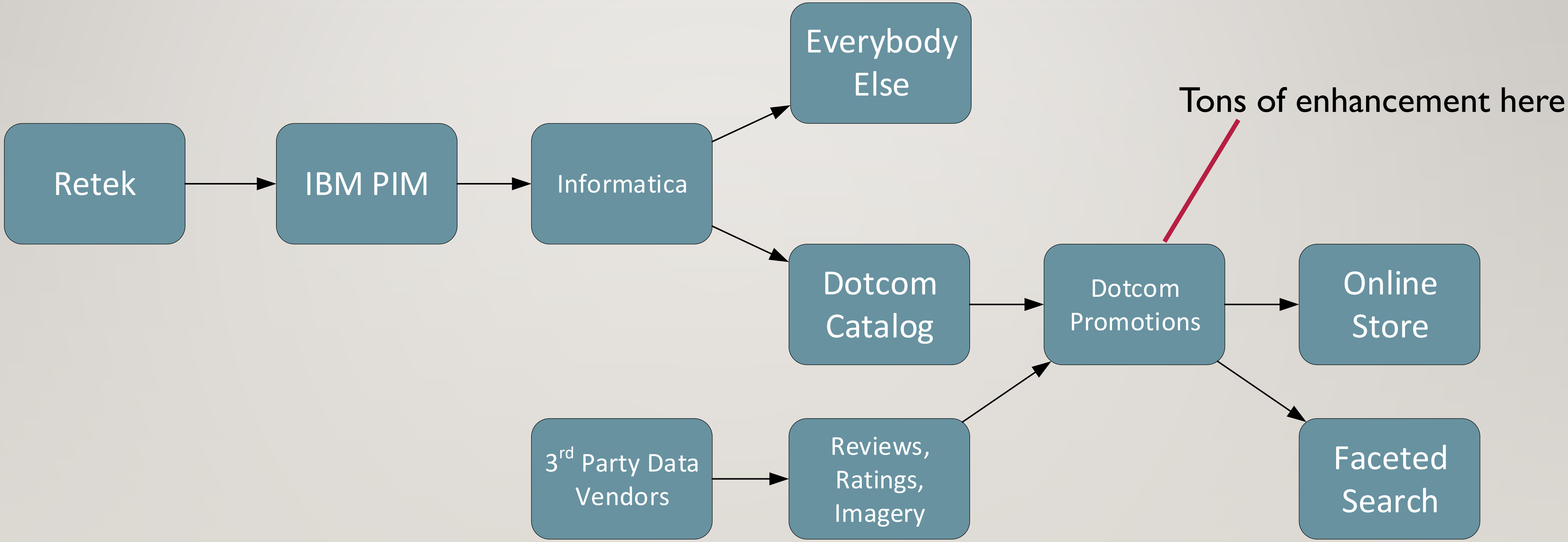
1. Augment Upstream
2. Contextualize Downstream

