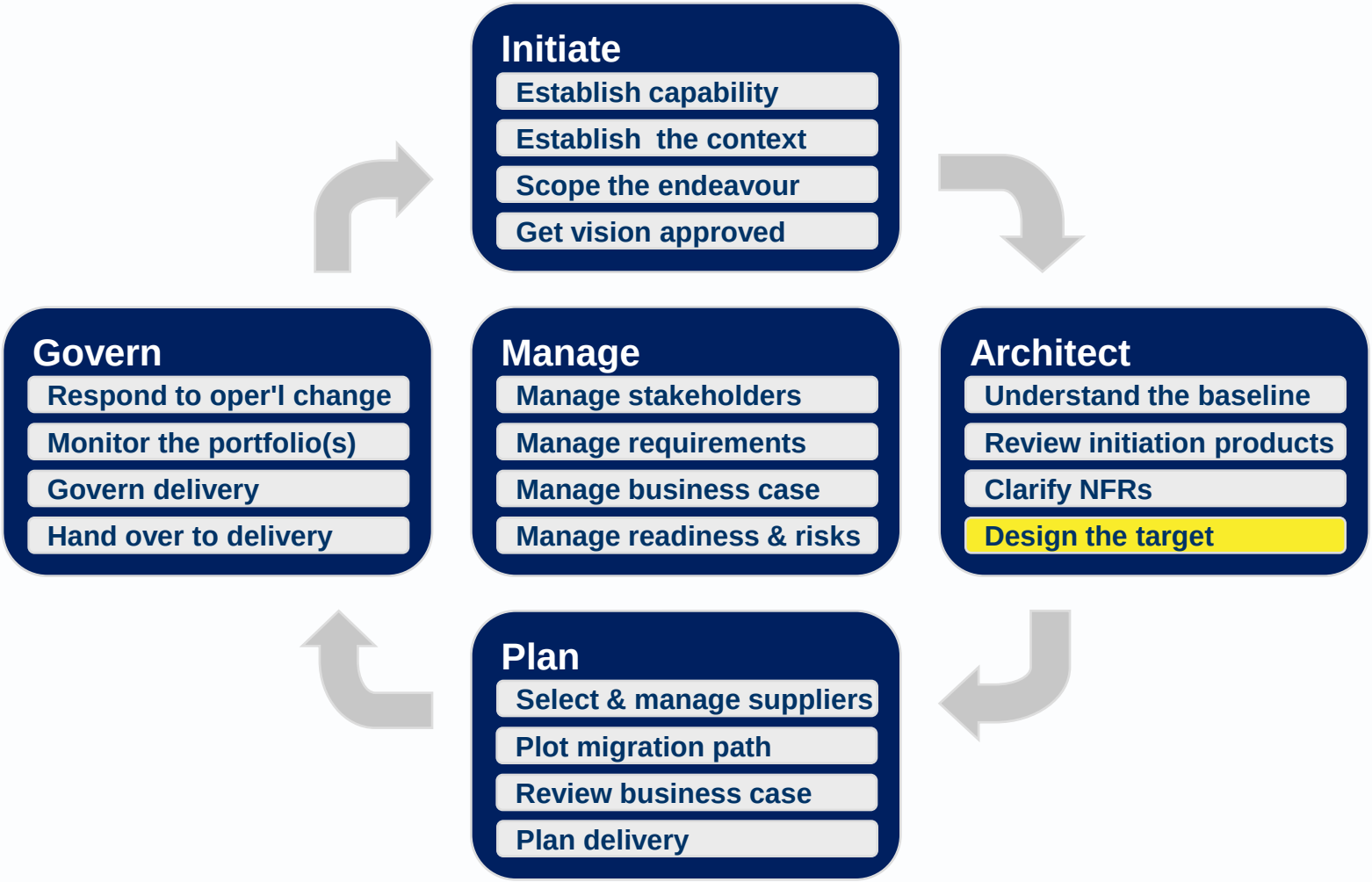
Avancier Methods (AM)
Applications Architecture

Design the target Applications Architecture

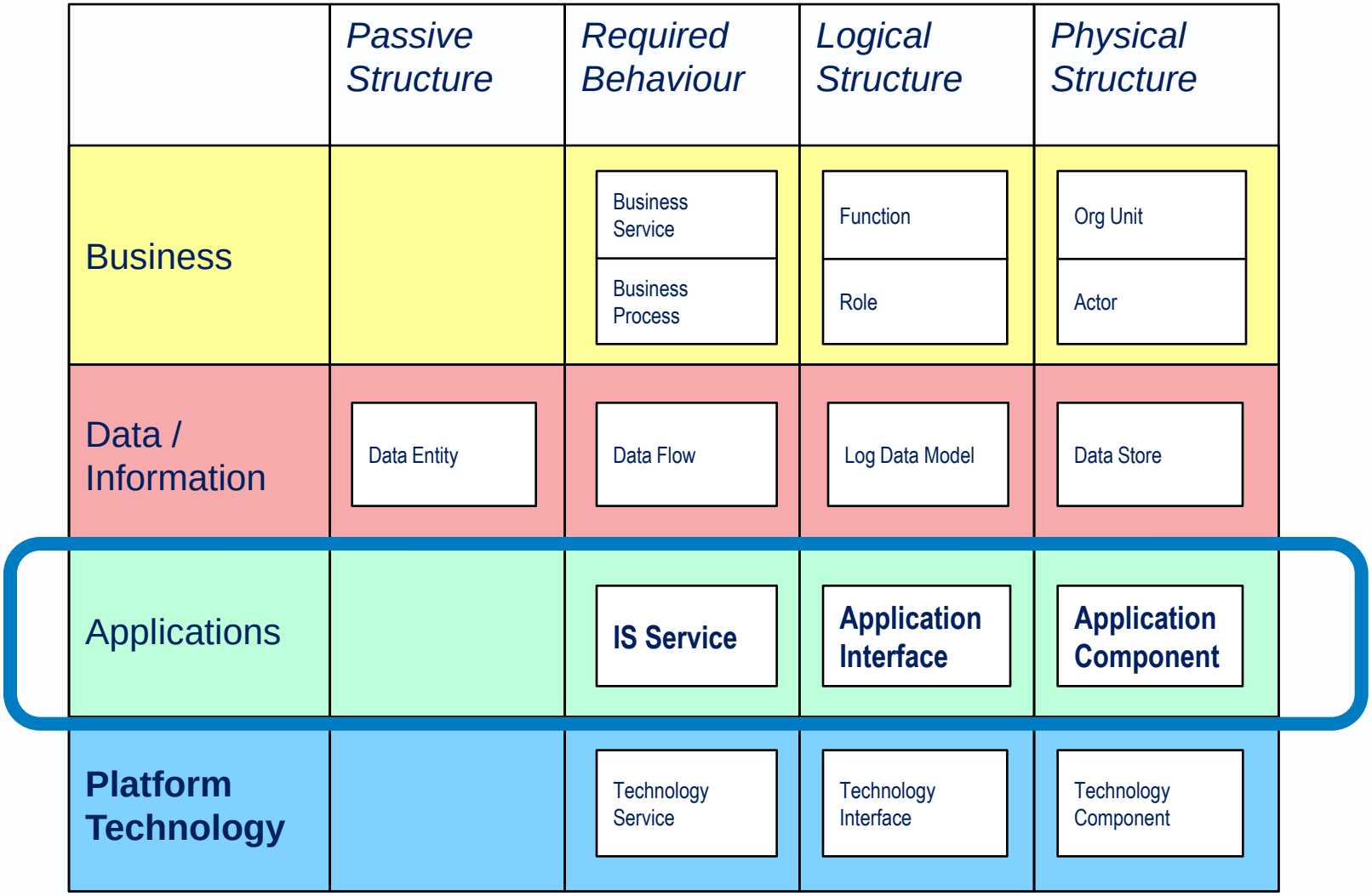
CONTEXT

- What is the AM level 2 process?
- Which domain are we working in?
- What is the AM level 3 process?

AM level 2 process



Which domain are we working in?



AM level 3/4 process: design applications architecture (EA)

Avancier

Design target applications architecture

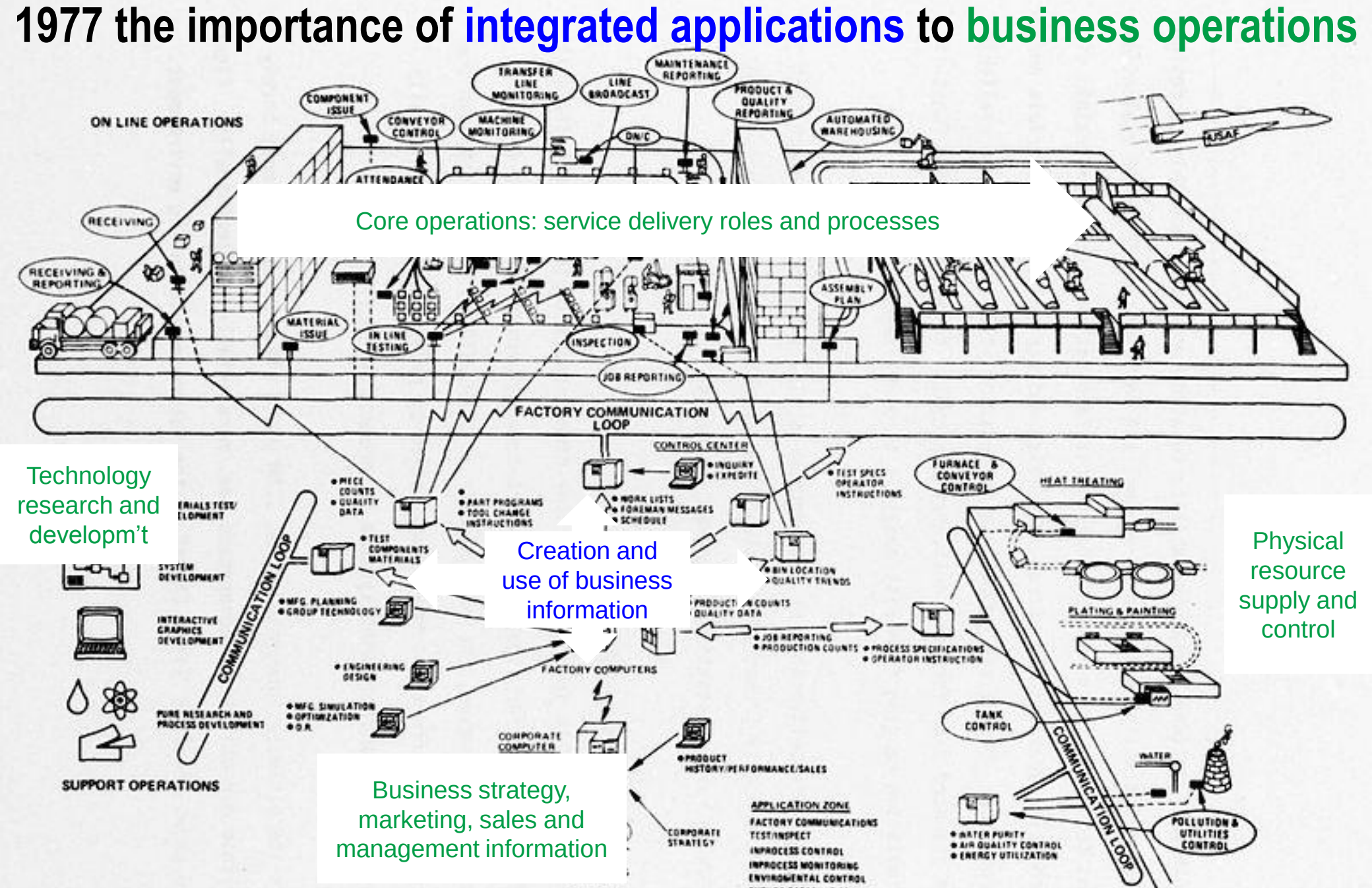
1. Scope application changes
2. Identify data flows, data stores and applications in scope
3. Select best-fitting Application Integration Pattern
4. Draw application communication diagram (aka DFD)
5. Draw sequence diagrams for key processes

Scope application changes

- Consider “request for architecture work”
- Consider enterprise app road map (if there is one)

App	2016	2017	2018	2019
ERP 1	Ignore	Ignore	Remove	
ERP 2			Deploy	Improve
CRM 1	Remove			
CRM 2	Deploy	Improve	Prize	Prize
Billing	Prize	Prize	Prize	Prize
DW/BI	Improve	Improve	Improve	Improve
Timesheet	Ignore	Rewrite	Prize	Prize

1977 the importance of integrated applications to business operations



Core operations: service delivery roles and processes

Technology research and developm't

Creation and use of business information

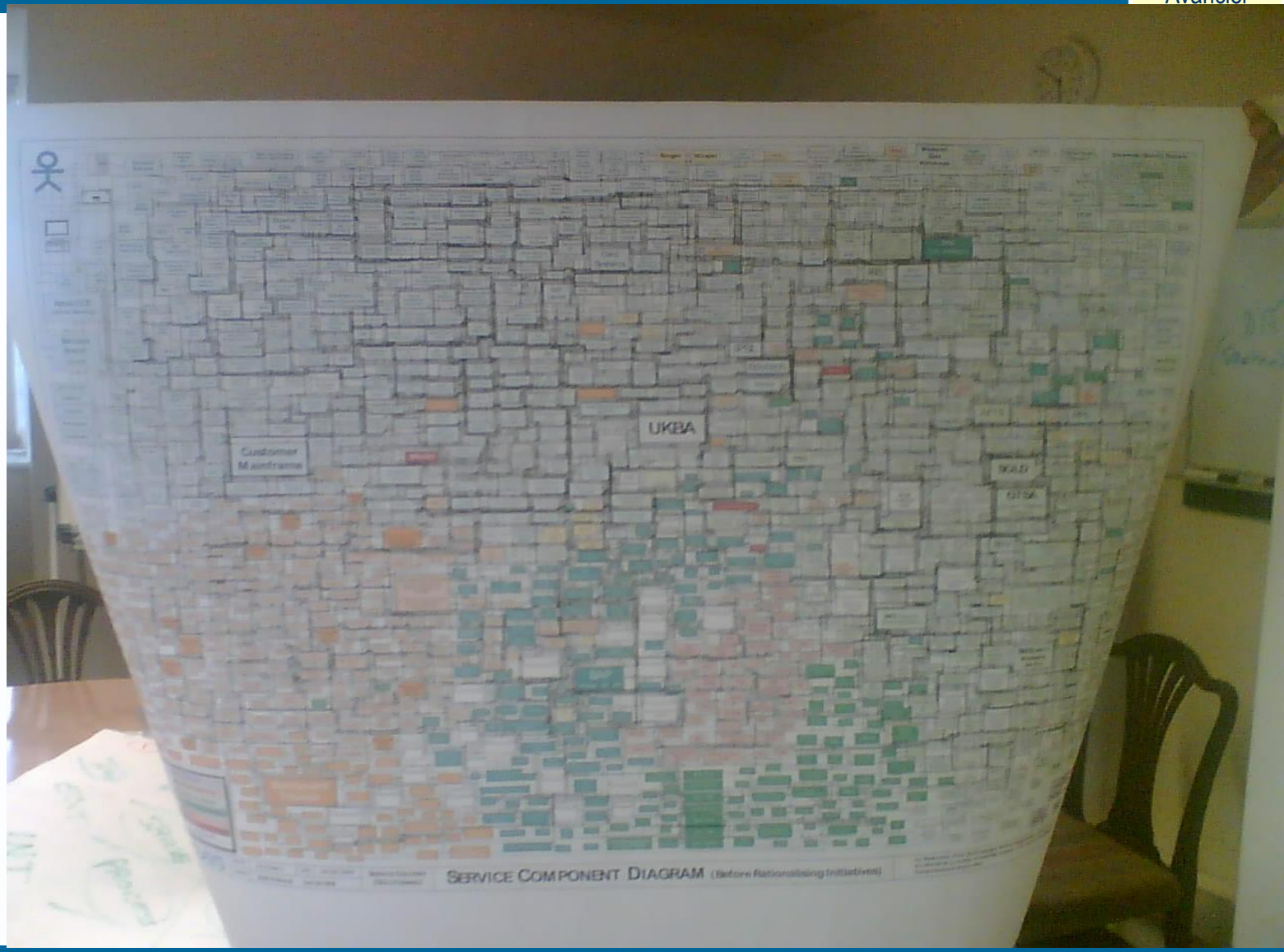
Physical resource supply and control

Business strategy, marketing, sales and management information

By Dennis E. Wisnosky - An overview of the Air Force program for Integrated Computer Aided Manufacturing (forerunner of IDEF). ICAM program prospectus. SME technical paper, Public Domain, https://commons.wikimedia.org

