Software architecture as a process

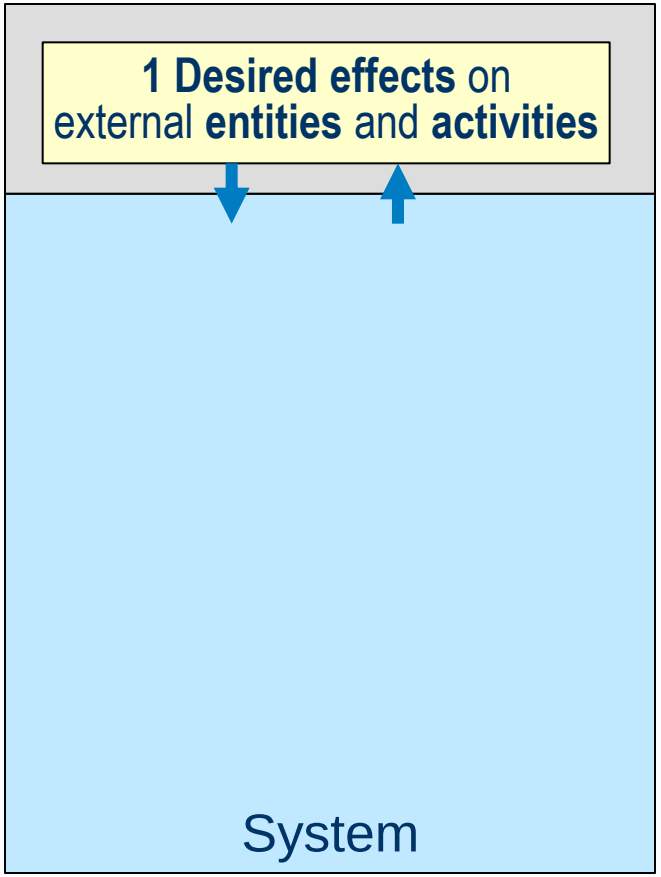
After General System Theory

A few messages for an architect

1. KISS
2. There is no single way to design systems
3. There are always trade offs
4. Vendors and fashion may push one design pattern, but an architect must be mindful of the opposite
5. Don’t overcomplicate the design to meet fanciful throughput and concurrency numbers that stakeholders agree are very unlikely in the next 5 years (YAGNI and the money will follow)
6. Beware that the design of data processing systems (e.g. division of data between data stores) has an impact on the business processes in the wider human activity system.
7. An architect must understand design patterns, trade offs and their impacts on the wider business system.

A system theory-based process

1. Define the desired effects or outcomes of a system



1. Define the desired effects or outcomes of a system

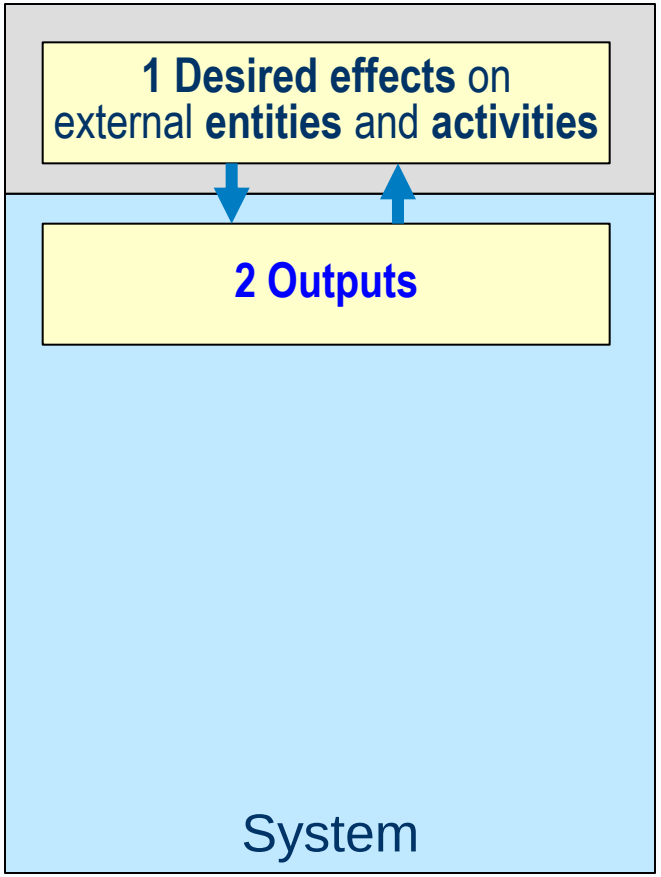
- Sponsors and domain experts should be able to advise as to

- the desired effects
- the required outcomes
- the purposes of the system
- how the entities and activities should be
  - enabled or supported
  - monitored or directed

A system theory-based process

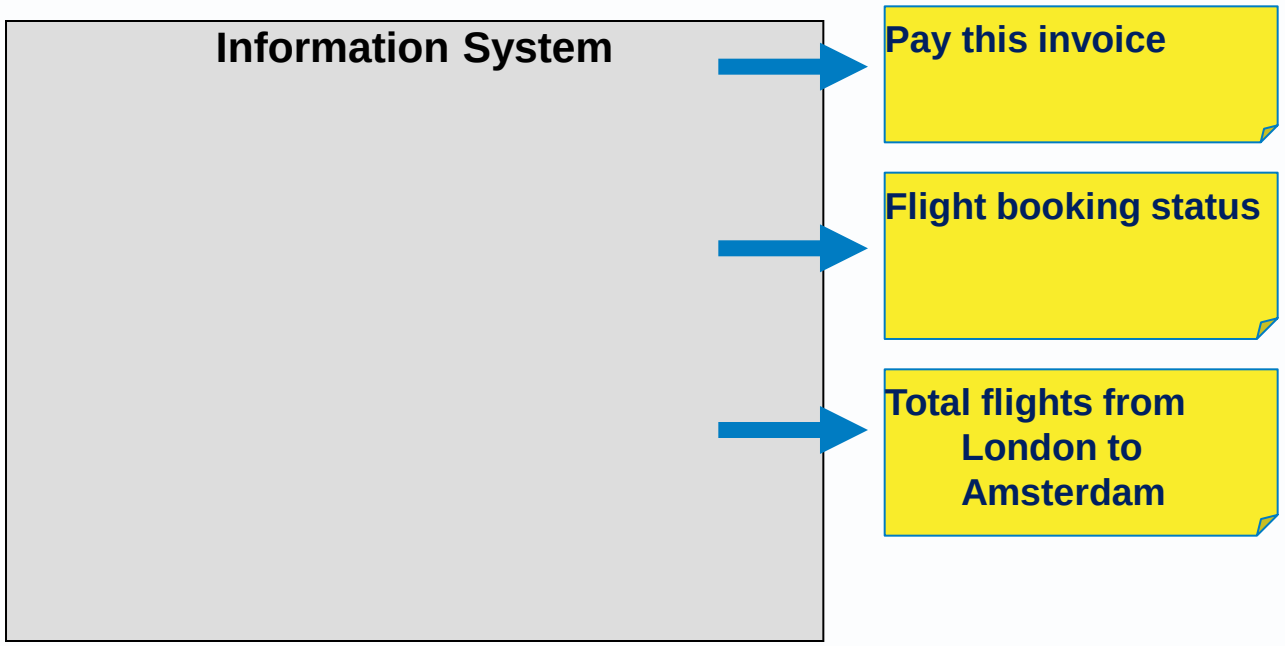
1. Define the desired effects or outcomes of a system
2. Bound the system by defining the **outputs** to produce 1

- Throughout, attend to any overarching principles, constraints, and trade-offs between design options.



