<JOIN>DEV CLUB</JOIN>

CQRS: DO YOU REALLY NEED TWO DATABASES?

Dejan Miličić

DevRel@RavenDB

BDD@DevClub

2023-10-30

TODAY WE'LL TALK ABOUT

- Why do developers actually **love** changes?
- Why you should read **ancient books**?
- How to recognize a **Sith**? (they do not always come in pairs)
- **How many databases** are enough for a single project?

WHAT IS OUR JOB?

- We are building **systems**
- **System** developed for 1y, will be maintained for 9y
- So, 10% building, 90% responding to **change**
- Do developers love **change**?
- **Changes** are hard!
- But **why**?

WHAT IS A SYSTEM?

- **System** is a whole that consists of **Parts**
- **Parts** are affecting **System** behavior/properties
- **Parts** are interdependent for the effects
- Not a single **Part** has an independent effect on **System**
- **System** is a whole that cannot be divided into independent **Parts**

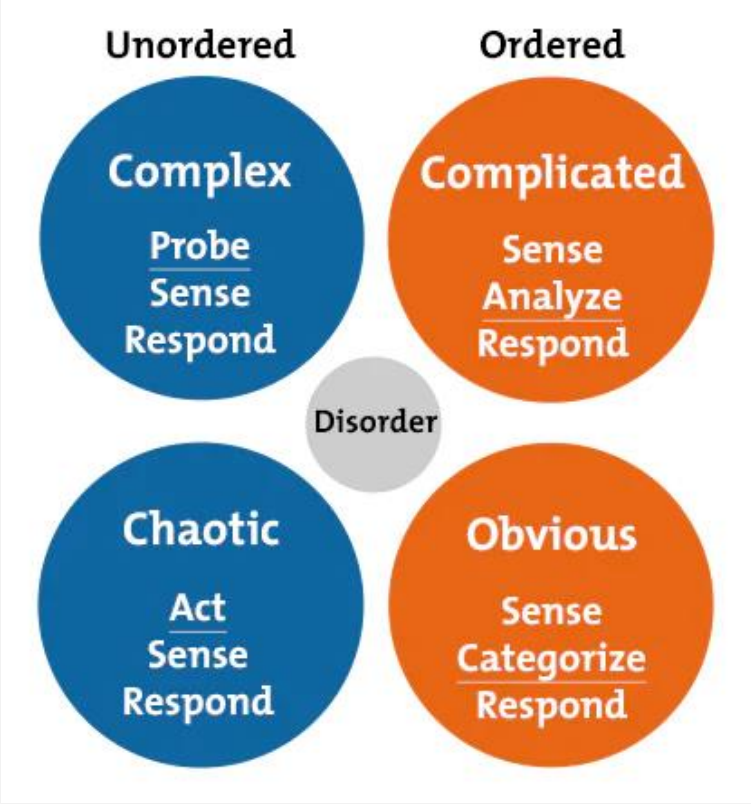
IMPLICATIONS

- **System** properties are properties of the whole which none of its **Parts** have
- When the **System** is taken apart it loses its essential properties
- **System** is not a sum of the behavior of its **Parts** but a product of their interactions.
- Reductionism