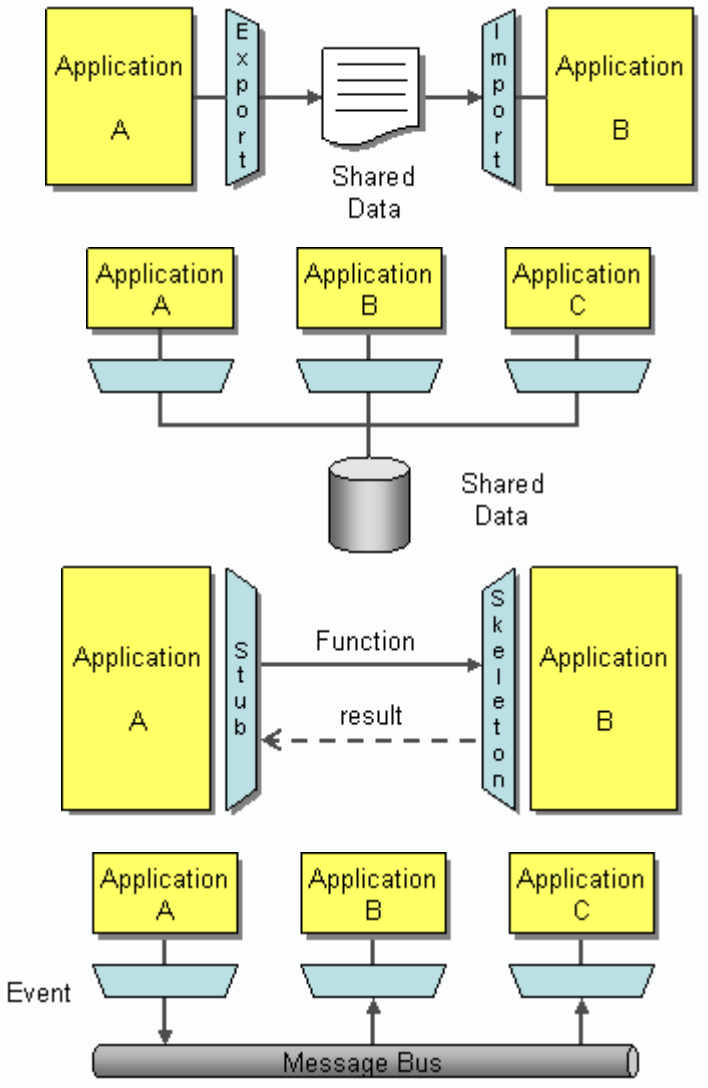
Identify data flows, data stores and applications in scope

- The diagram shows point to point data flows between c1,000 applications
- **How is that data transported?**



7.4 Applications Integration Tools (rarely examined)

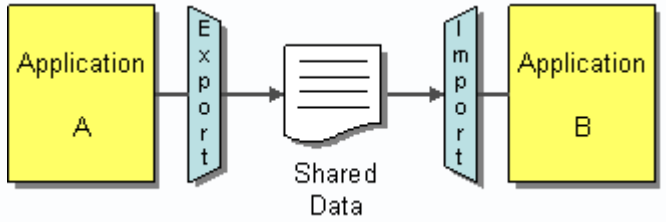
- ETL middleware
- Shared Data Space
- Point to Point (RPC/RMI/ORB)
- Web Service
- EAI/ESB Middleware



- File Transfer
- Shared Database
- Remote Procedure Invocation
- Messaging

ETL tools (Oracle, Ab Initio, Informatica, Power BI)

ETL middleware

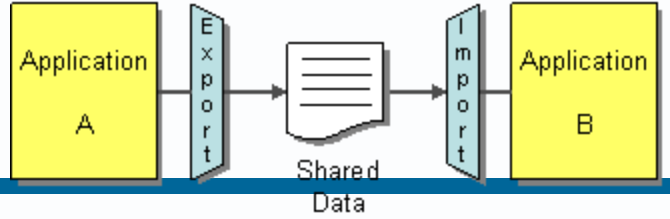


- Tools that help you to
  - E: Extract data from data sources/senders,
  - T: Transform data items from one format to another, and
  - L: Load the reformatted data into data stores

Useful for

- Loading a data warehouse on a regular basis
- Loading a database during a one-off data migration
- Move bulk data between databases

Where to clean the data?

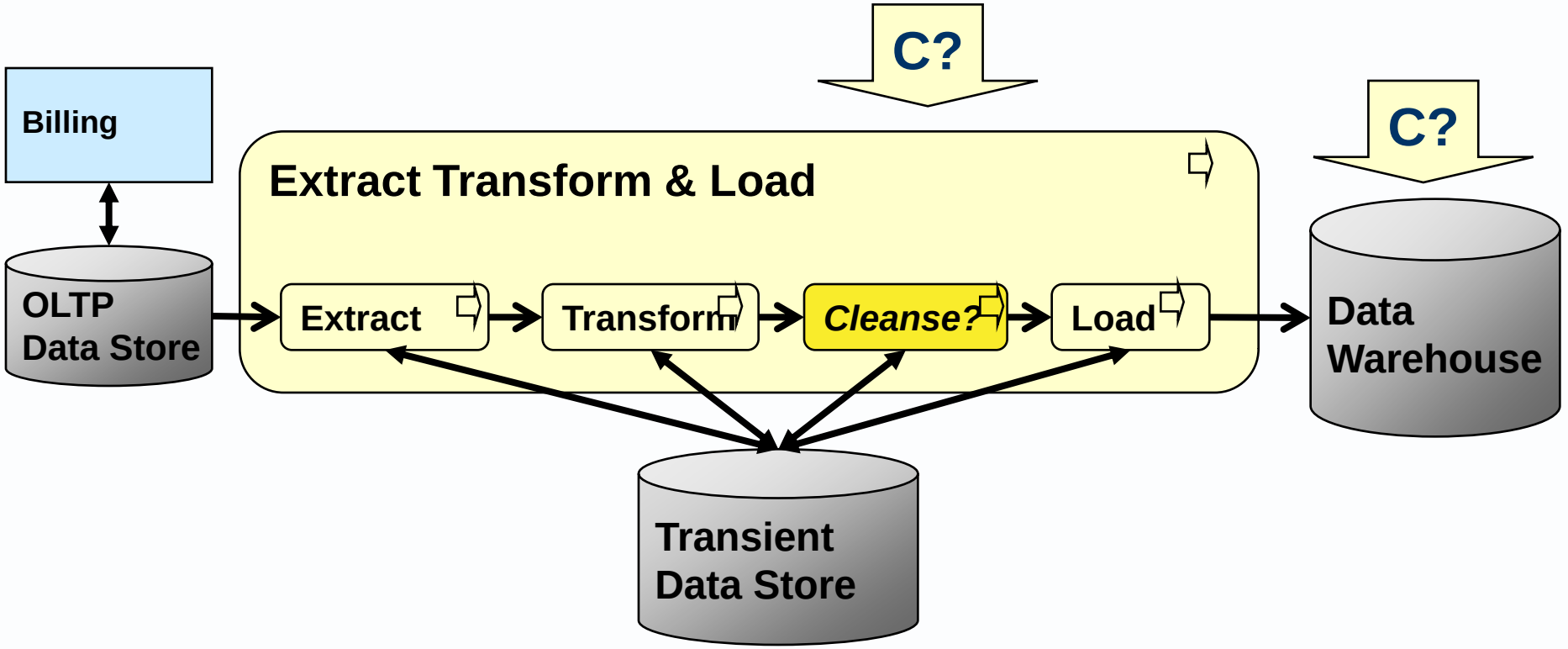


Avancier

Often requires data to be cleaned up before or after the transformation stage,

CETL? ECTL? ETCL? ETLC?

Or do not clean at all at first?



Shared Data Space

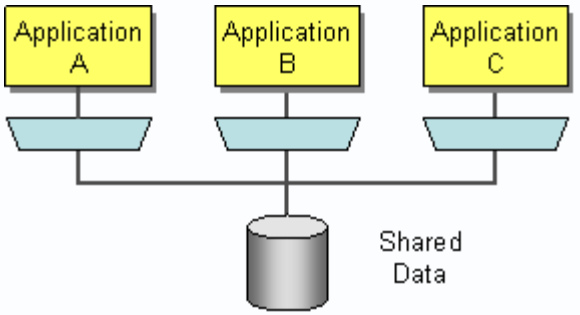
ETL middleware

Shared Data Space

Point to Point (RPC/RMI/ORB)

Web Service

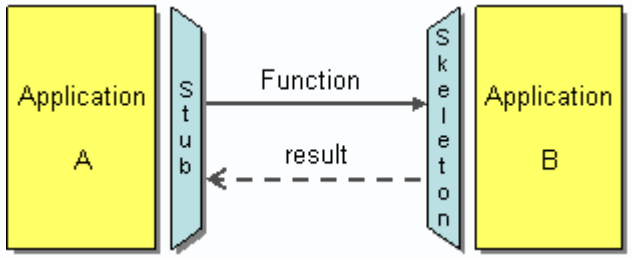
EAI/ESB Middleware



Point to point and web services tools

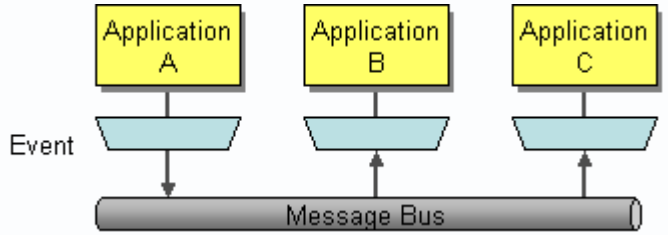
Point to Point (RPC/RMI/ORB)

Web Service



- **RPC:** [a process] by which a process on one computer calls a process on another computer. It is more complex, slower and less secure than a local procedure call. The term usually implies a synchronous request-reply style of interoperation.
  - **Object request broker (ORB):** Like RPC with extra features. It enables the objects of an OO program to be distributed. Software is coded as though all objects are on one computer. The ORB handles the distribution of objects between computers. So (in theory) the distributed system behaves like one OO program. It may add transaction management, security and other features. OMG's CORBA emerged as the standard.
- **Web Service**: [a component] that can be invoked over "the web" using an internet protocol and a published interface. It uses open standards like WSDL, XML and SOAP, but no particular standard is widely agreed as constraining what the term means.

EAI / ESB tools



- **ESB Middleware:** A platform application component that may

  - manage message queues
  - store, route and forward messages between distributed components
  - transform messages between protocols
  - transform messages between data formats

  - use a *canonical data model* in data format transformation
  - manage *federated/distributed transactions*
  - host procedures/workflows that *orchestrate* distributed components.
  - support EDA using *pub/sub* mechanisms

EAI/ESB Middleware

Messaging patterns

EAI / ESB Middleware

According to Gregor Hohpe
http://www.enterpriseintegrationpatterns.com/
After “Enterprise Integration Patterns” - the Book

Avancier

Messaging Systems	Message Routing	Messaging Endpoints
Message Channel	Content-Based Router	Messaging Gateway
Message	Message Filter	Messaging Mapper
Pipes and Filters	Dynamic Router	Transactional Client
Message Router	Recipient List	Polling Consumer
Message Translator	Splitter	Event-Driven Consumer
Message Endpoint	Aggregator	Competing Consumers
	Resequencer	Message Dispatcher
	Composed Msg. Processor	Selective Consumer
	Scatter-Gather	Durable Subscriber
	Routing Slip	Idempotent Receiver
	Process Manager	Service Activator
	Message Broker	
Messaging Channels	**Message Transformation**	**System Management**
Point-to-Point Channel	Envelope Wrapper	Control Bus
Publish-Subscribe Channel	Content Enricher	Detour
Datatype Channel	Content Filter	Wire Tap
Invalid Message Channel	Claim Check	Message History
Dead Letter Channel	Normalizer	Message Store
Guaranteed Delivery	Canonical Data Model	Smart Proxy
Channel Adapter		Test Message
Messaging Bridge		Channel Purger
Message Bus		
Message Construction	**Interlude: Composed Messaging**	
Command Message	Synchronous (Web Services)	
Document Message	Asynchronous (MSMQ)	
Event Message	Asynchronous (TIBCO)	
Request-Reply	Command Message	
Return Address	Document Message	
Correlation Identifier	Event Message	
Message Sequence	Request-Reply	
Message Expiration	Return Address	
Format Indicator	Correlation Identifier	
	Message Sequence	
	Message Expiration	
	Format Indicator	

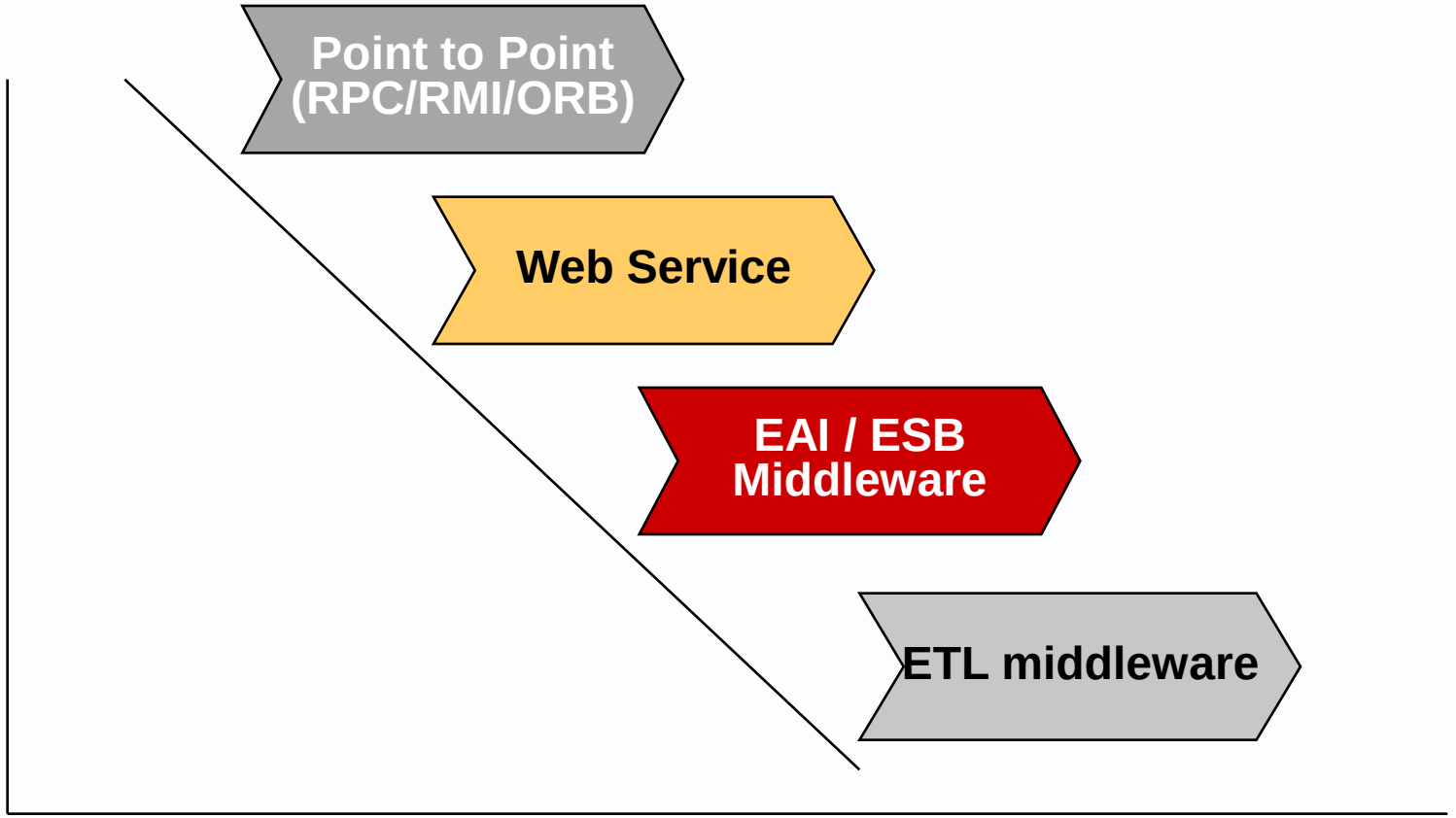
Design target applications architecture

1. Identify data flows, data stores and applications in scope
2. **Select best-fitting Application Integration Pattern**
3. Draw application communication diagram (aka DFD)
4. Draw sequence diagrams for key processes