2. Bound the system by defining the outputs to produce 1

- A system serves its purpose by producing output data
  - Sending messages – often with instructions (e.g. pay this invoice)
  - Reporting the state of a business process (e.g. booking a flight)
  - Reporting data aggregated from many entities or activities



At this point…

- Look for system or components that already produce the outputs you want, or near enough.

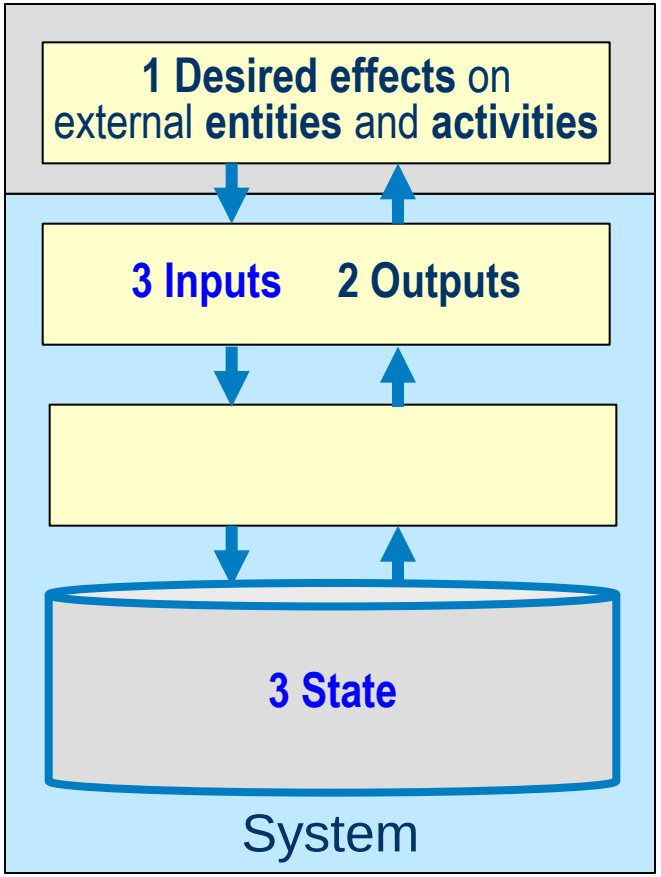
- The general principle being
  - Reuse before buy
  - Buy before build

- But before reuse or buy, study
  - Non-functional requirements
  - Stored data requirements
  - System integration requirements
  - TCO, including operation and maintenance

A system theory-based process

Avancier

1. Define the desired effects or outcomes of a system
   - (wrt entities and activities that are to be supported or enabled, monitored or directed).
2. Bound the system by defining the outputs to produce 1
3. Define the **inputs and state** needed to produce 2

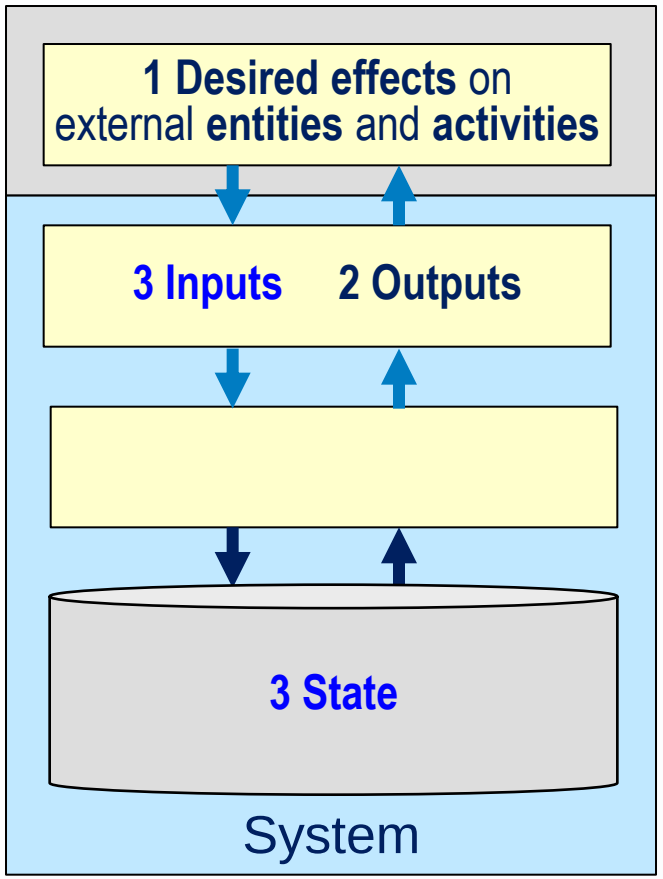


- Throughout, attend to any overarching principles, constraints, and trade-offs between design options.

3a Define the inputs needed to produce 2

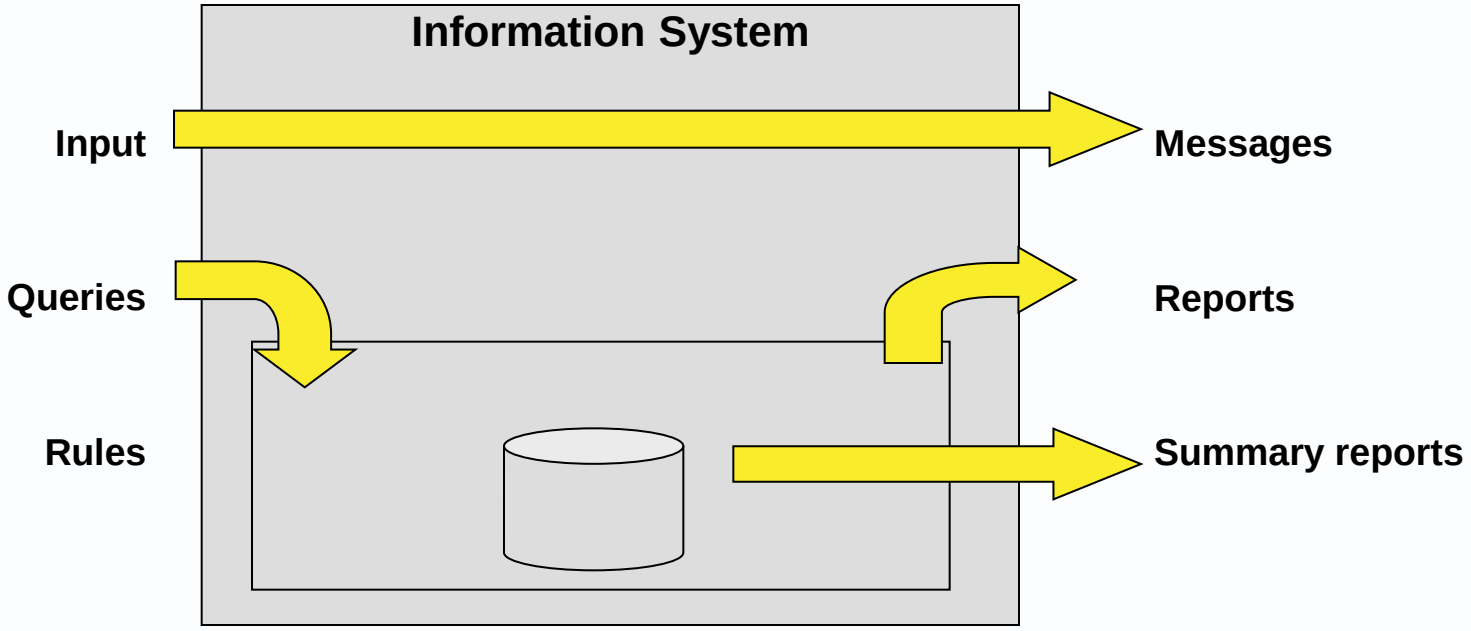
- To monitor the state of external entities and activities the system must collect input data