DAVE SNOWDEN: THE CYNEFIN FRAMEWORK, 1999

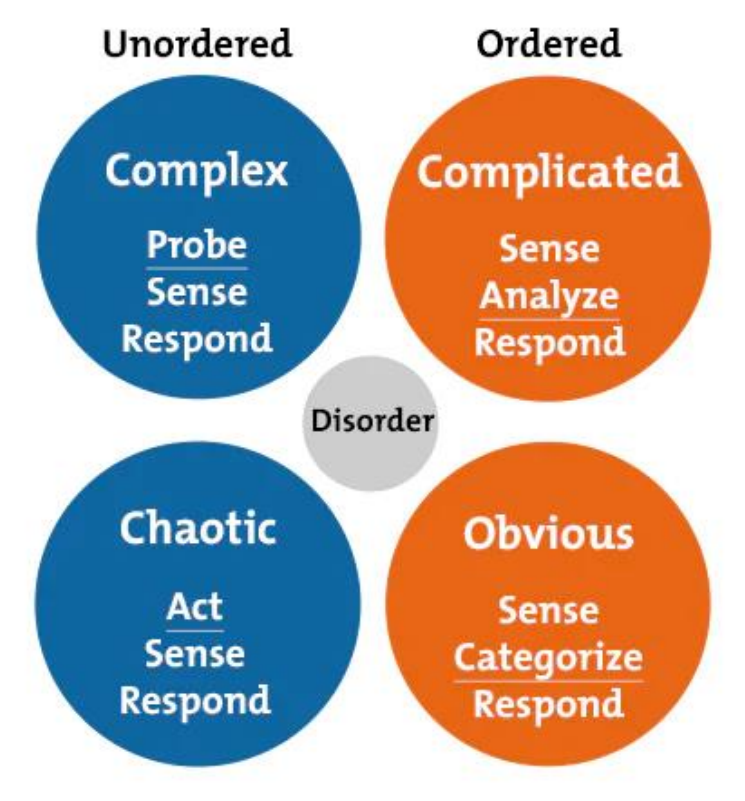
Manifestations of the system

1. Obvious/Simple
2. Complicated
3. Complex
4. Chaotic
5. Disordered



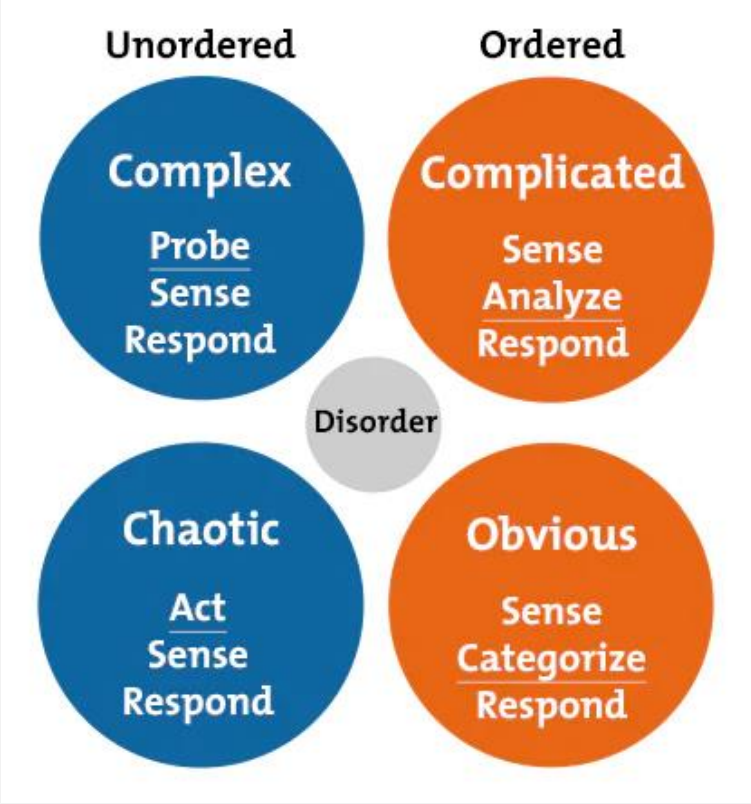
OBVIOUS SYSTEM

- direct cause-and-effect relationships
- little information
- few variables
- domain of best practices
- simple rules
- predictable effects



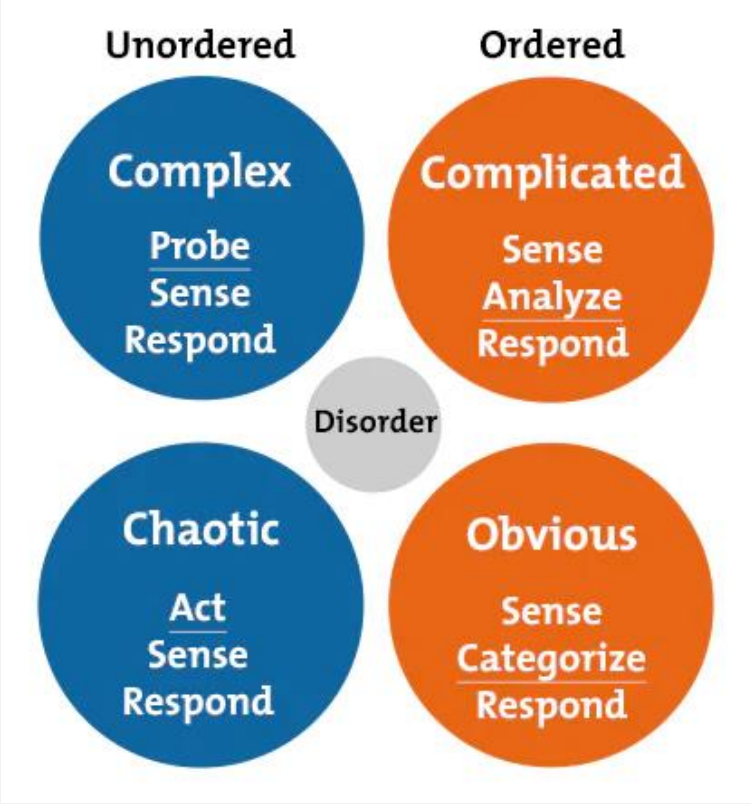
COMPLICATED SYSTEM

- complicated cause-and-effect relationships
- very high # of information
- very high # of variables
- domain of experts
- opposite of easy
- design patterns + good practices