Guidance on integration tool options from Tim Eyres

Avancier

- Message frequency



- Message size

Guidance on integration tool options from MIT

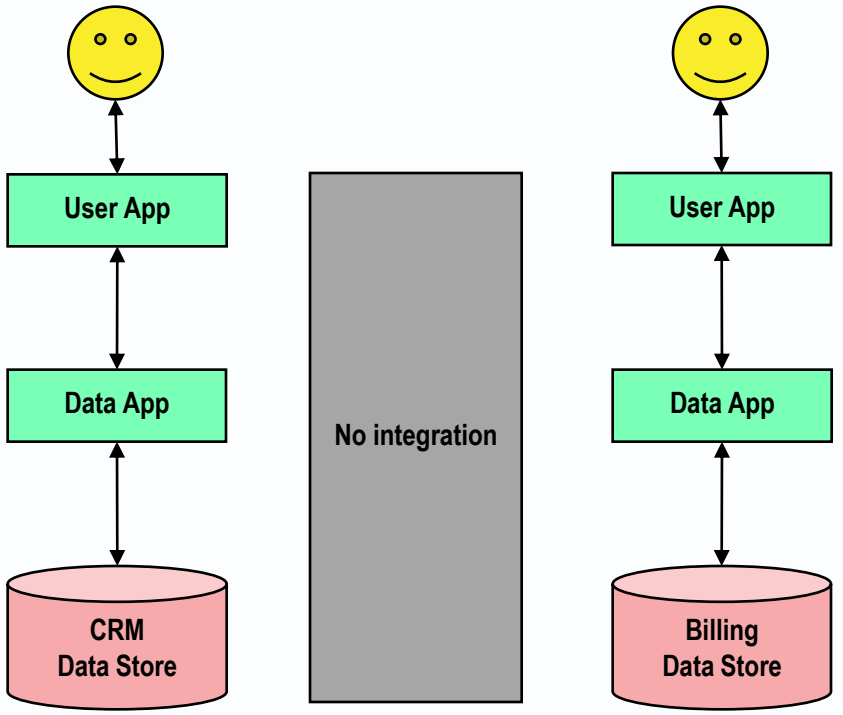
Avancier

	EAI/ESB Middleware	Point to Point (RPC/RMI/ORB)	Web Service	ETL middleware
Concept	• Publish/Subscribe mechanism • Most suitable for real time data needs • Loosely coupled	• Custom code for each integration need • Suitable for complex integration needs • Tightly coupled	• Standards based integration • Most suitable for inter-organization integration • Loosely coupled	• Suitable for large volumes of data • Generally used to move data between two or more databases
Strengths	• Reliability (guaranteed delivery) • Enables real-time business decisions • Out of box adapters for many enterprise systems	• Familiar technologies and processes • Many point to point integrations already exist • No major up front investment required	• Standards based integration • High degree of reuse • Wide tool support including open source • Low up front investment	• Metadata driven approach • GUI tools for most tasks (little coding) • Extremely efficient for large data volumes
Weaknesses	• High upfront cost • Relatively complex design patterns	• Costly over time • Tight coupling • Scalability issues • Opportunities for reuse are slim	• Lack of transaction support • Not a publishing model • Less established technology	• High upfront costs • Complexity of tool • Batch oriented
When to Use	• Real time data is important • High volume, low footprint data exchange • Many consumers of the same data	• Should be rarely used • When defined enterprise strategy cannot work • Proto typing	• Integration model is request/reply • Real time requirements • High volume, moderate data	• In conjunction with a data warehouse

Tool selection must follow pattern selection!

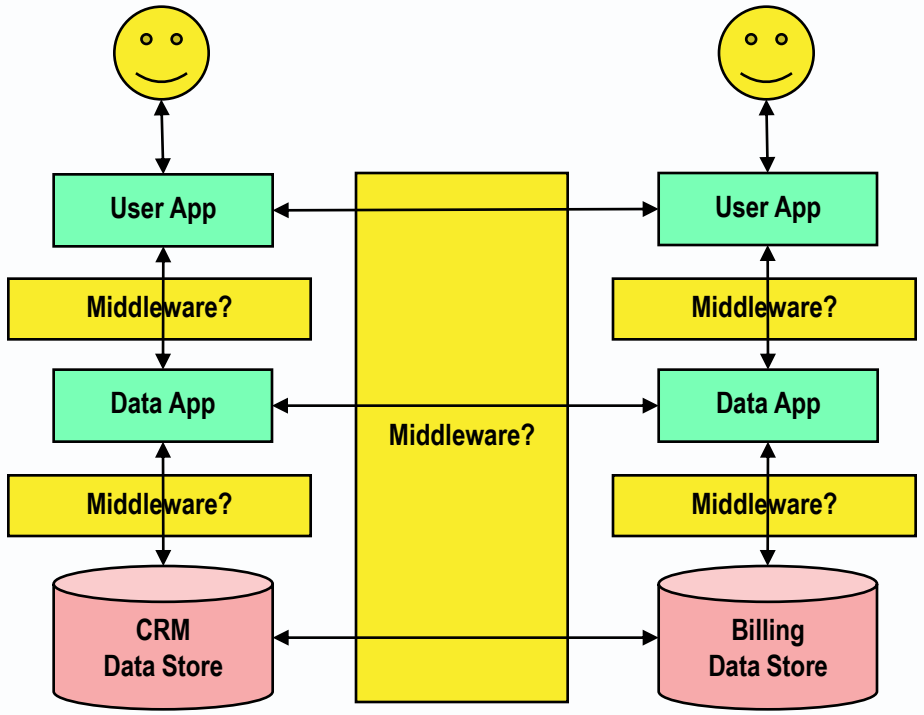
Silo apps in the baseline architecture

- Let us assume client-server layering
- User App components
  - client-side components that present data via a GUI window, an HTML page or a perhaps send it via a message queue
- Data App components (aka Business Components or MicroServices?)
  - server-side components that obtain and perhaps maintain business data



Where is messaging middleware best used?

- A matter of debate
- We're going to focus on integrating separate applications

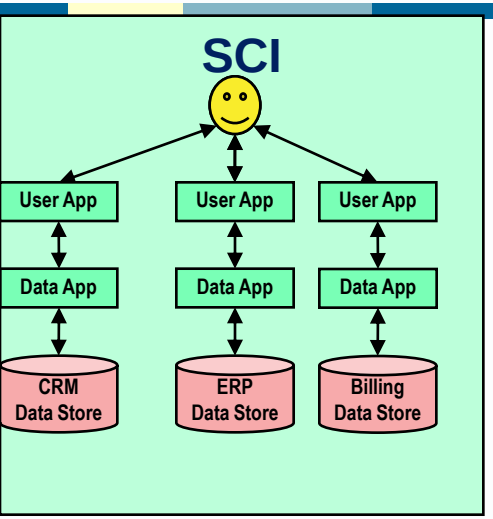


Applications Integration Patterns

- [a pattern] for sharing data currently stored in several enterprise business databases

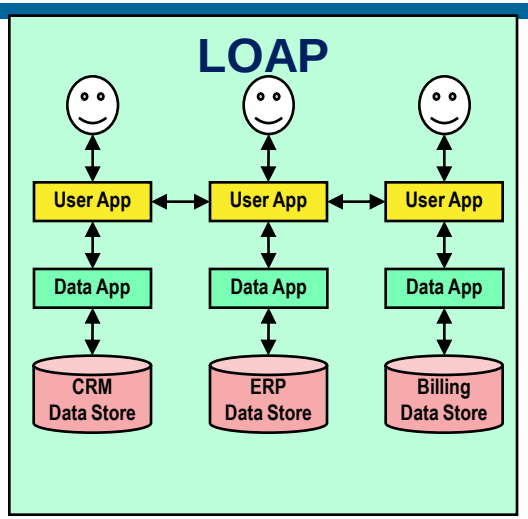
- Patterns to follow
  1. Swivel chair integration
  2. Lipstick on a pig
  3. Nosey neighbour
  4. Distributed transaction
  5. Run around (ETL?)
  6. Data warehouse (ETL?)
  7. Database/app consolidation
  8. Physical master data
  9. Virtual master data

Swivel Chair Integration



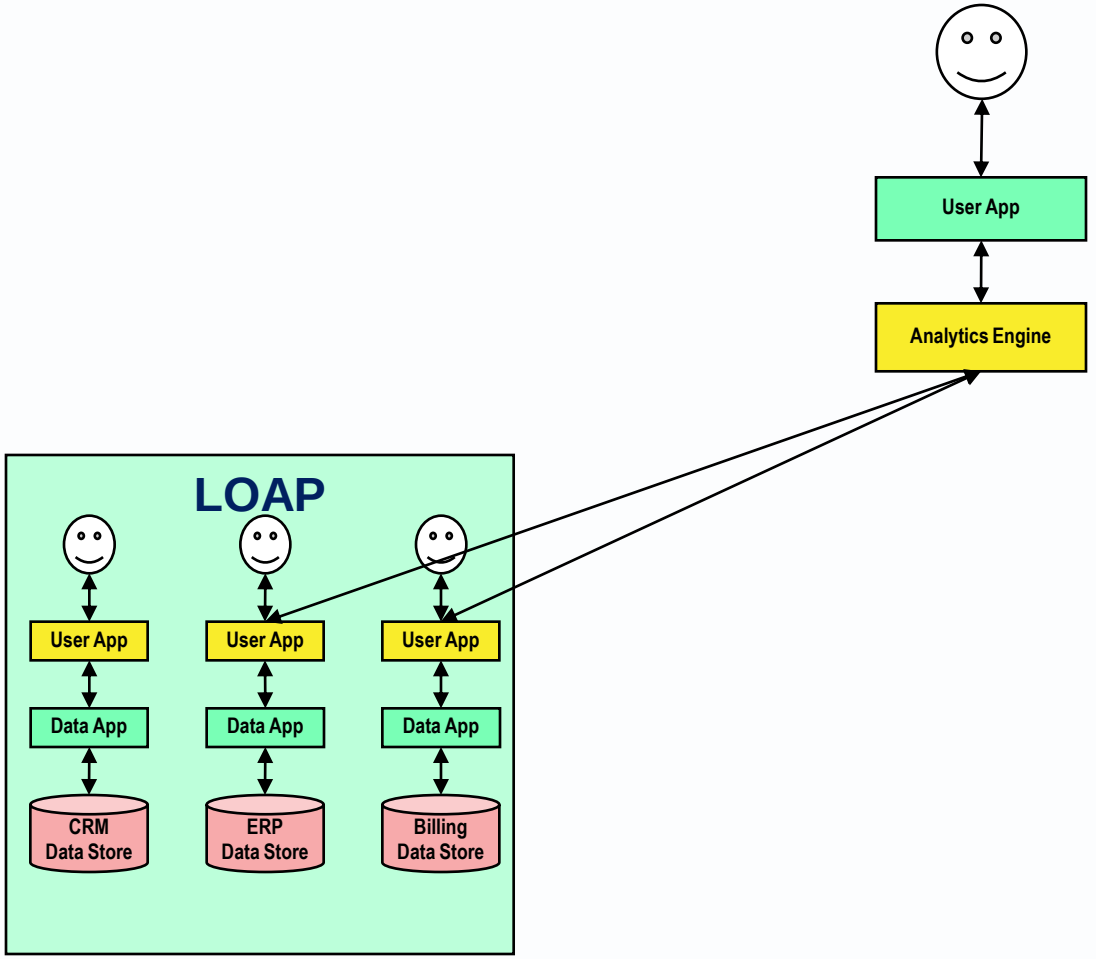
- Humans do the work
- Enter the same data into several applications

Lipstick on a pig – Robotic Process Automation (RPA)

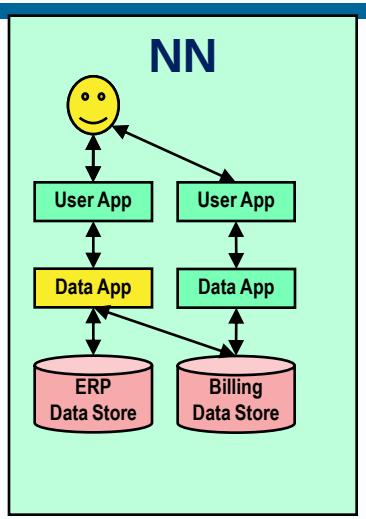


- Automated copy and paste between data entry screens
- (Variation, send data in email to somebody else for data entry)

Using RPA to do data analytics via the UI



Nosey neighbour

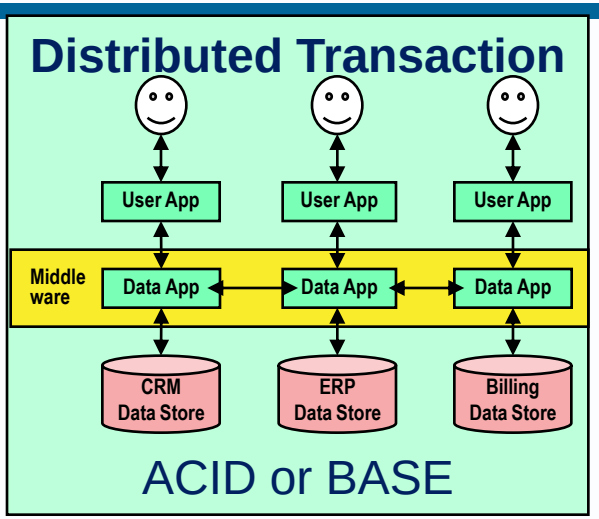


- Remember you are coupling the apps in terms of availability.