- And record the state of entities and activities in a data store



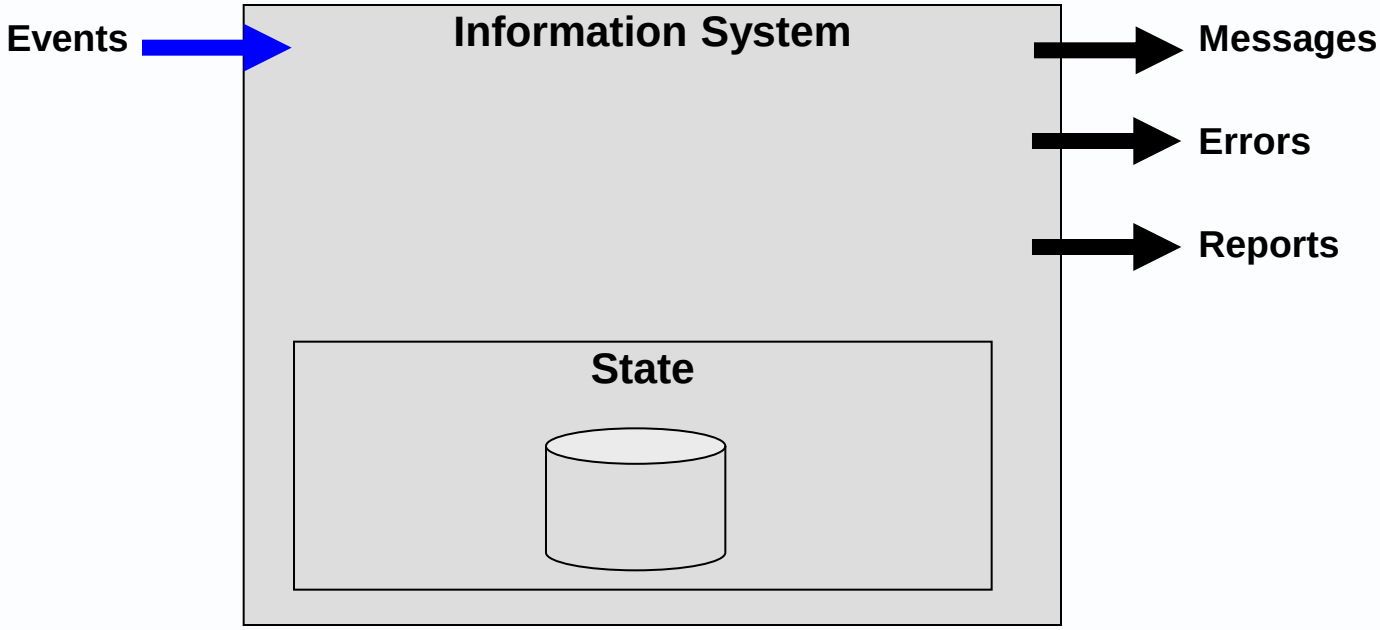
What can trigger the system to produce output?

- **Input -** immediately transformed into output
  - E.g. log events recognised in the input data stream.
- **Queries** – on data stores recording business entity state.
- **Rules** - an internal condition or time event.



Some input data represents events

- Sometimes a real world event that already happened
  - Gas meter connection
  - Gas meter reading (manual)
- Sometimes the happening of the event itself
  - Conference seat booking.
  - Gas meter reading (electronic)



Input data captures business events (not entities)

- Giving users screens to manipulate entity state data is dangerous.
- Data capture screens should be designed to help users record business events (not overwrite entity state data).

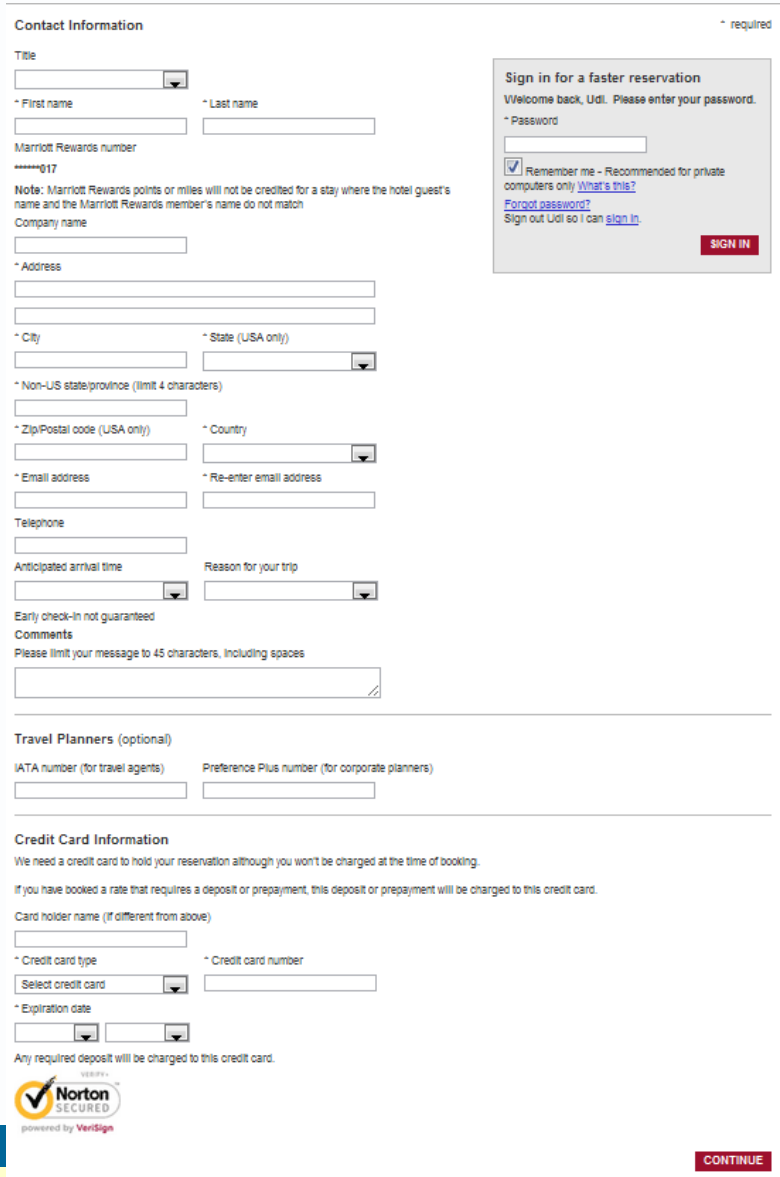
Hmm…

- A process control system processes each event as it arrives
- A business information system processes each event as it arrives, but also
  - Entity state copies
  - Queries and other triggers
  - Fix-up data