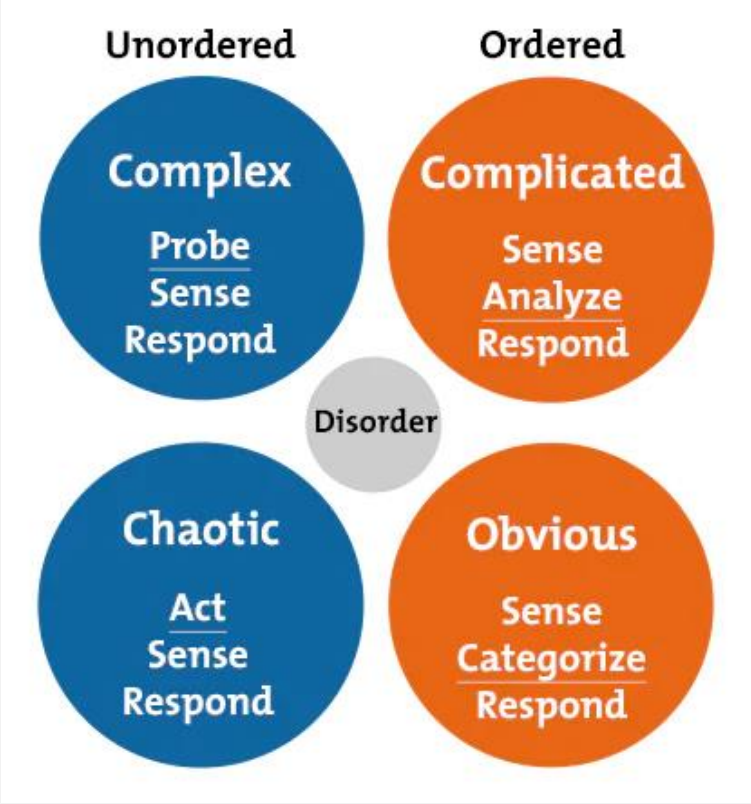
COMPLEX SYSTEM

- unmanageable amounts of information, factors, dependencies, variables
- accurate predictions are impossible, no planning
- domain of emergence
- explorative approach
- divide et impera
- opposite of simple



FRED BROOKS: NO SILVER BULLET, 1986

- **Essential** complexity
  - complexity of **domain**
  - cannot be avoided
  - cannot be changed
- **Accidental** complexity
  - complexity of **implementation**
  - problems engineers create and fix



FUNCTIONS IN MATHEMATICS

$$a = 25, b = 9$$
$$f(x) = \sqrt{x}$$

$$f(a), f(b)$$

FUNCTIONS IN MATHEMATICS

- no **side-effects** (immutability of variables)
- **deterministic** = always return same results for same args
- **pure** = no side effects + deterministic
- **referential transparency** = exchanging expression with its computed value

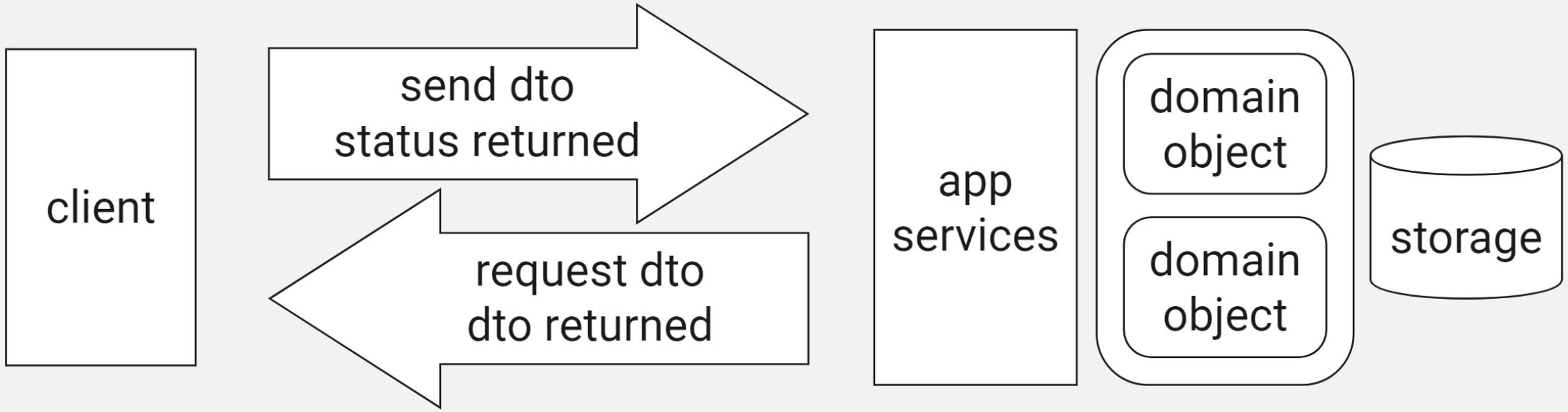
FUNCTIONS IN PROGRAMMING

- **can** have side-effects
- **can** be non-deterministic
- **can** be impure = have side-effects and/or be non-deterministic
- **can** lack referential transparency
- can interact with the environment (i.e. **not isolated**)
  - reading environment = non-determinism
  - modifying environment = generates side-effects

SIDE-EFFECTS : WHY SO BAD?