- Availability = availability * availability
- E.g. 98% = 99% * 99%

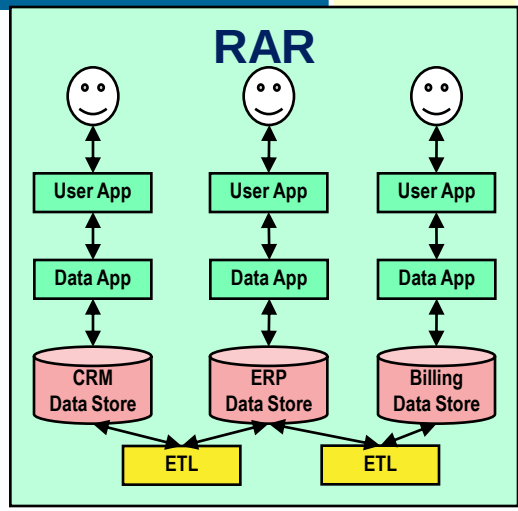
Distributed/Federated Transaction – On-line integration

- Concurrent transactions in two or more databases



- ACID – federated or distributed transaction
  - Consistency assured
  - Availability lower
- BASE – eventual consistency
  - Asynchronous updates
  - Compensation transactions needed!

Run around - Off-line integration



- [a pattern] in which discrete data stores are synchronised off-line,
- often by overnight batch processes, often using ETL tools.

- BASE – eventual consistency
  - Asynchronous updates
  - Compensation transactions needed!

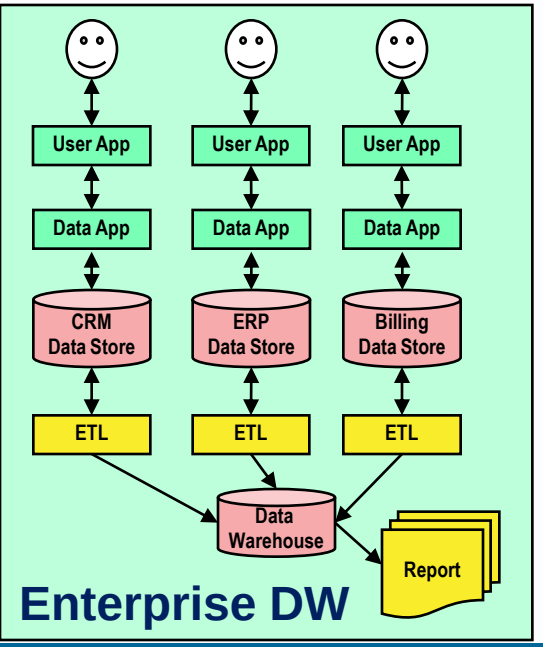
Change data capture (CDC) design patterns

- Run around integration may depend on on the identification, capture and delivery of data store changes

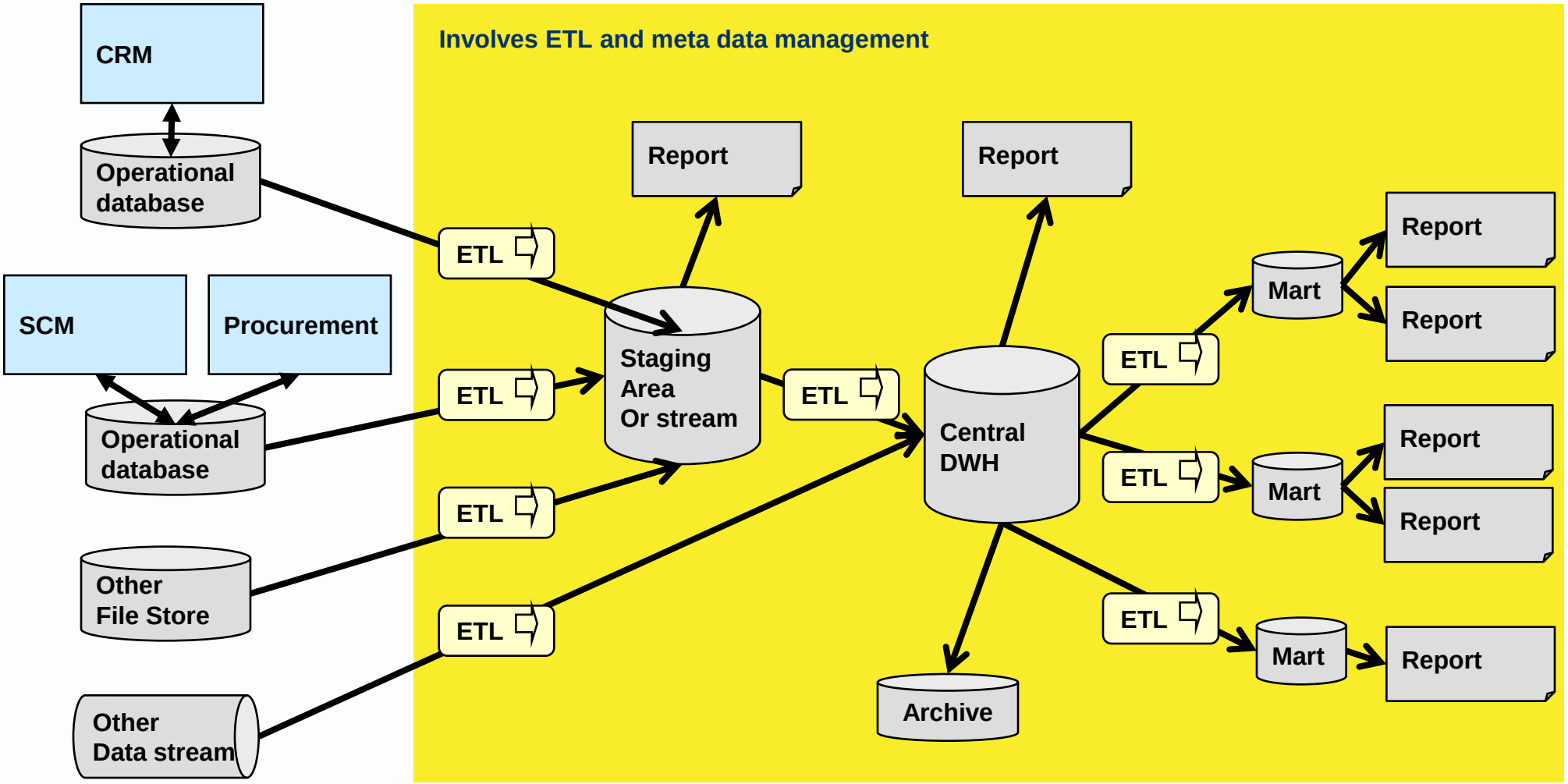
- **Mark changes** on rows/records
  - Version numbers, Timestamps and State variables
- **Find changes**
  - Scan database for marks on rows/records
    - “Capture all data for version 2.1 that
    - changed between 6/1/2005 12:00 a.m. and 7/1/2005 12:00 a.m.
    - where the state variable = ready for delivery."
  - Scan transaction logs (non-intrusive, but not easy)
- **Act on changes** (say, publish events)
  - Triggers on tables
  - Triggers from log scanning

Data warehousing

- [a pattern] in which business data is copied from on-line data stores into a central database for reporting, often using ETL tools.
- Data cleansing may be needed at any stage in the process.



ETL pattern for Data Warehouse



To succeed with ETL and DW, it helps to understand

- Outputs required
  - The data that is needed for business intelligence
  - The best source for that data
- How to structure the stored data
  - Stars, snowflakes and dimensions
- How to handle slowly changing dimensions (SCD)
  - E.g. Customer moves from one sales area to another
- How to mark, find and act on data changes (CDC)
- How to gather, conform, cleanse and transform data
- How to implement error handling and conditional processing.

Data warehouse success story

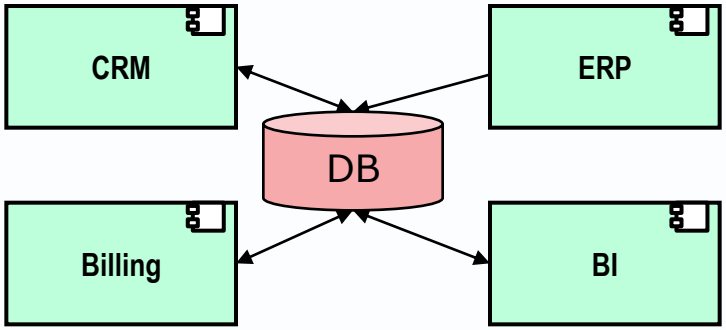
Avancier

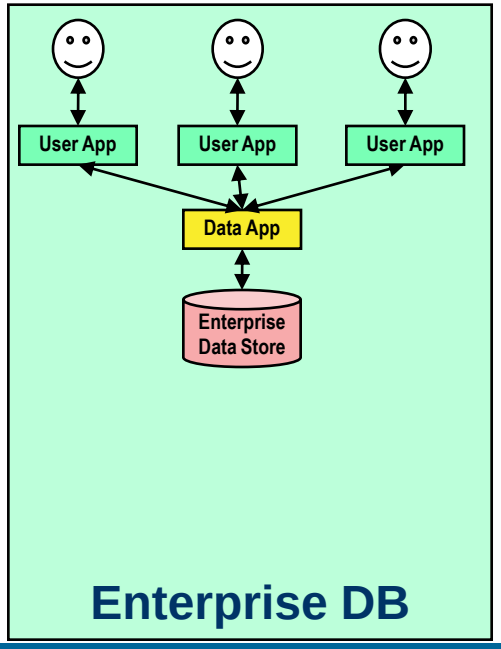
- **A 'Single Customer View' (SCV) project (**Information Builders**)**
- Implemented across Europe for a global retail banking group.
  - 7 banks
  - 20 legacy data sources
  - 50 million customers
  - 130 million accounts.
- This complex project:
  - was completed in record time, 10 months after kickoff.
  - a Master Data Management and Management Information Reporting solution
- Initiated in response to the UK Financial Services Authority's new rules for the Financial Services Compensation Scheme,
- the project included:
  - management and monitoring of file feeds from numerous data sources;
  - cleansing and standardising against multiple internal and external reference sources
  - matching and merging to create the SCV.

Database/app consolidation

- [a pattern] in which baseline applications become user application components accessing one shared database.

- ACID – transaction
  - Consistency assured





Physical master data

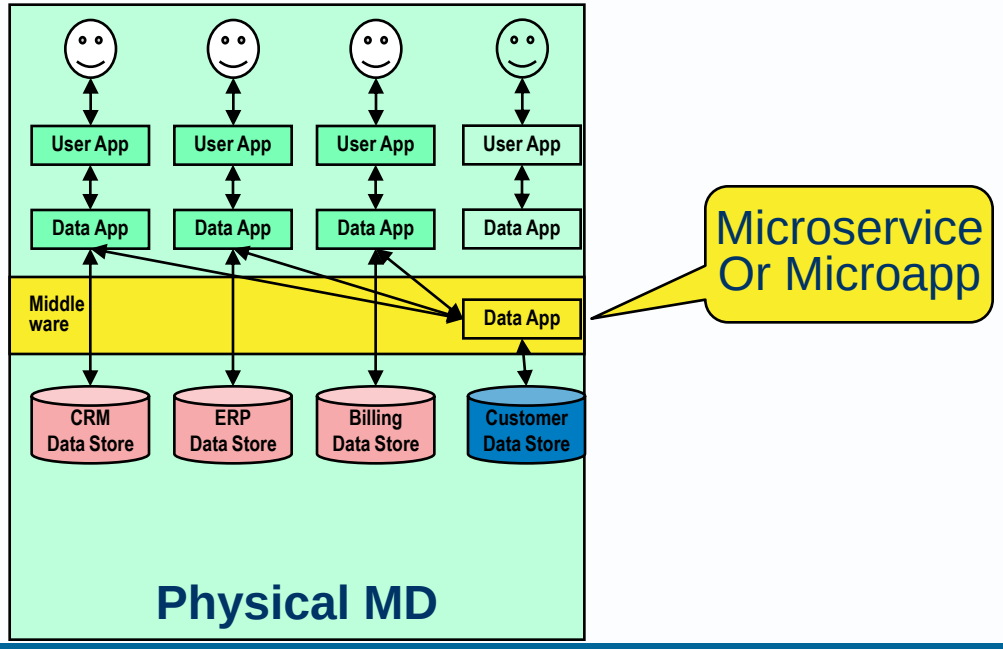
- [a pattern] in which a common data entity is stored in a discrete database, where it can be accessed by any application with a pointer to the common data.

Commonly duplicated data

- Customer?
- Employee?
- Person?
- Product?
- Asset?

Options

- Leave only pointers to new data
- Maintain copies



Virtual Master Data

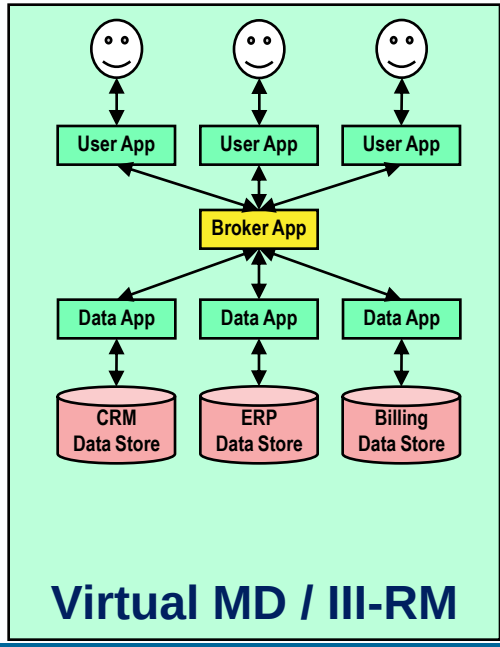
Avancier

- [a pattern] in which required data can be integrated at run time from several data stores or sources by some kind of broker application. It features three layers of software components.

- **User apps:** present user interfaces, capture events from them and invoke broker apps.