Entity state copies

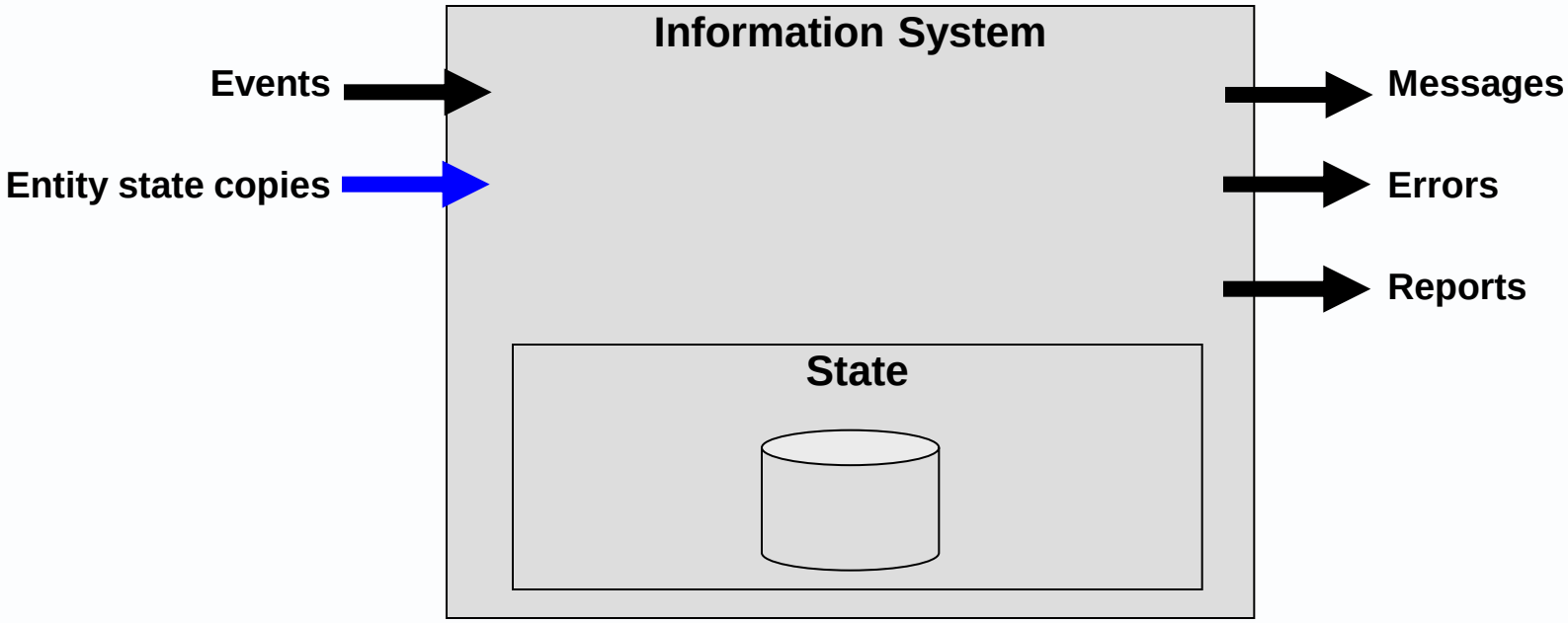
Avancier

- An application form summarises a life history covering many events.
- The customer
  - Was born
  - Moved into their current address
  - Joined their current employer
  - Were assigned to their current role
  - Obtained a current credit card.
- So the form represent the latest state of one or more life histories.



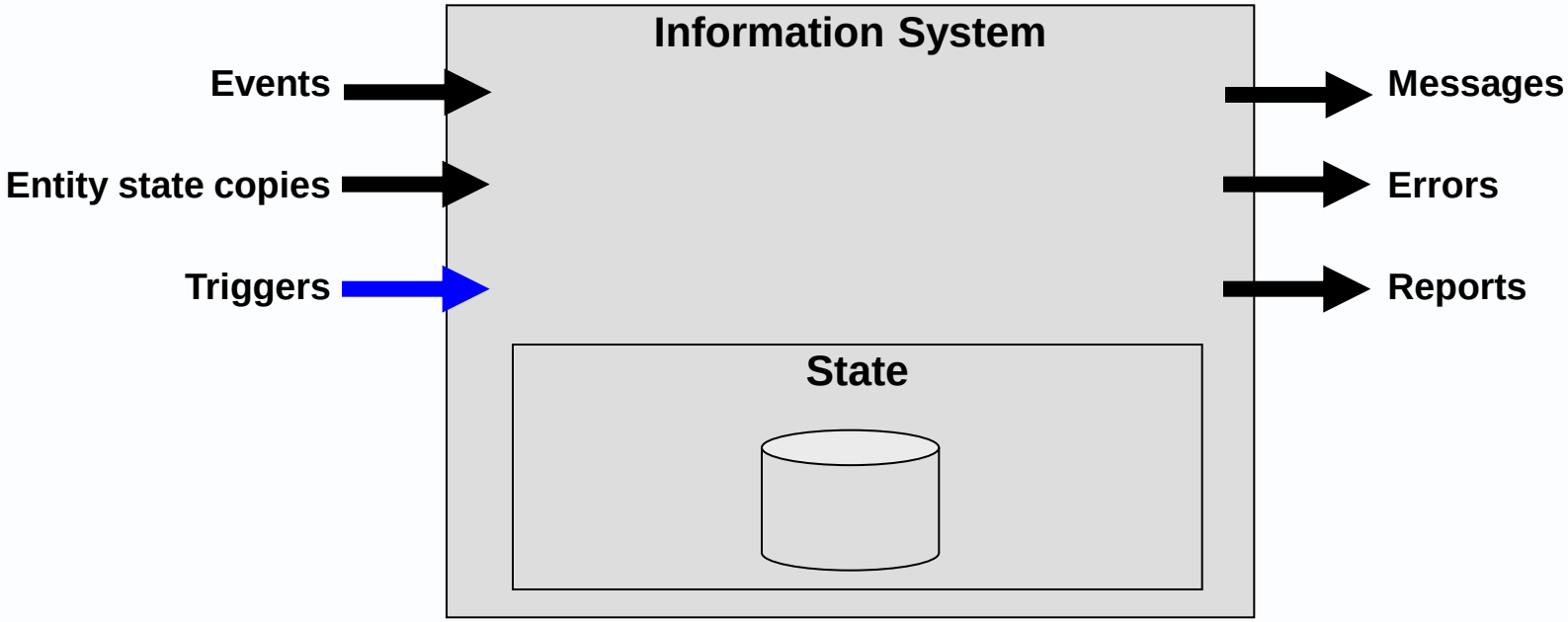
Entity state copies

- Application forms
- And data loaded from other systems.