- Predictability
- Reasoning
- Testing
- Reusability
- Concurrency
- Composition
- Refactoring
- Debugging
- Dishonesty
- Cognitive load
- Portability
- Idempotency

A STEREOTYPICAL ARCHITECTURE



BERTRAND MEYER: CQS, 1988/1997

- "class designer as a patient **craftsman**"
- Class Design = **Interface** Design
- Ch 23: Principles of class design
- p 751/**1370** : CQS

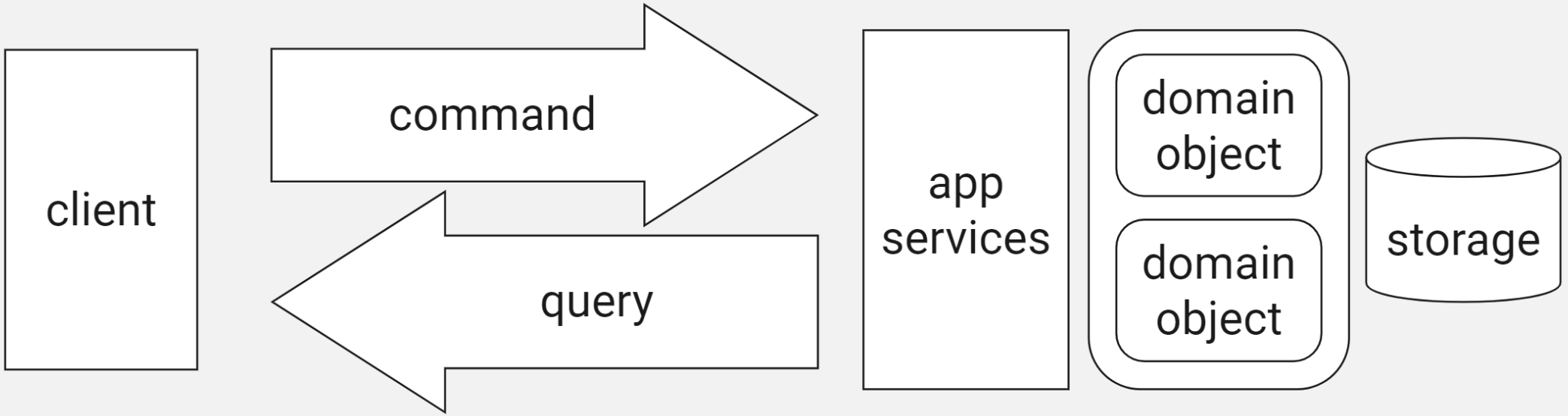
CQS : COMMAND-QUERY SEPARATION **PRINCIPLE**

- **command** changes state without returning results
- **query** returns results without changing state
- “asking a question should not change the answer”
- **query** should **not** produce **side-**effects
- applicable to “**surface**” (interface)

CQS : COMMAND-QUERY SEPARATION **PRINCIPLE**

- “may I return an object ID after saving it to the database?” a.k.a “Only a Sith deals in absolutes”
- popping value from the stack?
- CQS is a means, not a goal!