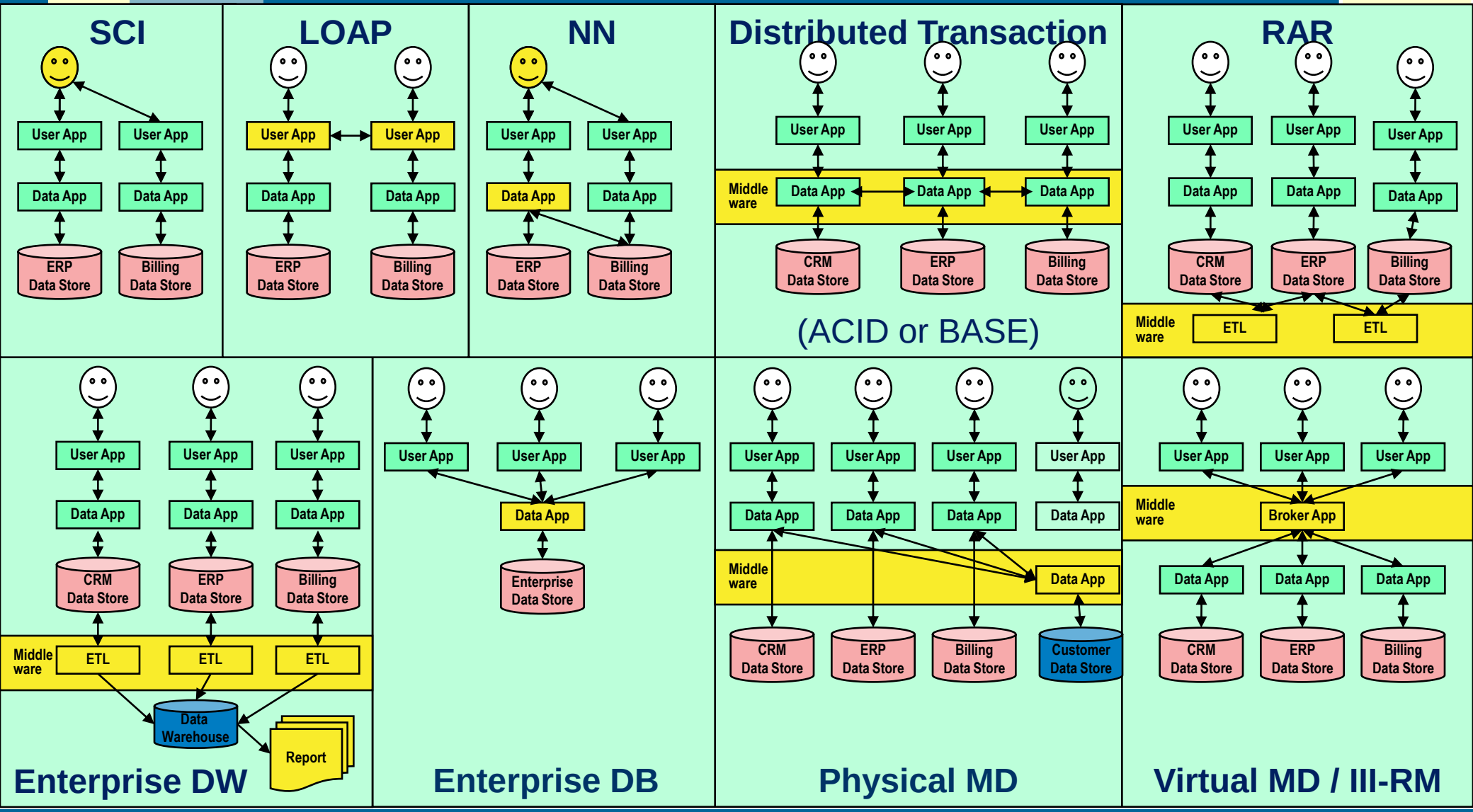
- **Broker apps:** decouple by providing automated business services to user apps, and invoking data services from data app(s)

- **Data apps:** provide automated data services to put/get data to/from a particular database or other data source.



Application Integration Patterns

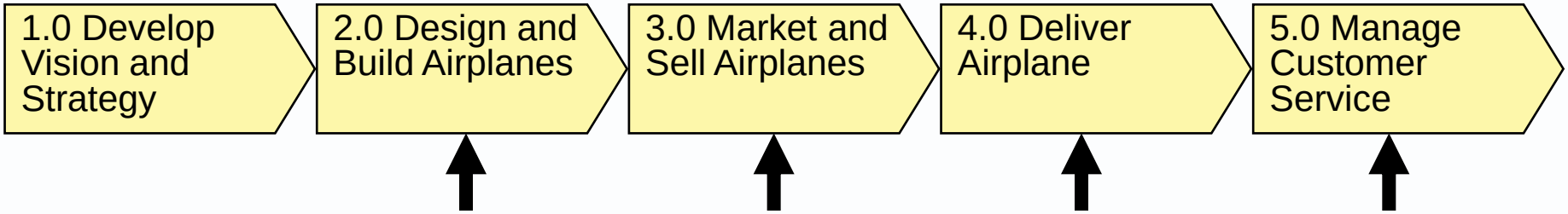
Avancier



So which option suits this story?

Database consolidation?
Physical master data management?
Virtual master data management?

Avancier



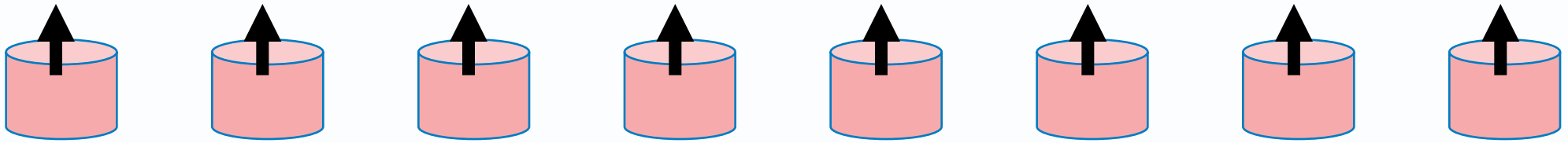
100 of 500 major user apps need data about an airplane

Name: Give me airplane description

Input: Airplane identifier

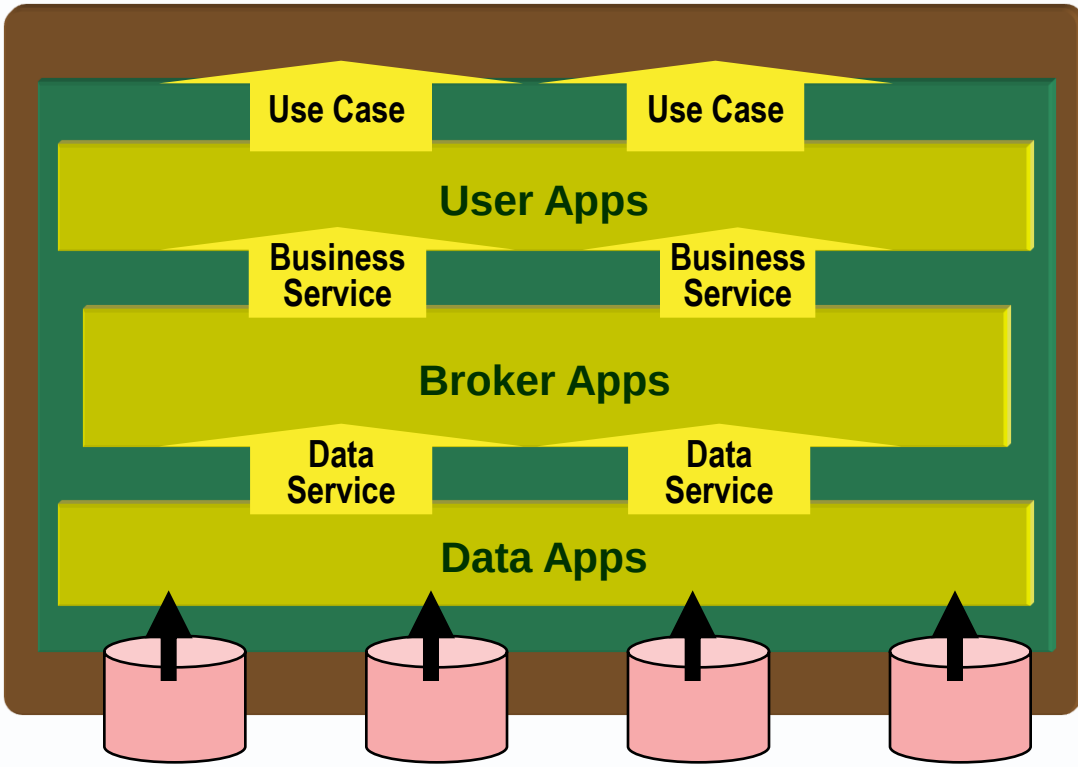
Output: Airplane identifier, Model, Version, Length, Fuel capacity, Wing span, Wing area, Flying weight, Wing loading, Thrust, Engine supplier

100 of 500 major databases contain data about an airplane



App/IS Services in the BCS reference model

- **Use Cases**
  - Uses made by users
- **Automated “business” services**
  - Automated IS services that are invoked using data types in a canonical data model
- **Automated data services**
  - Automated IS services that need to understand data types in a local data sources

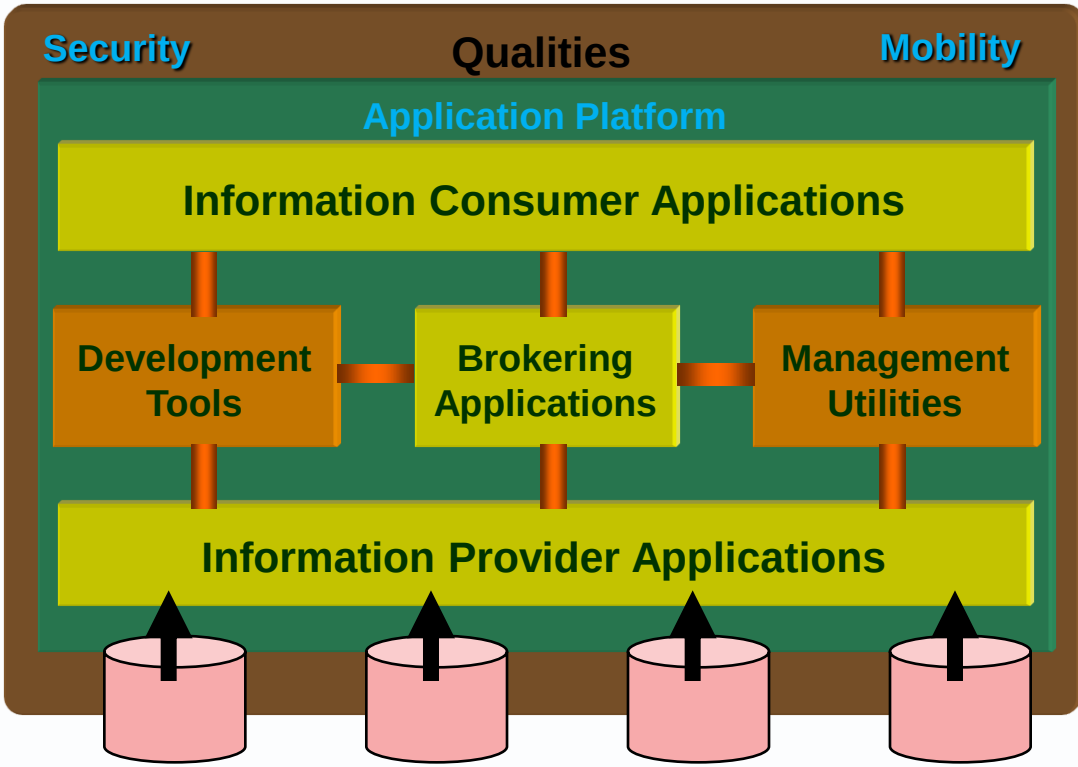


The III-RM in TOGAF (Integrated Information Infrastructure RM)

- **Information Consumer Applications**
  - deliver content to the user of the system,
  - provide services to request access to information in the system on the user's behalf
- **Brokering Applications**
  - manage the requests from any number of clients
  - to and across any number of Information Provider Applications
- **Information Provider Applications**
  - provide responses to client requests
  - and rudimentary access
  - to data managed by a particular server

- The overall set creates an environment that provides a rich set of end-user services for transparently accessing heterogeneous systems, databases, and file systems.

- **TOGAF v9**



Themes of Avancier's architect training

- 1) Think about the business context
- 2) Don't forget the numbers
- 3) You have to balance trade offs – between ways to build something.

How to choose between app integration patterns?

- There are always trade offs
- Evaluate options against the criteria that matter to you

Quality goals	
Confidentiality	Low
Integrity	Medium
Availability	Medium
Change requirements	
Budget	Low
Deadline	High
Resources needed	Low

Where are the business rules

- Fowler promotes “smart end points, dumb pipes”

- Meaning?
- Don’t put business rules in middleware or messaging tools
- Except data format transformation rules

- (Otherwise the middleware becomes an application
- And rules become separated from master data sources)

- Important rules are often best placed with or near the master data store or source

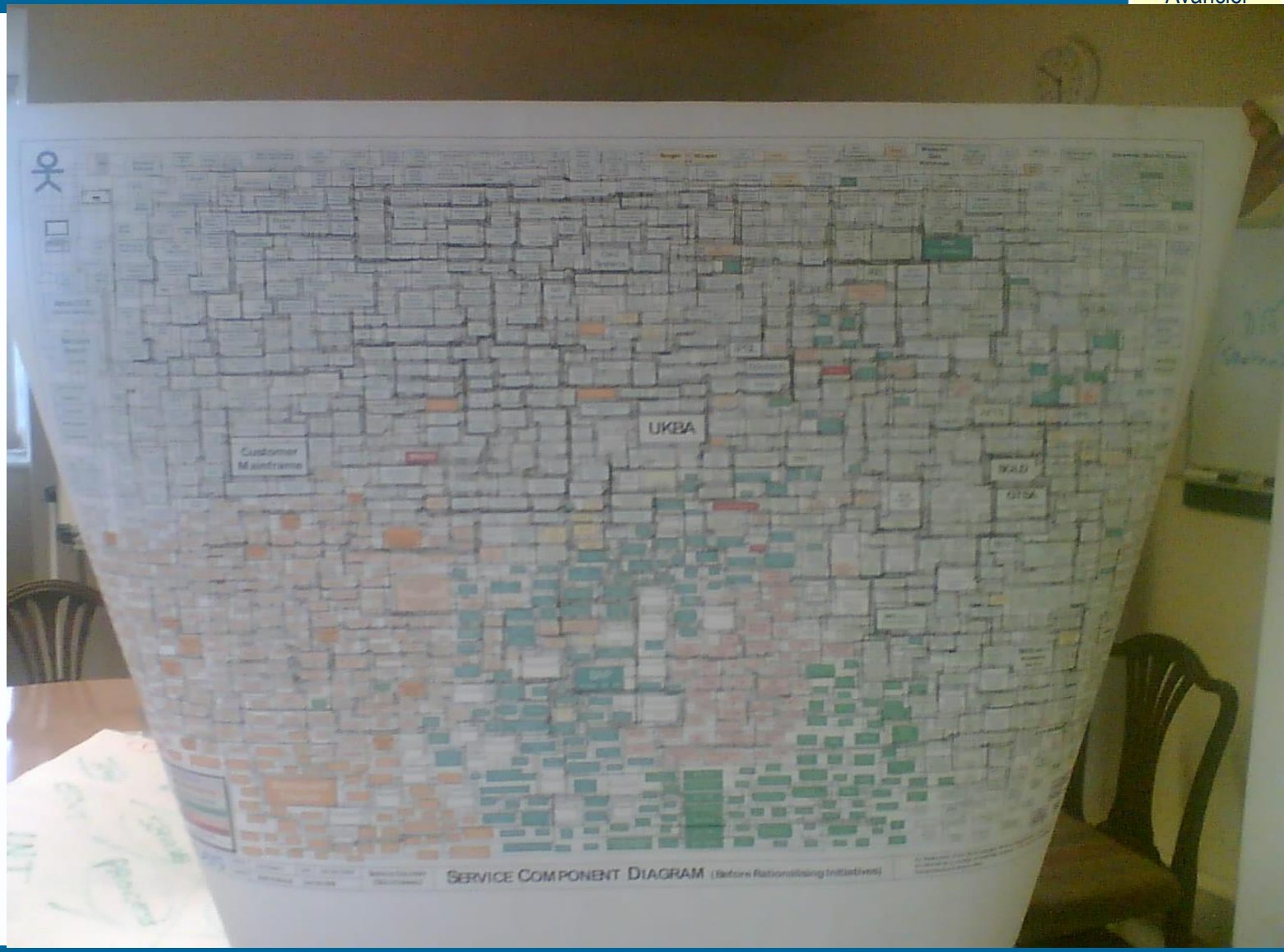
Design target applications architecture

Avancier

1. Scope application changes
2. Identify data flows, data stores and applications in scope
3. Select best-fitting Application Integration Pattern
4. **Draw application communication diagram**
5. Draw sequence diagrams for key processes

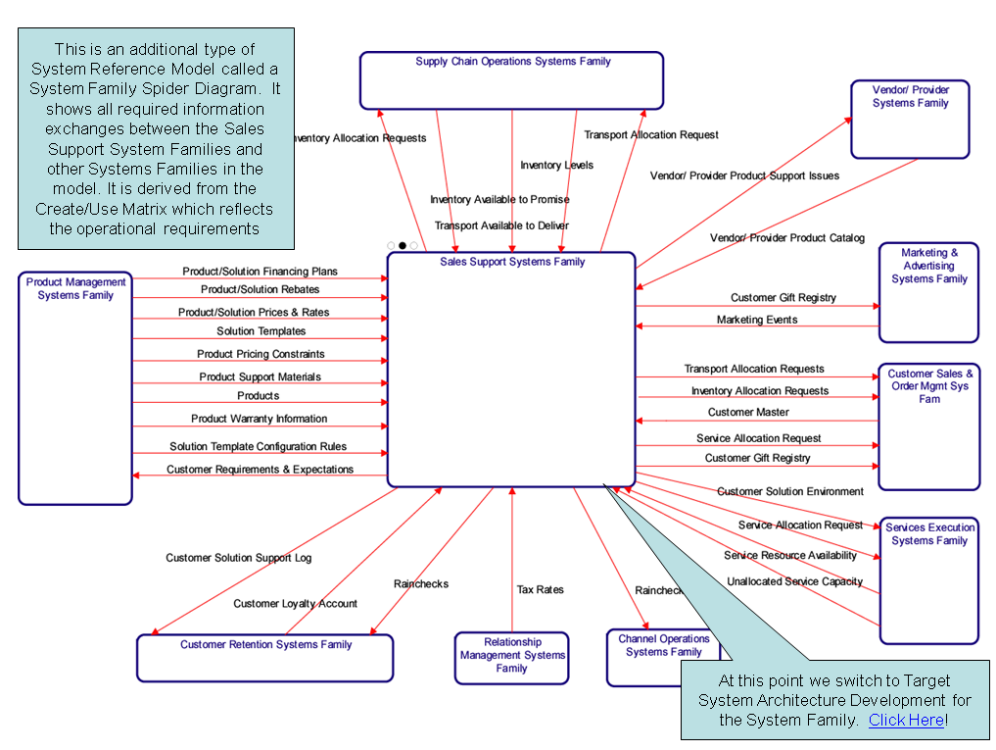
Top-level shock and awe diagram

- How to simplify it?