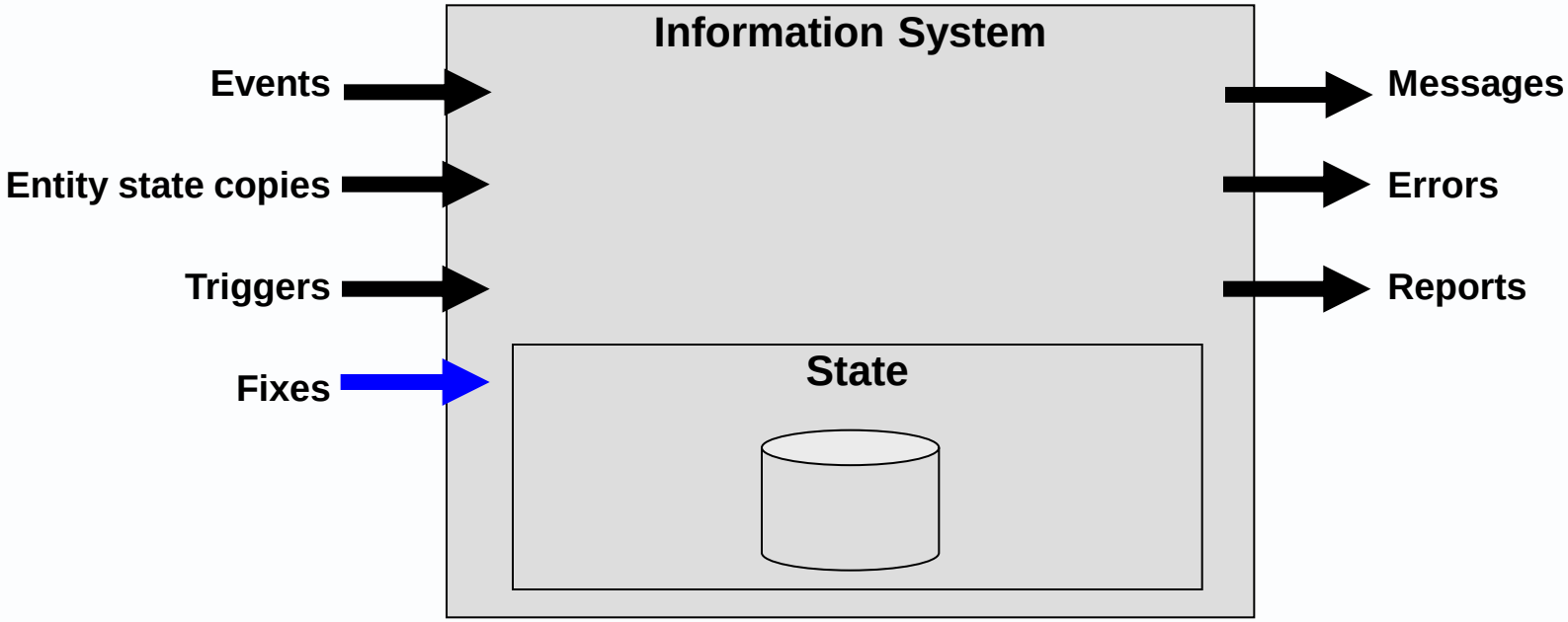
Queries and other triggers

- Introduce no data into the system
- But identify a process to be started or an output to be produced
- But could be recorded as events?



Fix-up data

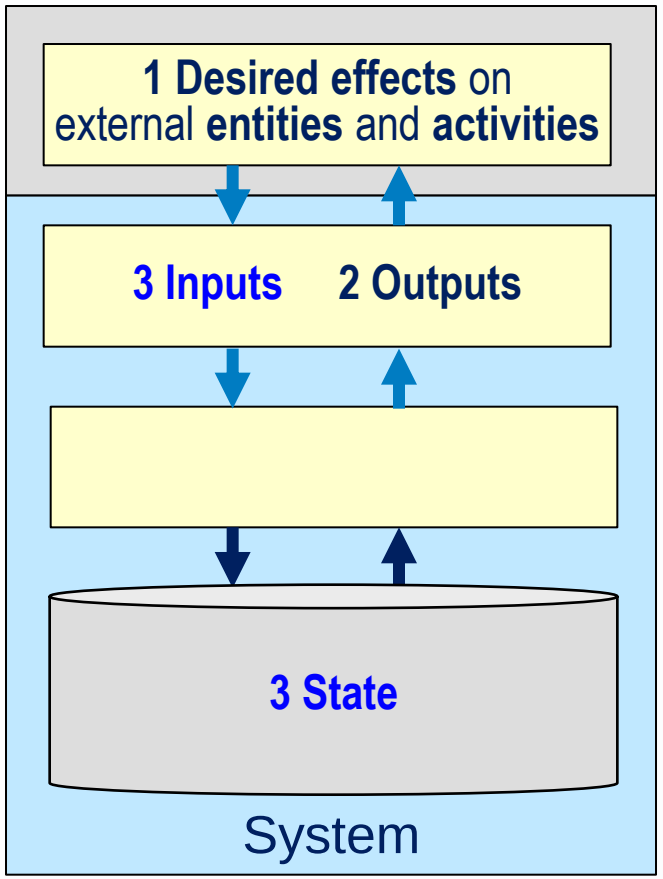
Inputs that fix errors in entity state data by directly manipulating it.



3a Define the state needed to produce 2

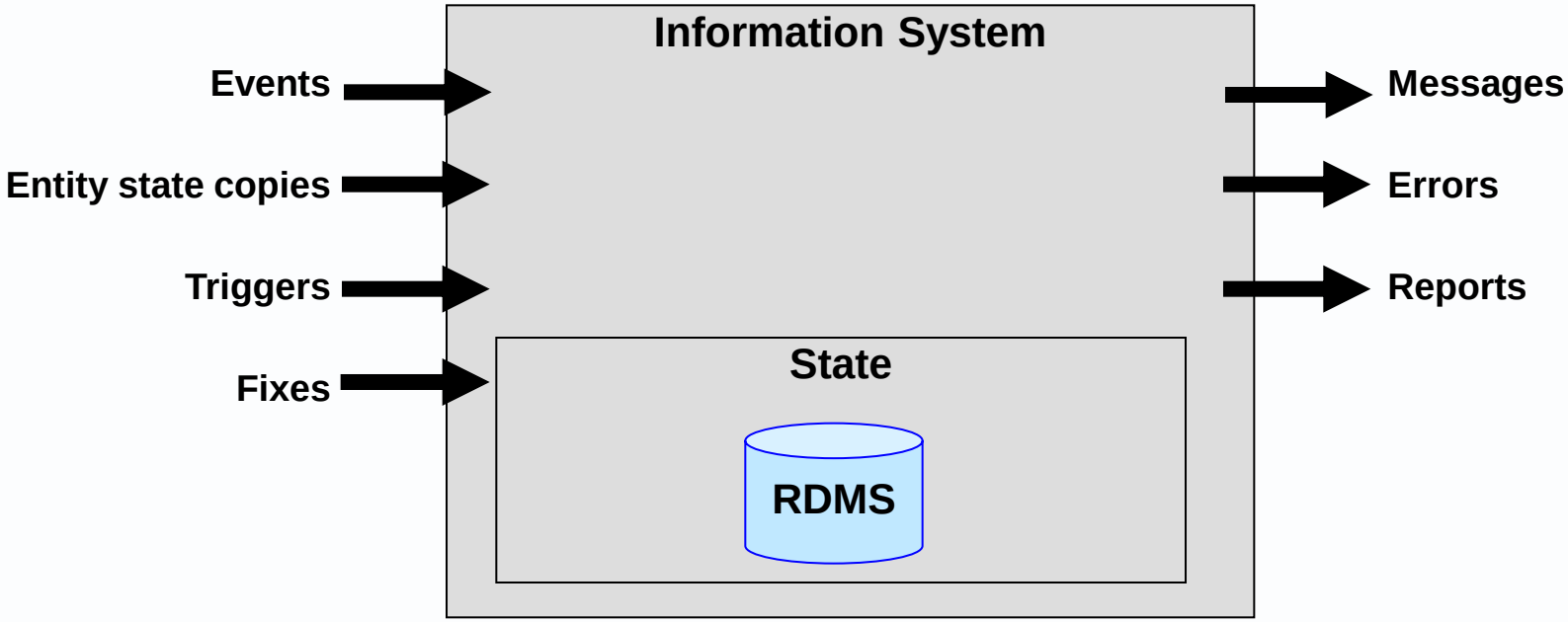
- To monitor the state of external entities and activities the system must collect input data

- And record the state of entities and activities in a data store



1st variant – traditional RDBMS

- If you want data to be consistent, conformant to rules, correct and up to date, then consider an RDMS with a normalised database structure and referential integrity checking by ACID transactions



Applying the KISS / YAGNI principle

- What data must be stored?
  - Only the data needed to produce the currently required outputs
- How much historical data must be maintained?
  - Only the data needed to produce the currently required outputs
- How de-normalised should the data store be?
  - Only so far is necessary to meet NFRs

What data must be stored?