CQS-FLAVORED STEREOTYPICAL ARCHITECTURE



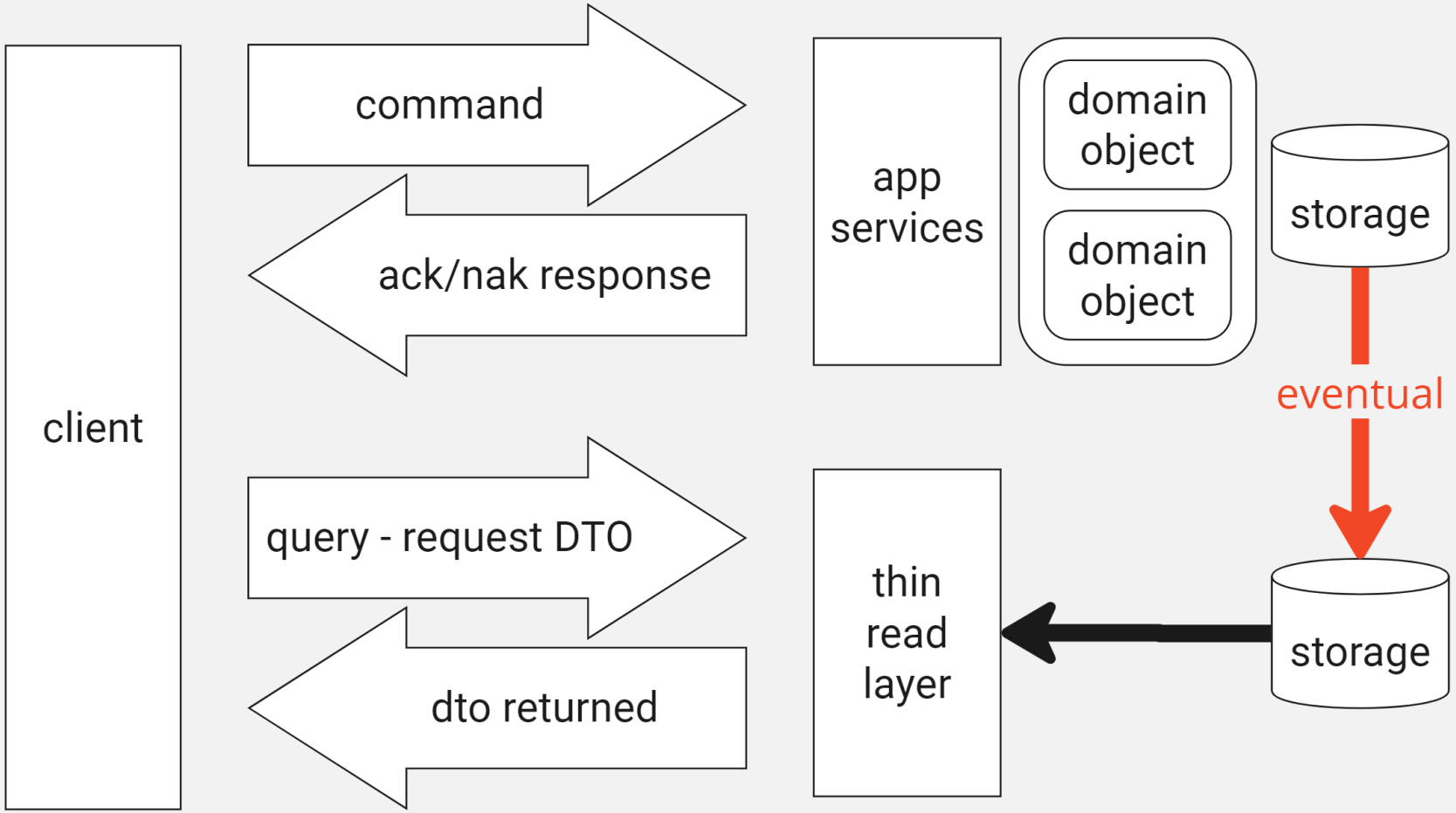
GREG YOUNG: CQRS, 2010

- **C**ommand and **Q**uery **R**esponsibility **S**egregation
- Evolution of CQS, propagating from surface
- CQS: Operates at the method or class level
- CQRS: Operates at the architectural level
- Separates the entire subsystem responsible for handling **commands** from the one responsible for handling **queries**