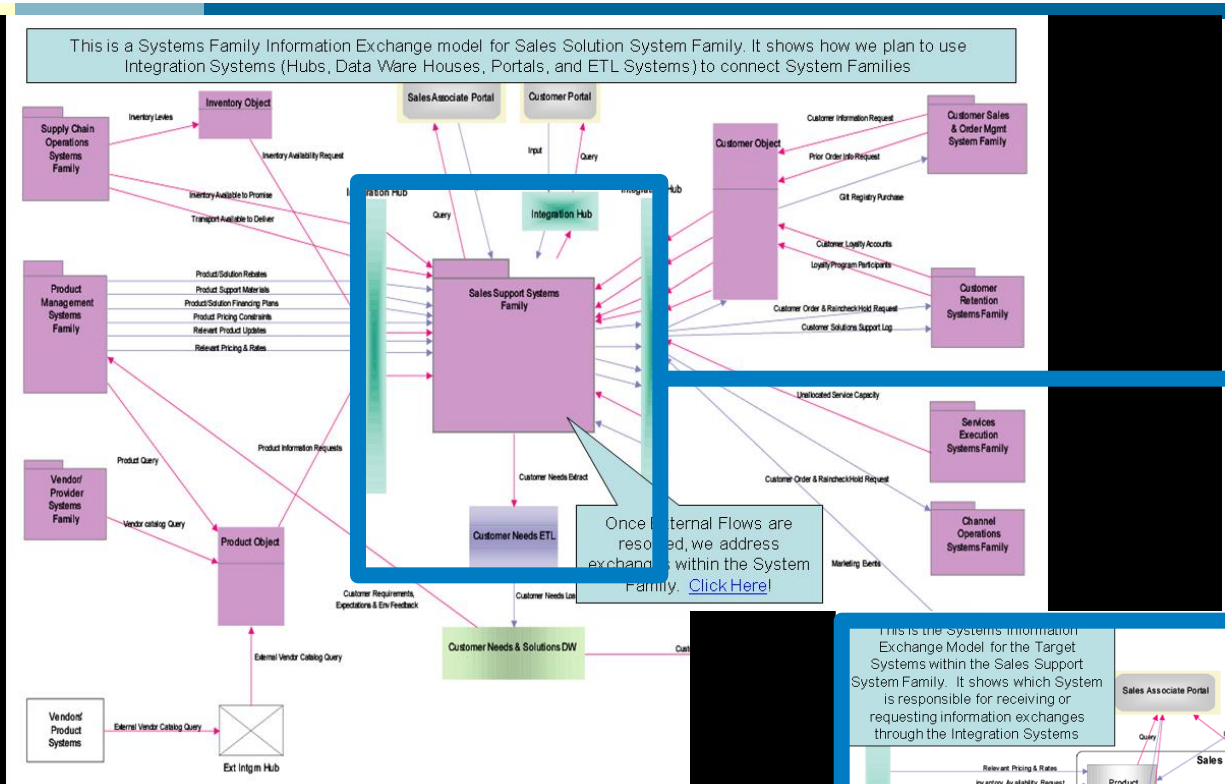
A possible diagram simplification strategy

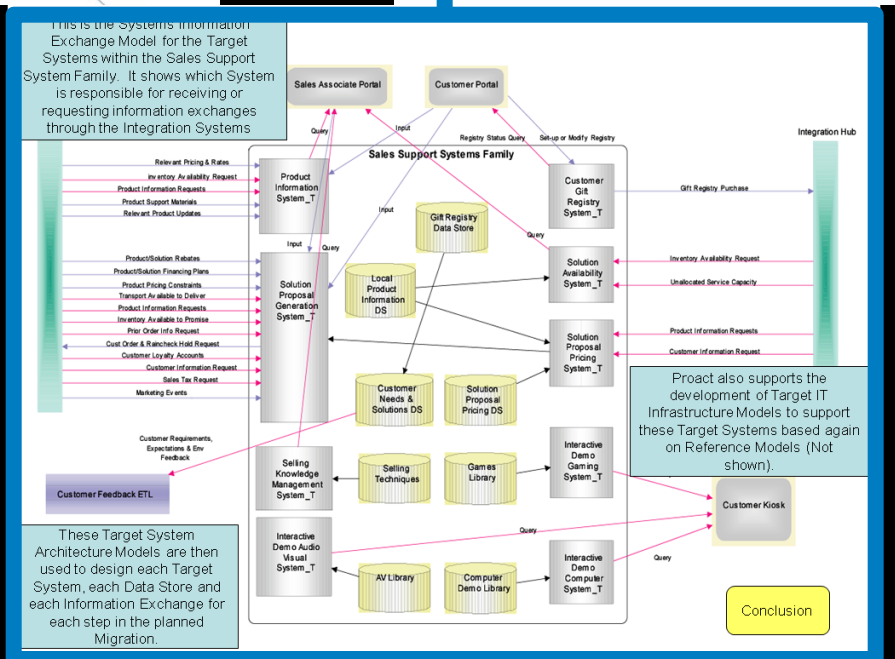
- Group cohesive applications into system families
- Draw 1 higher level diagram showing N system families