- Only the data needed to produce the currently required outputs

- The OO mantra used to be that we should try to 'model the real world'.
- The world is infinitely rich and complex, and can be modelled in infinitely many ways.
- So ‘modelling the real world‘ is meaningless without a requirement – without knowing the specific needs of the business.
- Shape your data model to fit the data structures the system must capture from the business and provide to the business.

How much past or historical data must be maintained?

- Only the data needed to produce the currently required outputs

- Updating data erases knowledge of earlier states

- Reason not to store history include
  - To save time and money
  - To simplify design, save design and maintenance effort
  - To save data storage space
  - To optimise system performance.

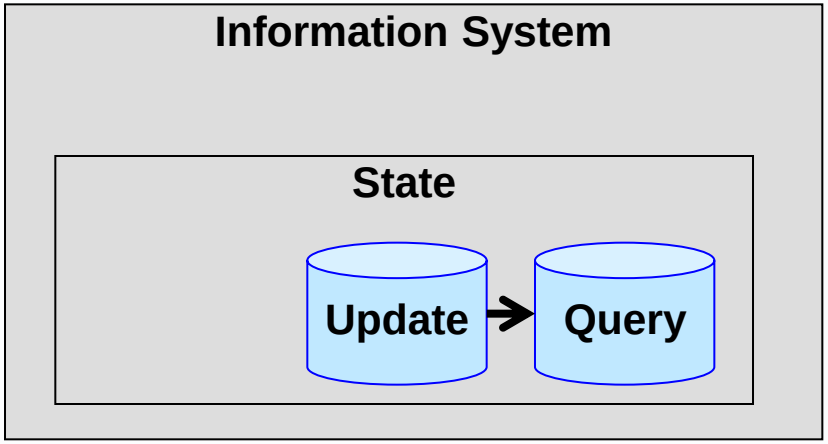
How de-normalised should the data store be?

- Only so far is necessary to meet NFRs

How to process queries?

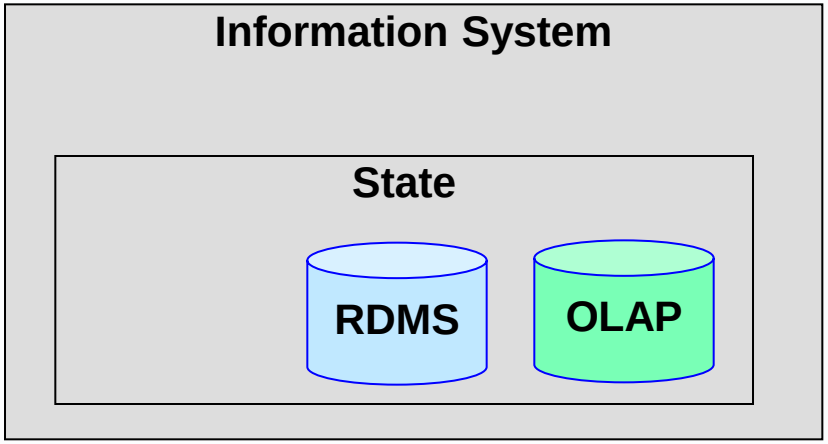
- Queries on the main data store may slow things up
- If queries are long or resource-hogging, then can you
  - instrument them to tell users how long they will take (with a cancel option)?
  - postpone them for overnight processing?

- Can you create a read-only data store copy for queries?



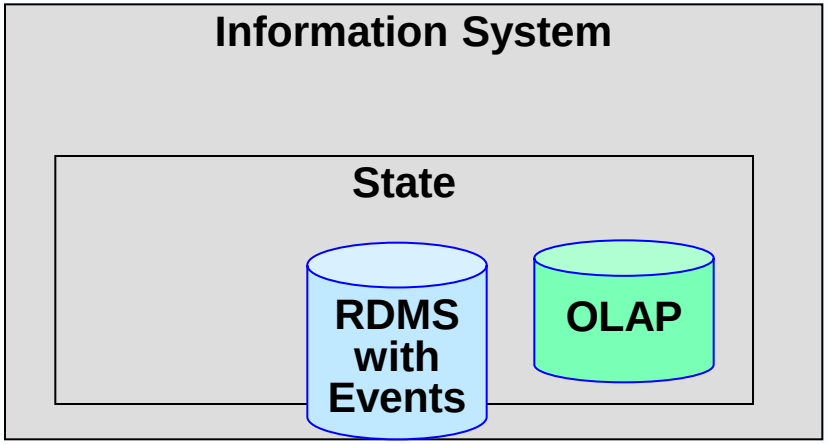
2nd variant: separate store for “business intelligence”

- De-normalised OLAP for management information reports
- This data store can hold more historic data, distilled



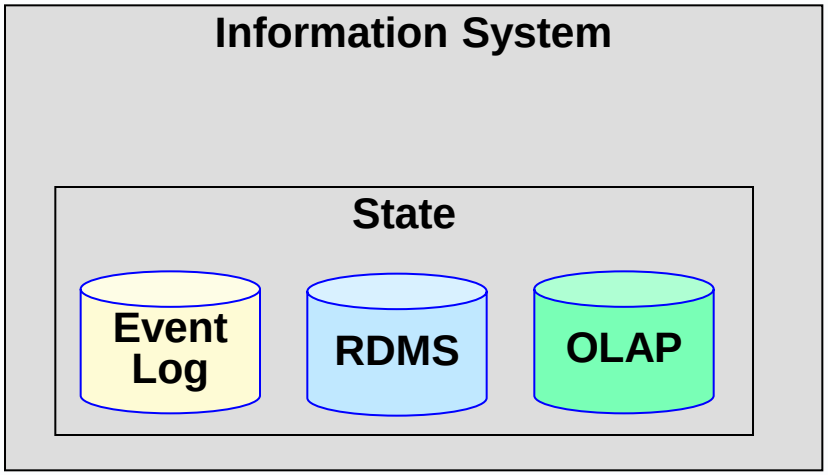
What if we need remember past states?

- Current state data enables us to answer many questions.
- An OLAP data warehouse can hold some history
- But what if we want to
  - query past events
  - reconstruct past states
  - correct events misapplied in the past?



3rd variant: Separate store for “event sourcing”

- Store update events separately from current state

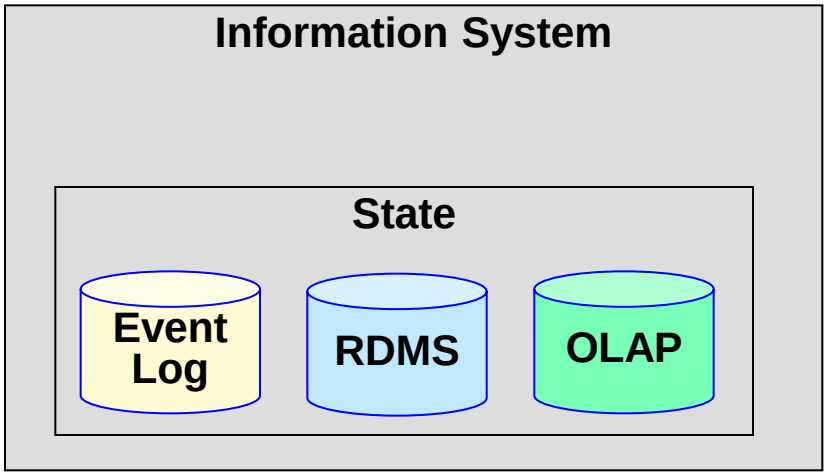


Three possibilities

- Complete Rebuild:
  - Discard the state completely and rebuild it by re-running the events from the event log
- Temporal Query
  - Determine the application state at any point in time.
  - Start with a blank state and rerun all events up to that time
  - Or start from a past state, and rerun events since
- Event Replay (if past event incorrect or out of sequence)
  - Reverse events to the point before the wrong event, then replay the right event and later events
  - Or else do a complete rebuild
  - (Out sequence is a common problem with systems that communicate with asynchronous messaging.