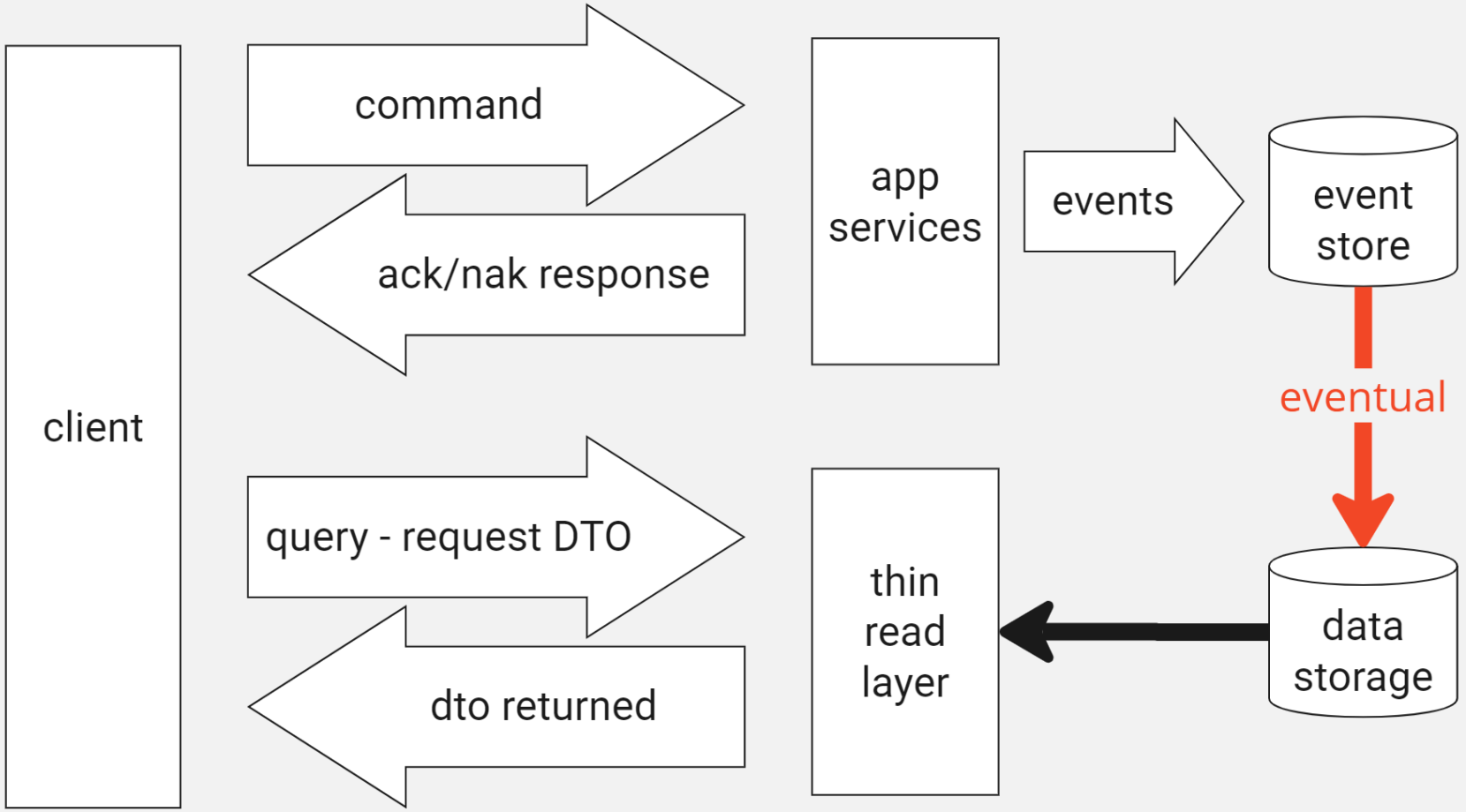
CQRS: CONSEQUENCES

- **Scalability**: reads and writes can be scaled independently
- **Flexibility**: optimizing read models for performance without affecting write side
- **Complexity Management**: complex business logic for writes can be isolated from read side which is much simpler
- **Security**: commands and queries can have different security policies