- And N lower level diagrams, each for 1 system family

A lower level diagram, showing apps within 1 system family

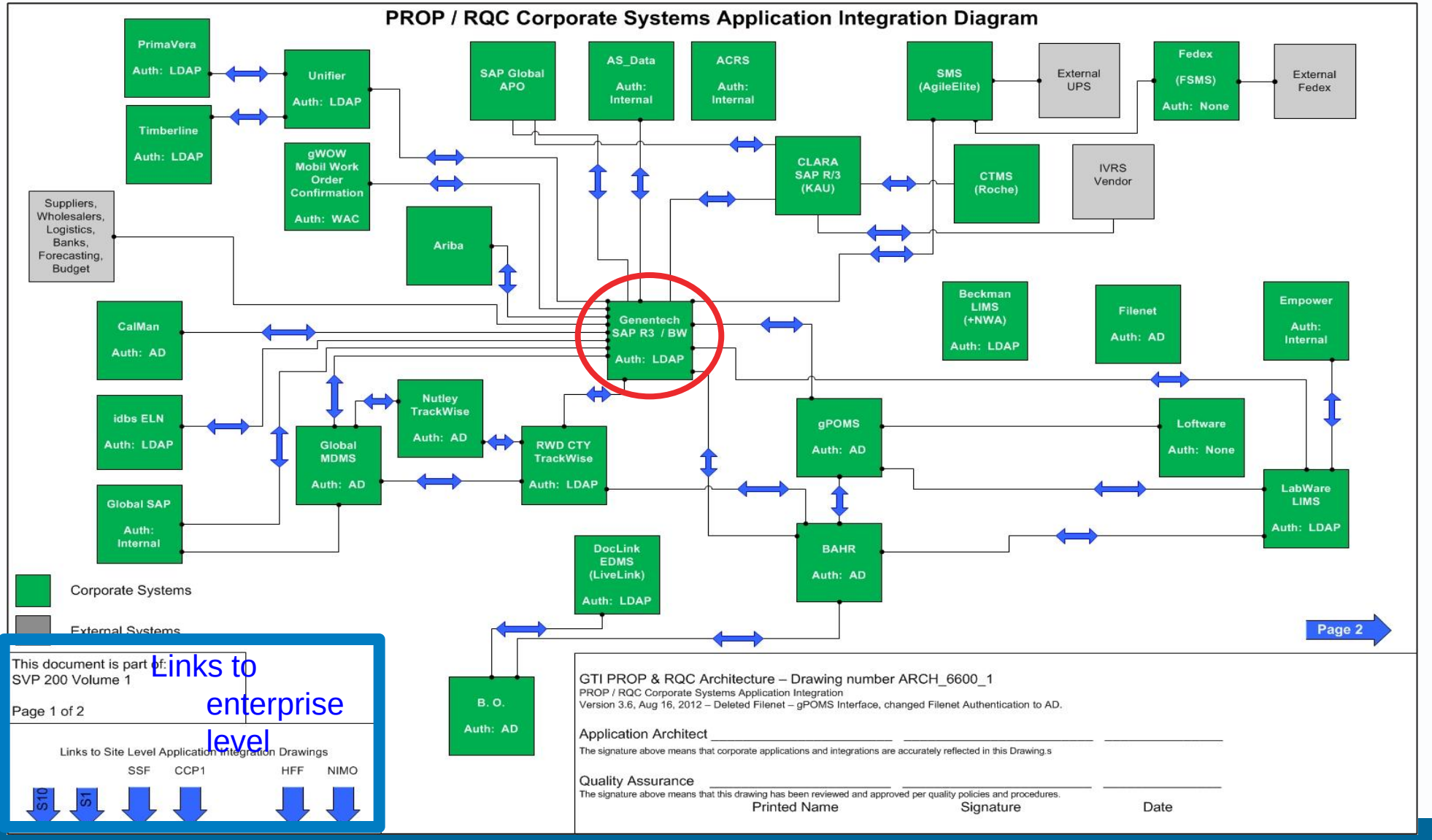




- Notice the middleware seems to be the source and destination of the data

One system family – centered on SAP R3

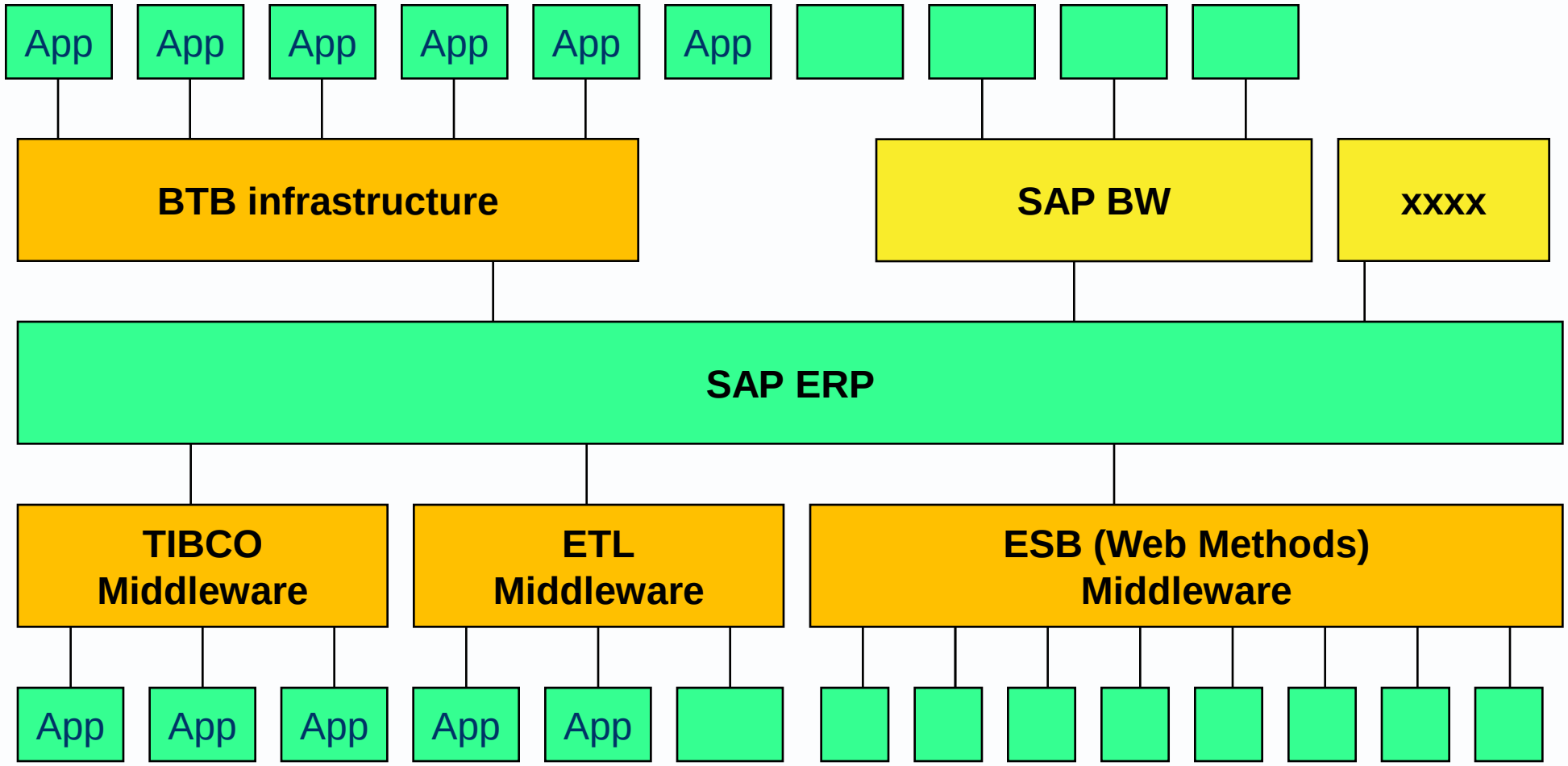
Avancier



Same system family – centered on SAP R3 – showing middleware

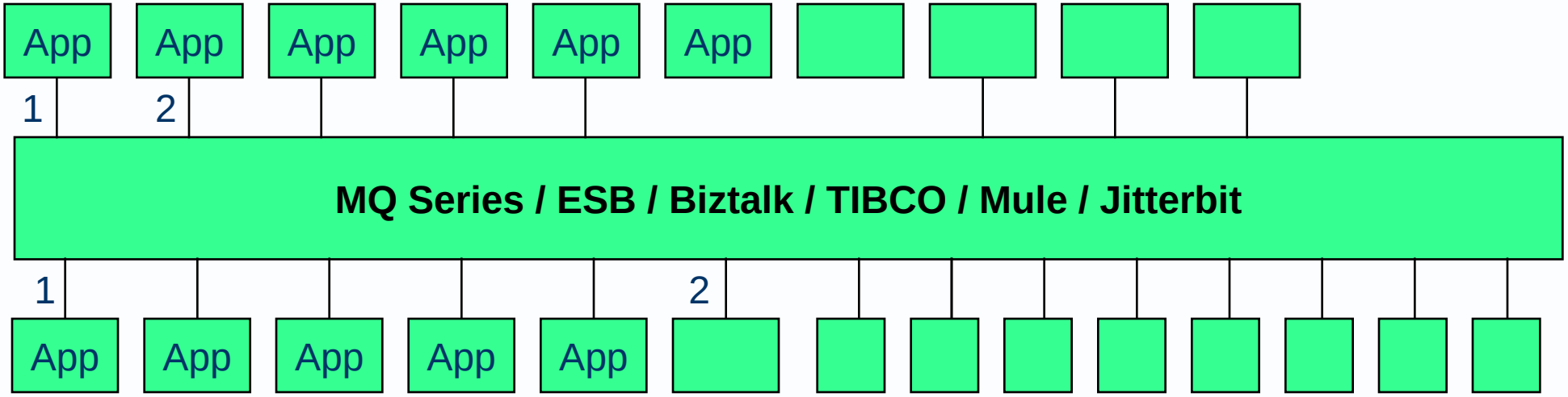
Avancier

- Shows middleware usage - *not specific data flows*



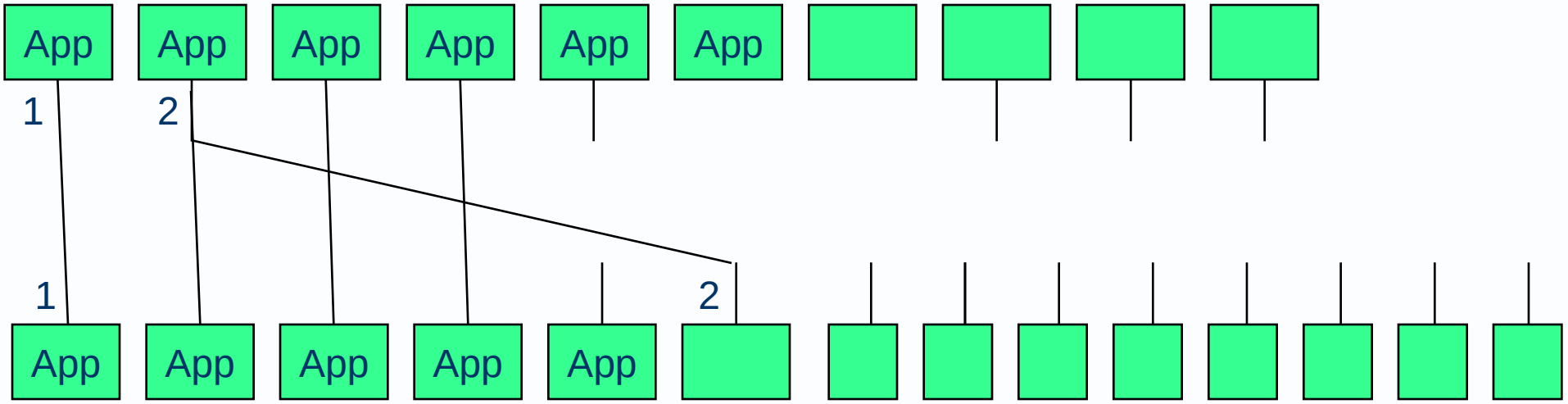
A different customer

- The real example had about 50 apps top and bottom and c 100 data flows

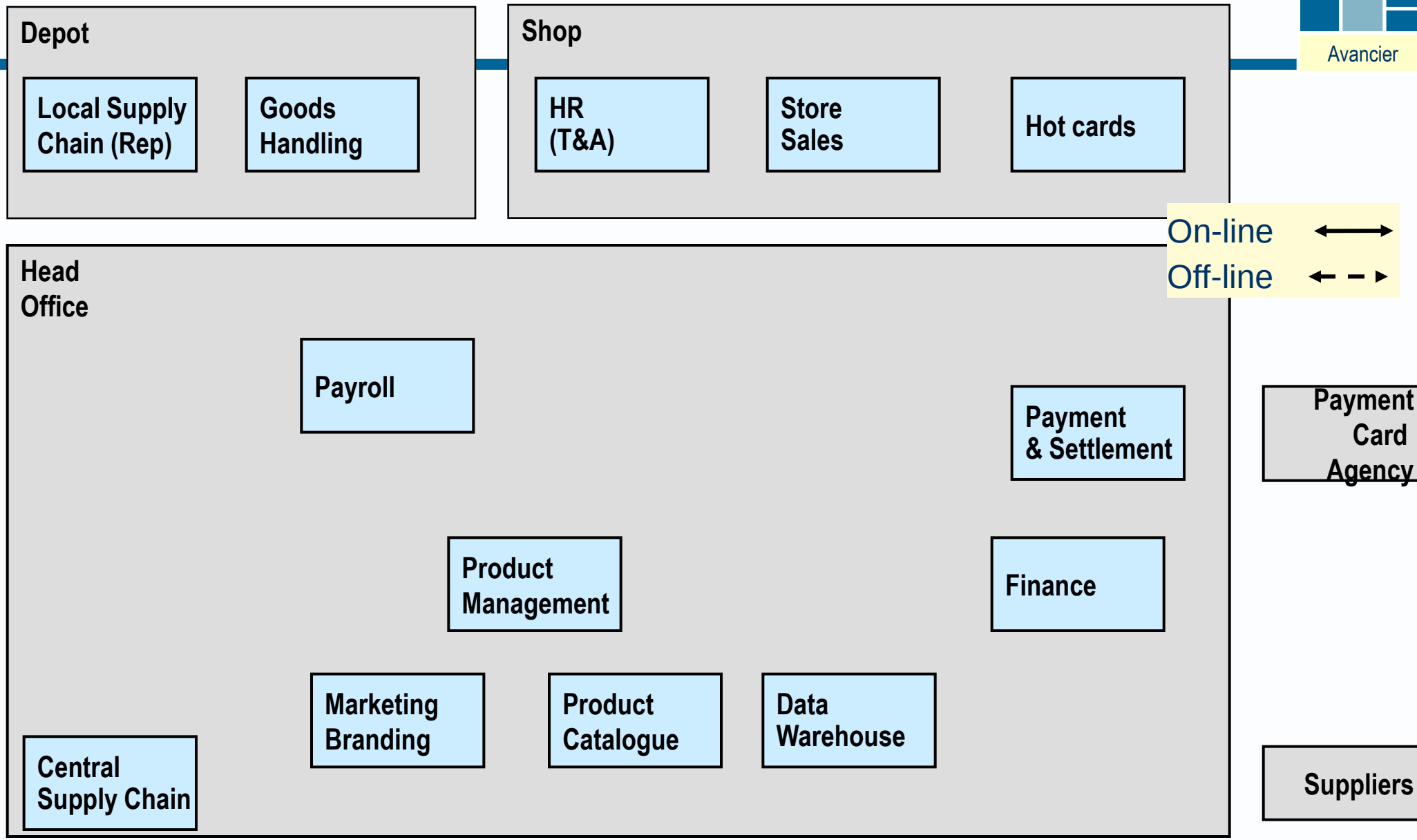


A different customer

- The real example had about 50 apps top and bottom and c 100 data flows



Q) Draw known data flows (note esp. 3.2 and 3.4)



FOOTNOTES

Physical decoupling not = logical decoupling

- Architecture principles often include **decoupling**
- It turns out to be a complex and multi-faceted idea

- Two application components may be **physically decoupled**
- Yet still be **logically coupled** by the need for data consistency

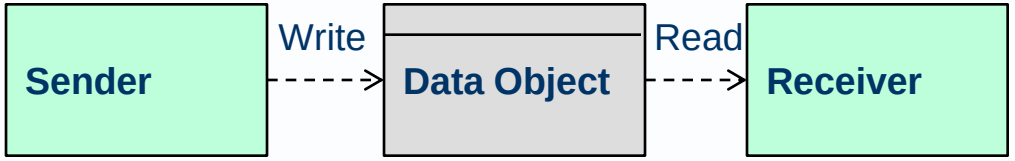
- Somebody has to
  - know where data comes from and where it goes to
  - understand how important data consistency is *in this context*
  - understand the business impacts caused by temporary inconsistency
  - design compensating transactions the business will accept

1 to 1 data flows in ArchiMate

Avancier

- Data flow as *relationship line*

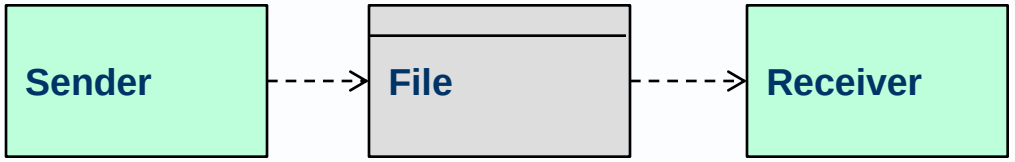
- Data flow as *data object between access arrows*



Using different notations to indicate different data flow kinds

- Use data flow line for a discrete message?

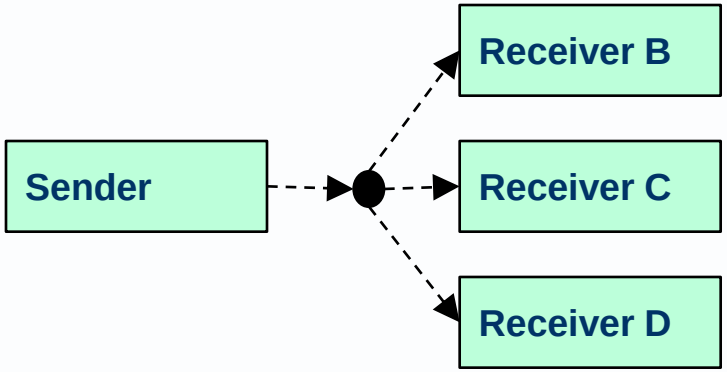
- Use access>data object>access – for a file or message batch?



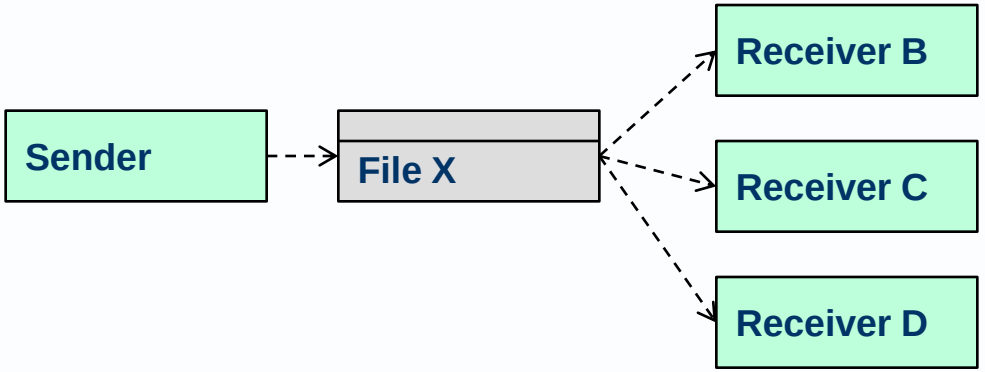
- Much communication is 1 to 1, but what about 1 to N?