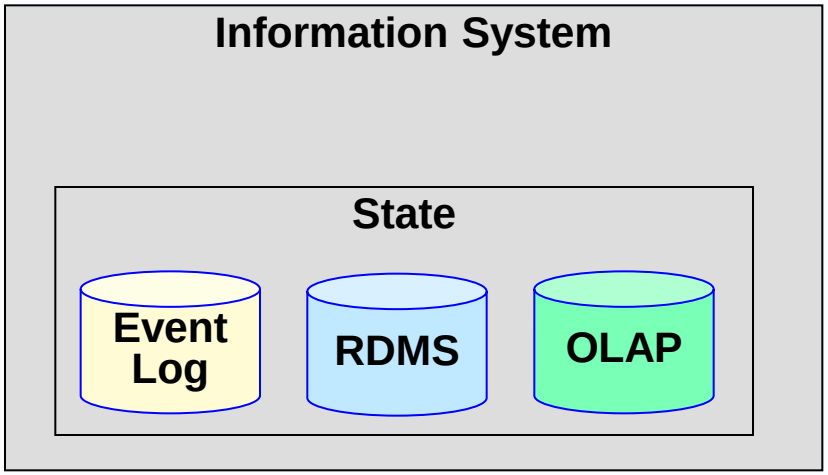
Which data store to query?

- An event log is a more complete model of the real world than the current state data in an RDMS
- It holds the complete history, though un-normalised
- Is potentially useful for data mining - deep or complex queries.
- How about directing queries to the event log?



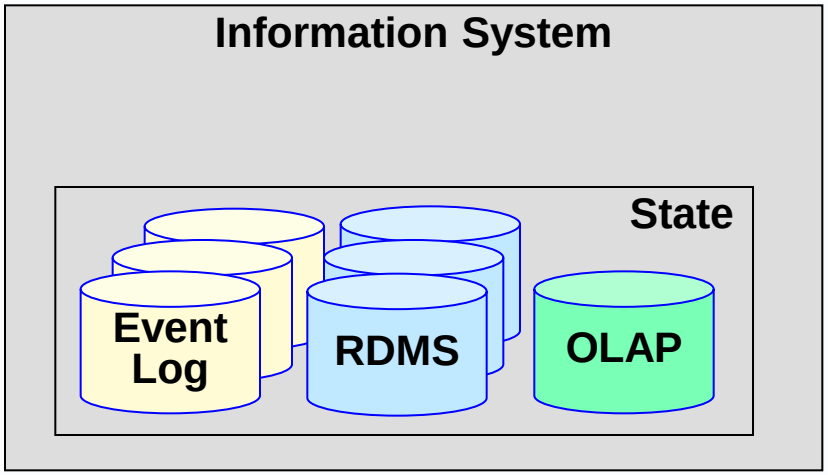
Meeting NFRs

- Parallel data stores enable parallel processing
- Update RDMS first
  - Copy successful events to Event Log
  - Copy what is needed to the OLAP store
- If copying is asynchronous, beware disintegrity



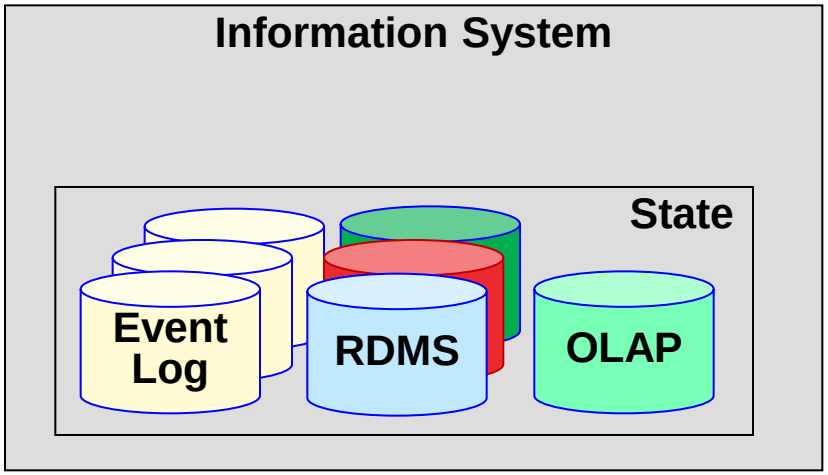
4th variant: separate data populations

- Partition the RDMS database *rows* (e.g. different geographies) and replicate the application
- Good for local performance (throughput and response time)
- Difficult to maintain shared data (reference data)
- Difficult to maintain OLAP
- Many event logs?



5th variant: separate into micro apps (aka micro services)

- Partition the RDMS database *tables* for maintenance by different micro services
- Difficult to maintain shared data (reference data)
- Difficult to maintain OLAP
- Many event logs (different events)



Simple thoughts

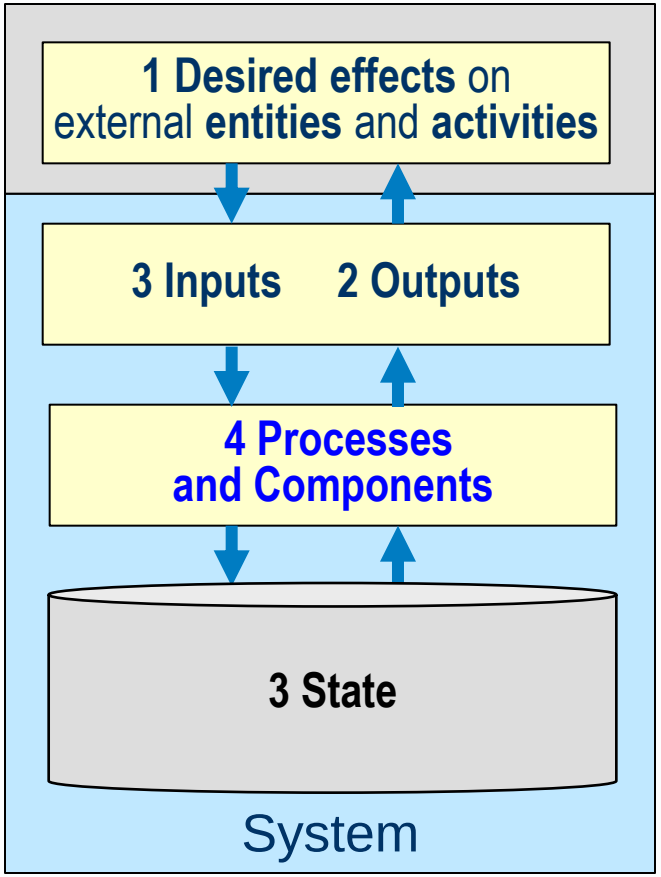
- *Consolidation* - for simplicity and data integrity
- *Physical replication* –for throughput & availability.
- *Logical partitioning* – for agile micro apps

- But KISS
- Don’t add more data stores than you need
- Look for a technology platform to hide distribution complexity from you
- And keep the design trade offs in mind.