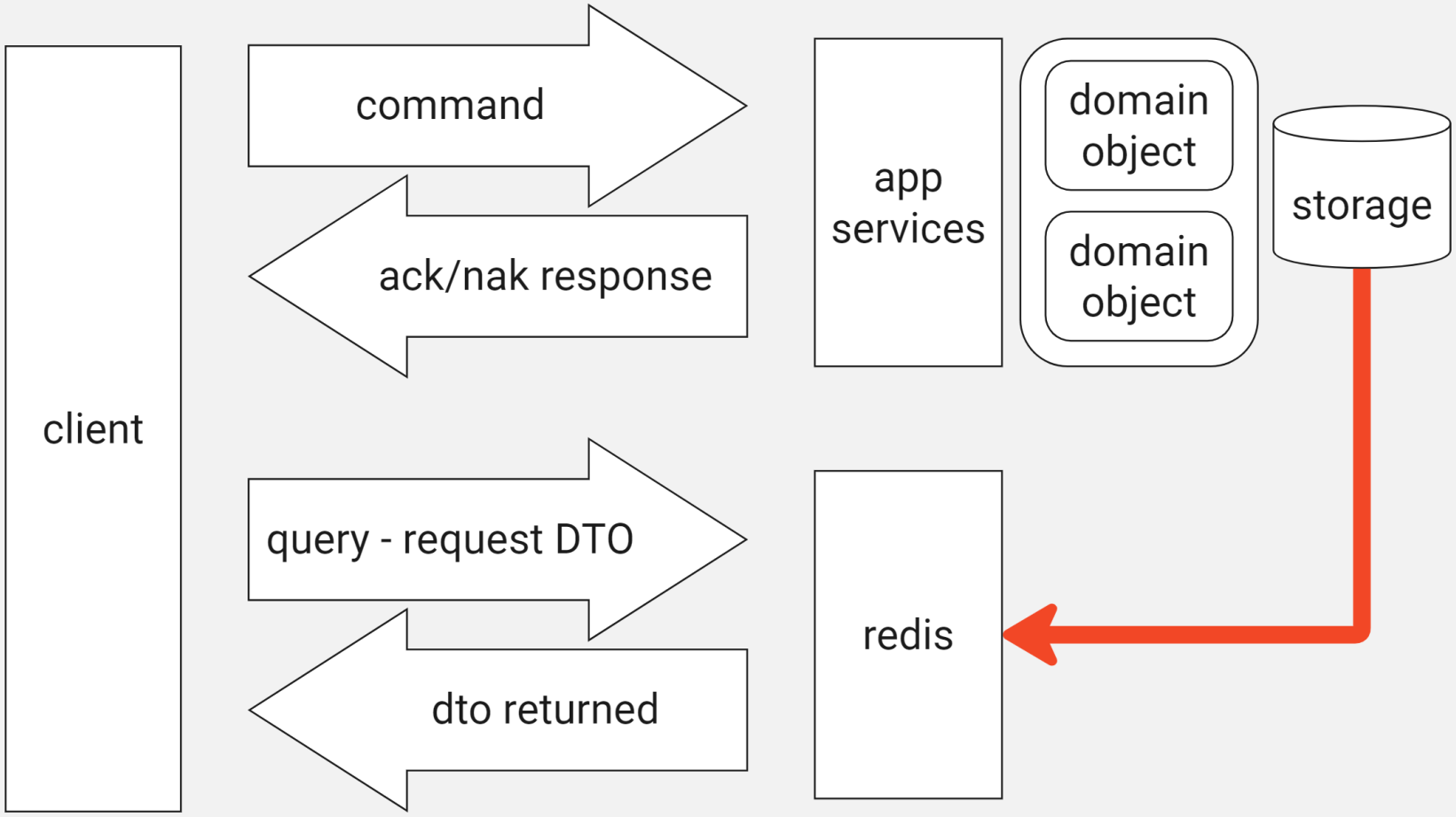
CQRS ARCHITECTURE



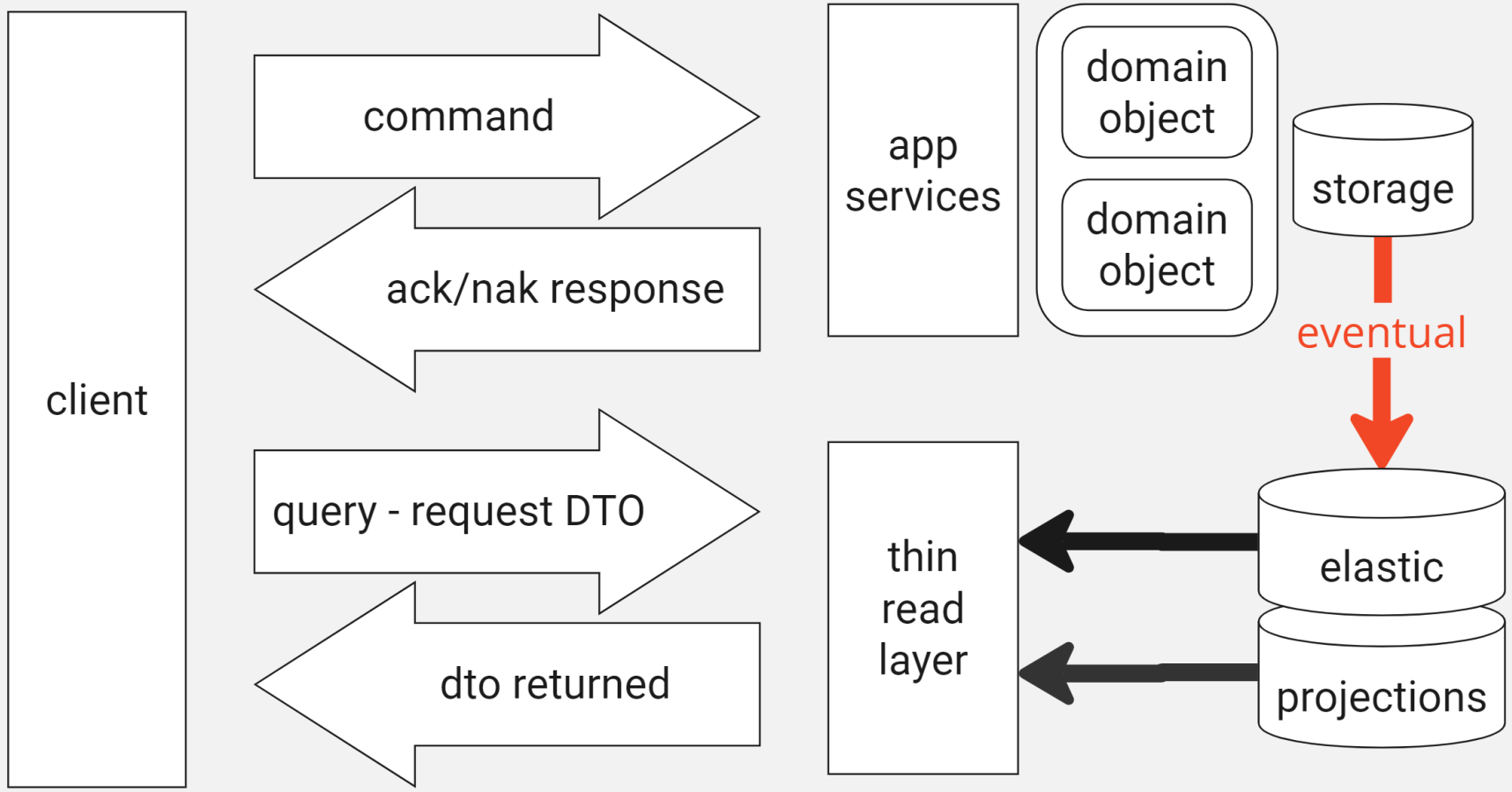
CQRS + EVENT SOURCING



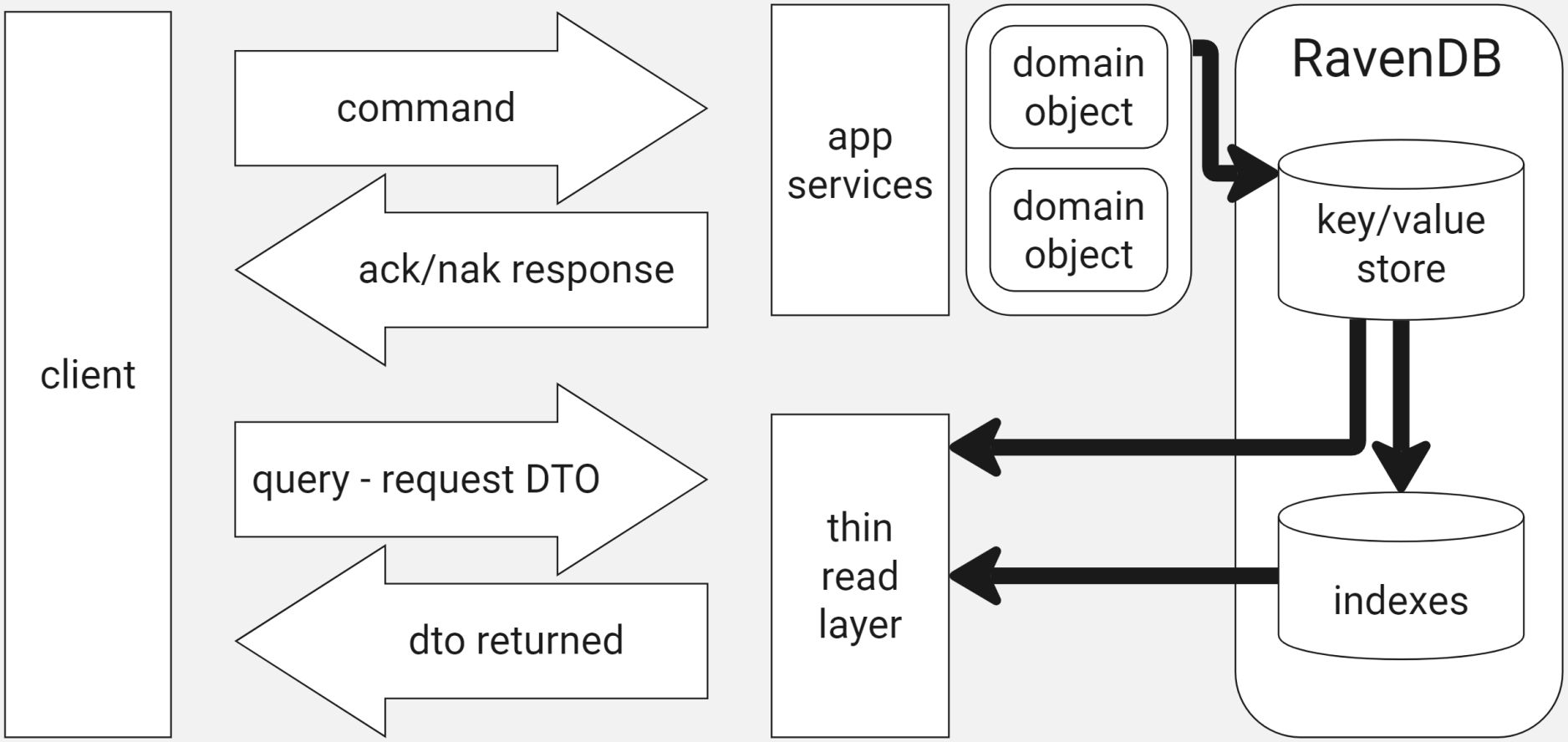
CQRS: CACHING FLAVOR



CQRS – THREE DATABASES



CQRS – ONE DATABASE



CQRS: BUT WHAT ABOUT…

- Eventual Consistency
- CQRS without Event Sourcing
- SQL or NoSQL
- Projections without Aggregates

CQRS: ADDITIONAL SCENARIOS

- Actor Model (a.k.a. The Future of Concurrency)
- Async operations
- Microservices
- Read-Write Disparity: high difference between the number of reads and the number of writes
  CQRS can help in optimizing for both separately

CQRS: CONS

- Complexity
- Eventual consistency
- No benefits for CRUD apps

SO, TODAY WE LEARNED

- Developers love changes because they keep them **employed**
- Old books contain **wisdom** – ignorance is harmful
- **Siths** deal in absolutes – do not be one
- How many databases are enough for a single project?
  IT DEPENDS

AND ALSO

- What are **pure** functions
- Why are **side-effects** bad… most of the time
- There is no silver bullet
- Be **pragmatic**

<JOIN>DEV CLUB</JOIN>