1-to-N and N-to-N data flows

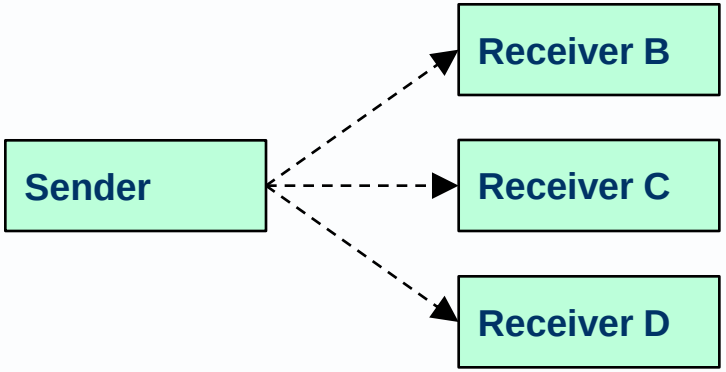
Sending 1 message to N recipients



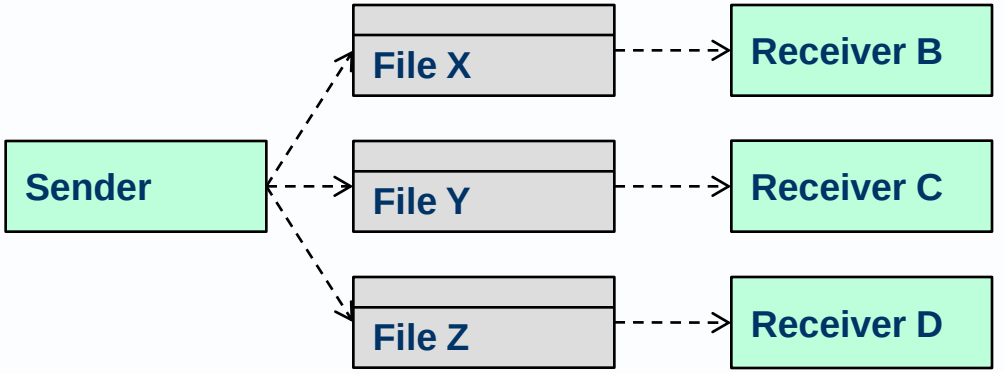
Sending 1 file to N recipients



Sending N message to N recipients



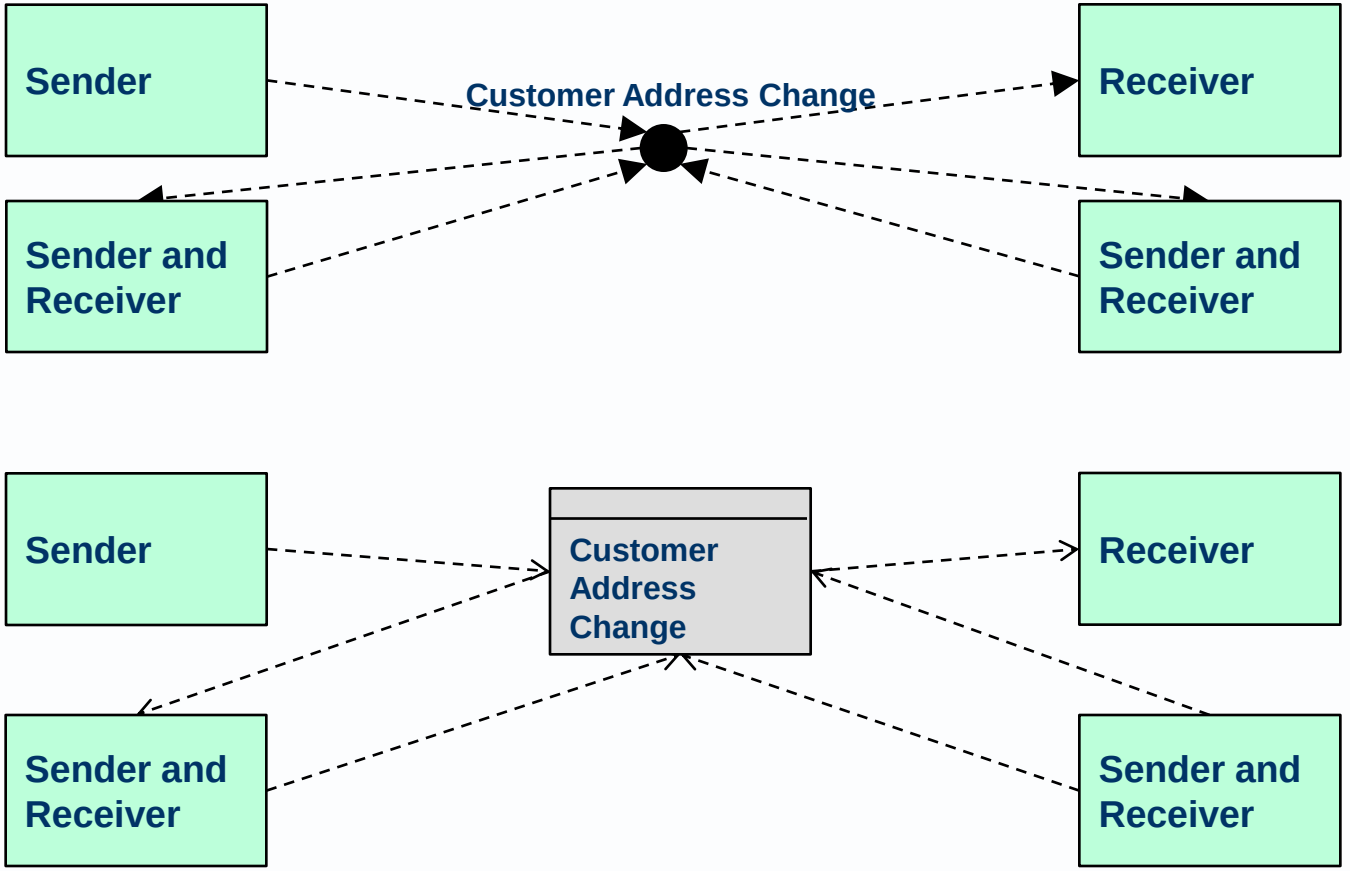
Sending N files to N recipients



Representing an N to N data flow

Avancier

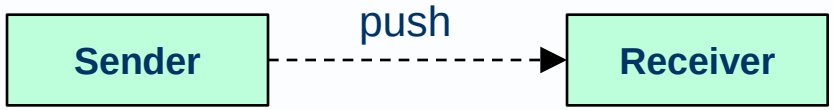
Possibilities include



Patterns for data interchange using middleware

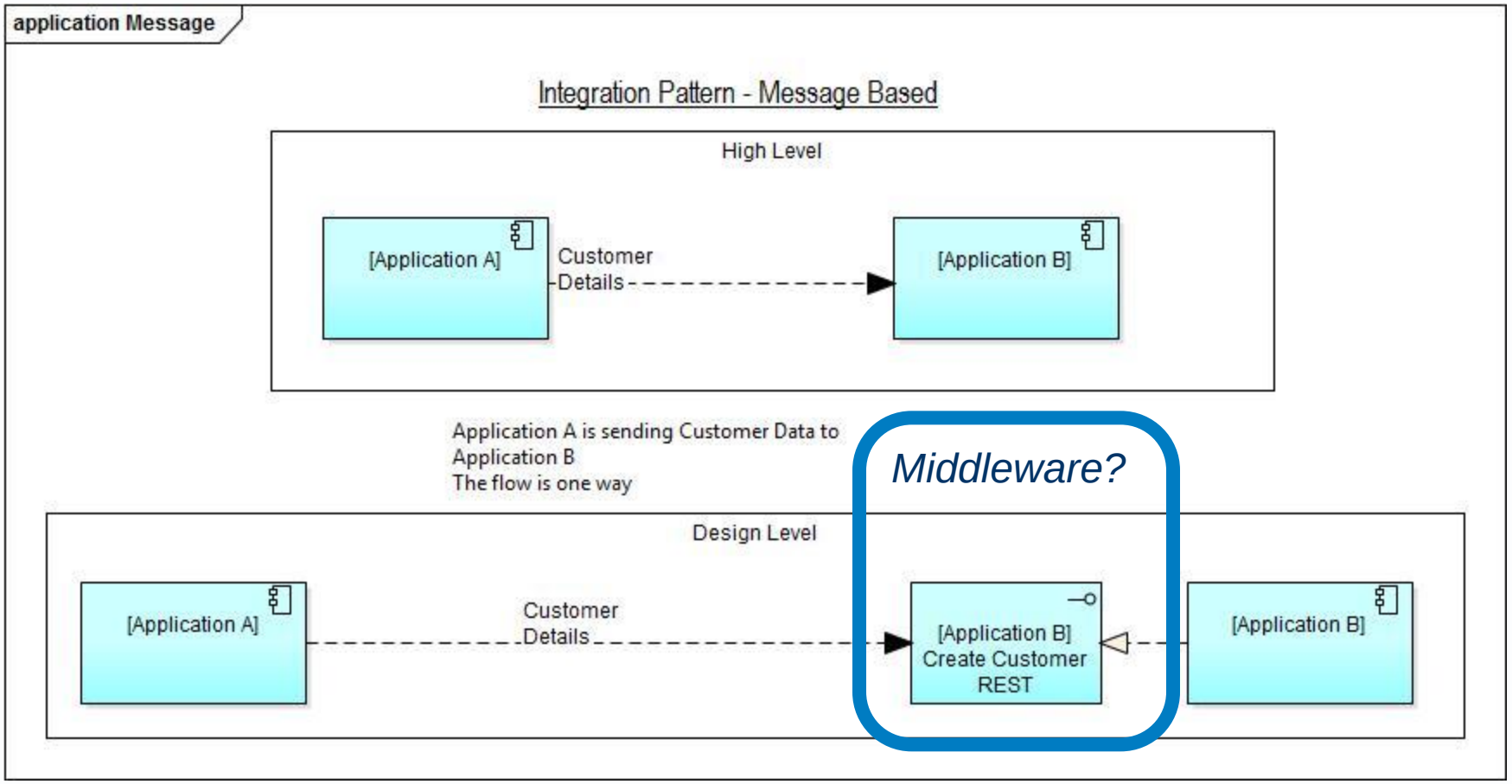
Push and pull data flow triggers

- Use data flow for push (fire-and-forget) message passing?

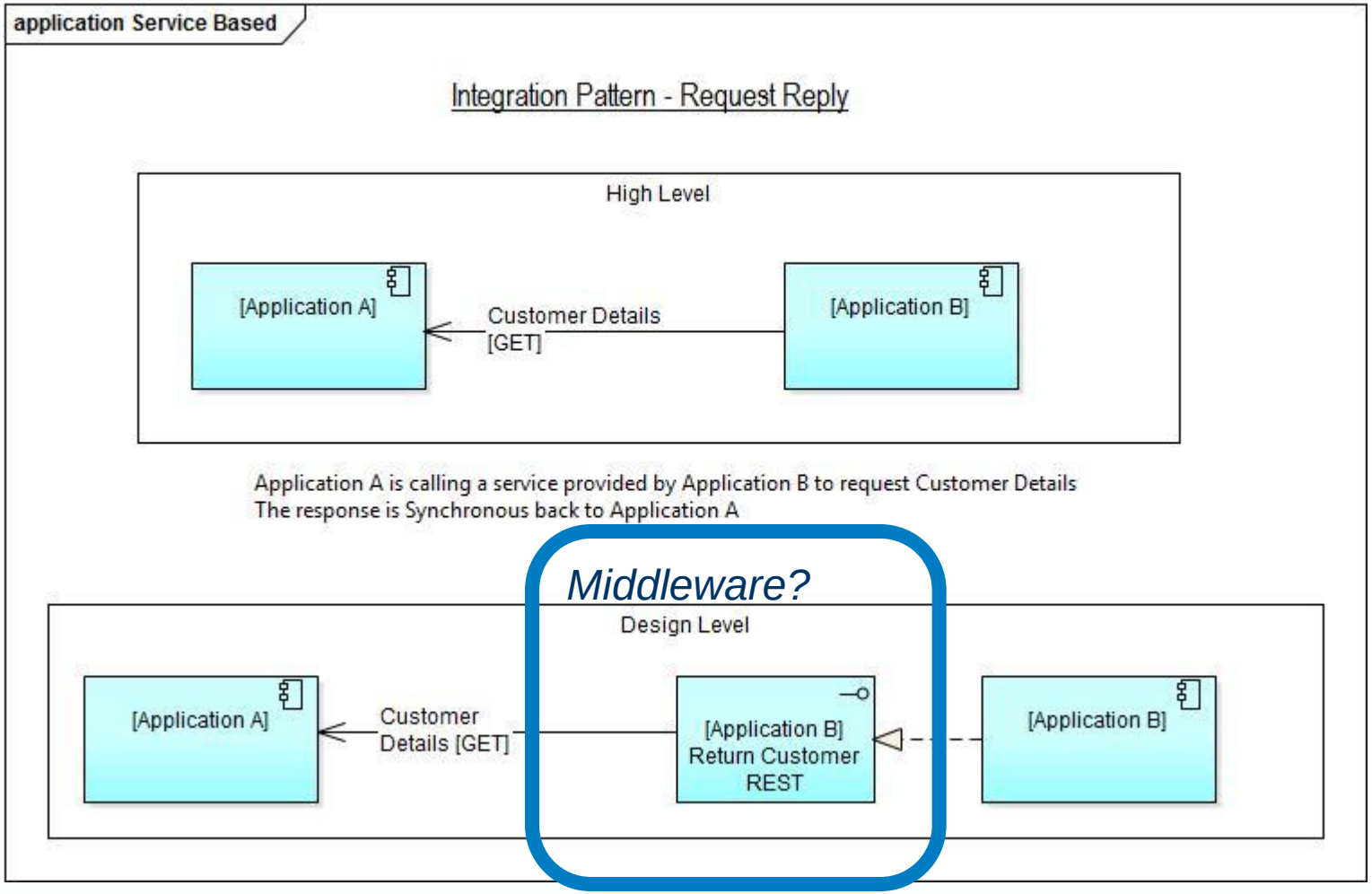


- Use serves for pull (request-reply) message passing?
  - (implies trigger from client to server)

Push - using middleware



Pull - using middleware

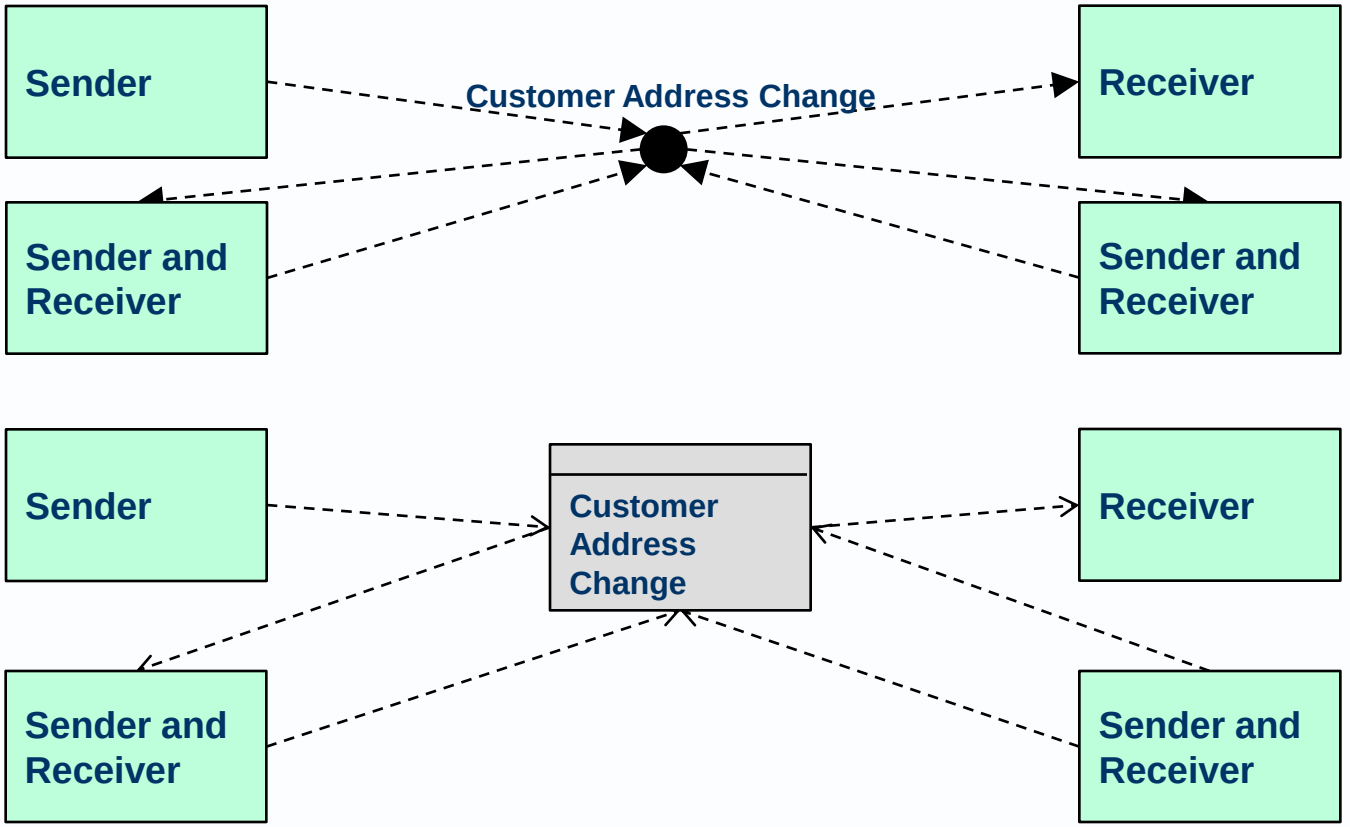


Event-Driven Architecture (N to N Data flows)

Avancier

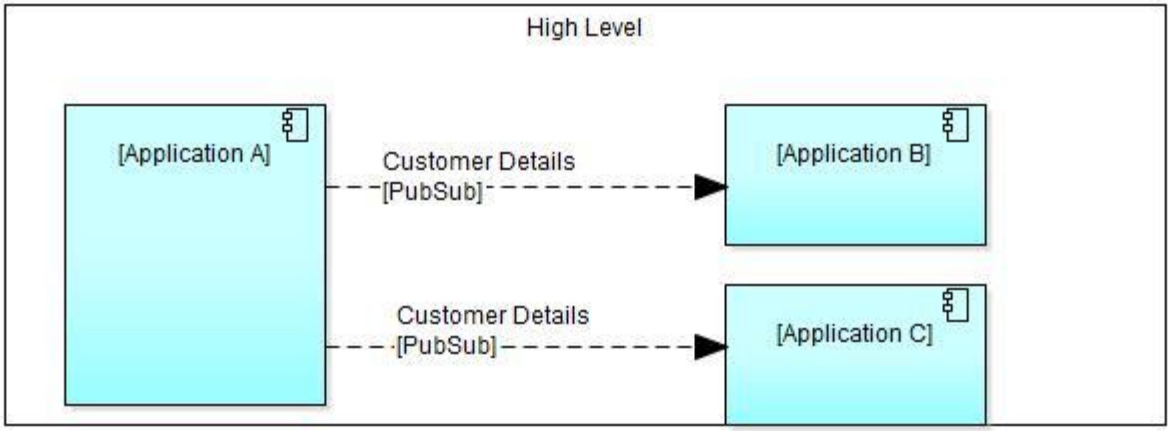
Senders publish events

Receivers subscribe to receive events



application Event-Driven Messaging - Push Model

Integration Pattern - Event Based - Push Model



Application A is Publishing Customer Details
Application B and C are Subscribers to the Customer Details
The data is being Pulled by Application B and Application C from Application A