AUGMENT UPSTREAM

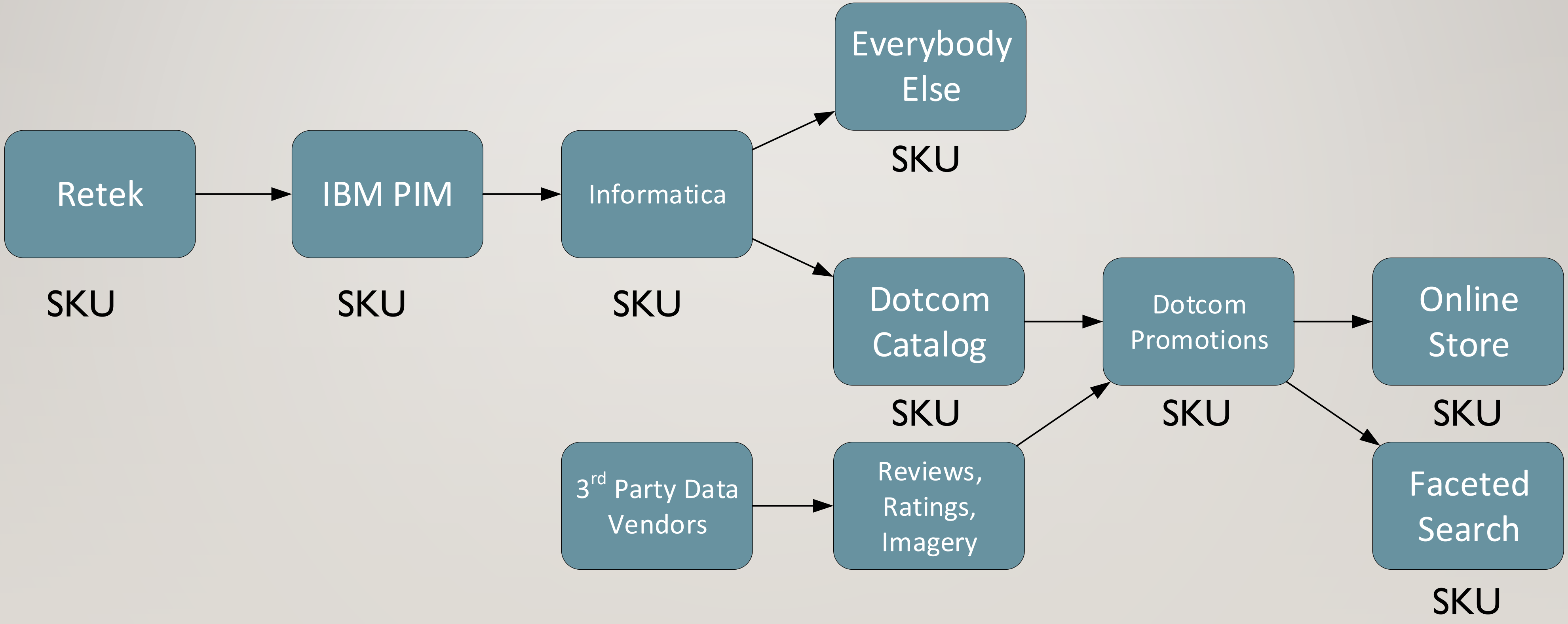
AUGMENTING

- Add to data as "early" as possible
  - Combine sources
  - Add human judgement
  - Apply ML models
- Avoid creating privileged downstreams
- Everybody wants the best data available

COUNTEREXAMPLE



ALSO AN EXAMPLE OF SEMANTIC COUPLING



SKU WAS A COMPOSITE

- Many types of attributes carried together
- Historically, these were **always** a unit
- People thought of "SKU" as a real thing, forgot that it's just a label for a collection of attributes that sometimes describe the same thing.

- More to that story later…

COGS
Distribution
Stocking
Presentation
Pricing
Delivery
Inventory

KINDS OF AUGMENTATION

- Adding attributes
- Connecting entities from different sources
- Adjusting cardinalities
- Making aggregates
- Adding derived or discovered attributes

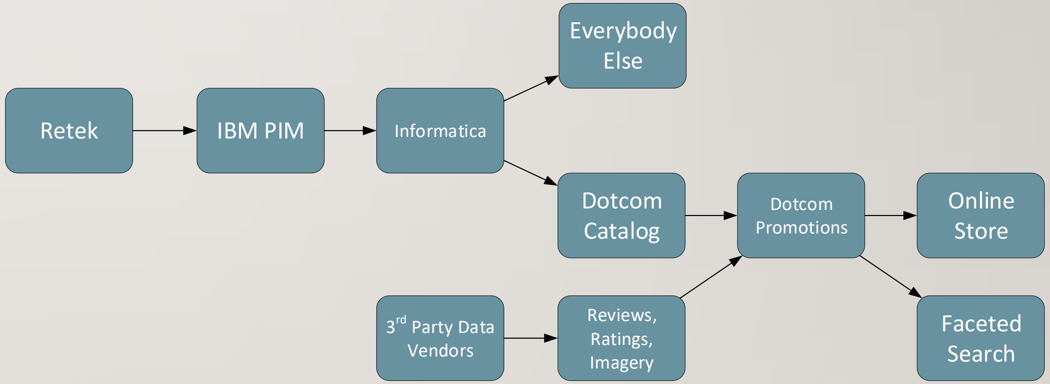
CONTEXTUALIZE DOWNSTREAM

CONTEXTUALIZING

- Applying policies and restrictions
- “isValid”
- Limiting the extent of an entity (i.e., restricting which instances to offer)
- Limiting the breadth of an entity (restricting which attributes to offer)

EXAMPLE: STREET DATE

- “Street date” – released for sale
- SKUs not passed from PIM until after street date
- Decision about display to end customer also impacted users of internal systems
- Cannot prepare for online display
- Cannot take pre-orders!



EXAMPLE: STREET DATE

- Augment upstream:

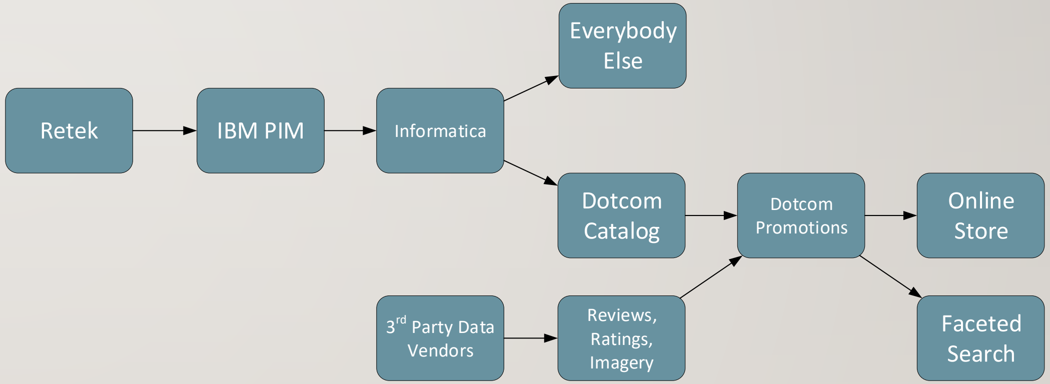
  Add attribute "street date in past?"

- Contextualize downstream:

  Send the SKUs,

  GUIs decide whether to show

  APIs decide whether to show


Retek
IBM PIM
Informatica
Everybody Else
Dotcom Catalog
Dotcom Promotions
Online Store
Faceted Search
3rd Party Data Vendors
Reviews, Ratings, Imagery