A system theory-based process

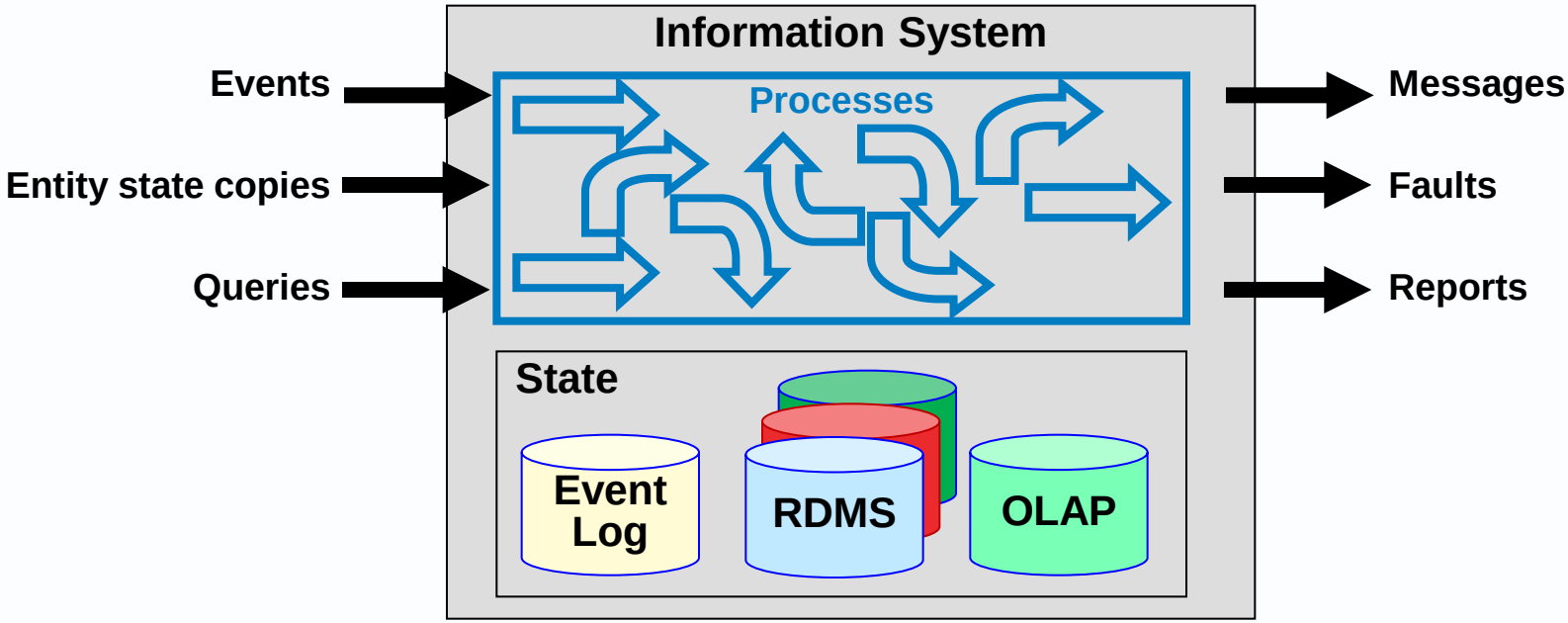
1. Define the desired effects or outcomes of a system
   - (wrt entities and activities that are to be supported or enabled, monitored or directed).
2. Bound the system by defining the outputs to produce 1
3. Define the inputs and state needed to produce 2
4. Define the **processes** and **roles** needed to perform them.

- Throughout, attend to any overarching principles, constraints, and trade-offs between design options.



4 Define the processes and roles needed to perform them

- Pre-event validation?
- On-event validation?
- Post-event validation?



How to process events?

- When, where and how to validate the business rules that determine whether an event is accepted or rejected?

- Service name: “Confirm hotel reservation”

- Preconditions:
  - The customer’s email address is correctly formatted
  - The customer’s post code is correctly formatted
  - The credit card payment is accepted
  - There are enough rooms left in the hotel.

Pre-event validation of input

- Aggressive client-side validation – before updating the database

- Use drop-down lists to constrain data entry values

- Preconditions:
  - The customer's email address is correctly
  - The customer's post code is correctly formatted
  - The credit card payment is accepted
  - There are enough rooms left in the hotel.

Send email to address

Send messages to 3rd party systems

Pre-event query to test the current business data

True, another user may get in and change the stored data before you commit the event, but it is rare for events to fail later on the server-side for this reason."

On-event validation

- ACID transaction on database

- E.g. check the hotel booking on the server side
- Rollback the transaction if any precondition failed

Post event validation (as in CQRS)

- Server-side app accepts every event
  - After simple client-side validation
- Replies immediately with an acceptance message
  - “Thank you for your booking details.”
  - “We will send you an email to confirm”.
- So, the client does not have to wait for long and complex distributed processing to be completed
- The event will be processed asynchronously
- Any failures will be dealt with by compensating transactions

What if the credit card payment is not accepted?

- "We don't charge the customer's card when we collect the input data, so there is no issue with a failed credit card payment that needs to be handled [immediately] by the client-side javascript.
- There are only some specific types of rates for which the hotel charges your card when you make a reservation.
- But yes, sooner or later, trying to collect the money may fail.

Compensating Transaction

- The typical ecommerce approach is to send an email to the customer asking for an alternative form of payment, also saying that their purchase won't be processed until payment is made.
- In any case, it is only after the reservation is placed that the responsible service would publish an event about that.
- The service which collected the credit card information would be subscribed to that event and initiate the charge of the card when that event arrives (or not, depending on the rate rules mentioned)."

This example after one discussed by Udi Dahan

What if there are not enough rooms left in the hotel?

Avancier

- "First, there is overbooking
  - Hotels accept more reservations than rooms available because they know that some customers will cancel, and some just won't show up.

- Second, there is a manual compensation process
  - If too many people show up, then a hotel will
  - bump you up to a higher class of room (if available), or
  - call a "partner" hotel nearby and put you up there instead."

This example after one discussed by Udi Dahan.

Compensating Transaction

The need for an entity event model

- In analysis you need some kind of entity event model showing
  - persistent entity types
  - transient event types (think roll-backable transactions).

- How you use these models in system design is a dictated by non-functional requirements
  - The entity-oriented view may be used to inform database design and/or “domain model” design
  - The event-oriented view may be used to inform the design of “control objects” and entity interactions.

A few messages for an architect

1. KISS
2. There is no single way to design systems
3. There are always trade offs
4. Vendors and fashion may push one design pattern, but an architect must be mindful of the opposite
5. Don’t overcomplicate the design to meet fanciful throughput and concurrency numbers that stakeholders agree are very unlikely in the next 5 years (YAGNI and the money will follow)
6. Beware that the design of data processing systems (e.g. division of data between data stores) has an impact on the business processes in the wider human activity system.
7. An architect must understand design patterns, trade offs and their impacts on the wider business system.

Left overs

Deploy - with NFR trade offs in mind

- Define nodes and platform technologies needed
- Define the deployment of modules to platform technologies and nodes
- Define how the nodes interact

The process must involve

- Iteration
  - Interleave thinking about outputs, processes and components.
  - Decompose systems and processes, then repeat
- Prioritisation
  - Attend to critical and risky outputs first
  - If it looks to be costly, risky or impractical, a change request may be made
- Requirements change management
  - Continual through the process
- Governance
  - Designers are governed from above
  - Designers govern system builders

How to modularise?

- Modularisation decisions are an outcome of
  - the time gap between inputs and outputs
  - the structure and content gap between inputs and outputs
  - physical distribution of suppliers, consumers and components
  - technology choices
  - general design principles and design patterns
  - trade-offs between quality ("non-functional") attributes
  - social pressures from vendors, colleagues and others
  - fashion

- There are many ways for the system design to become more costly, complex and difficult to maintain than it should be
- And so, a lot for a software architect to understand