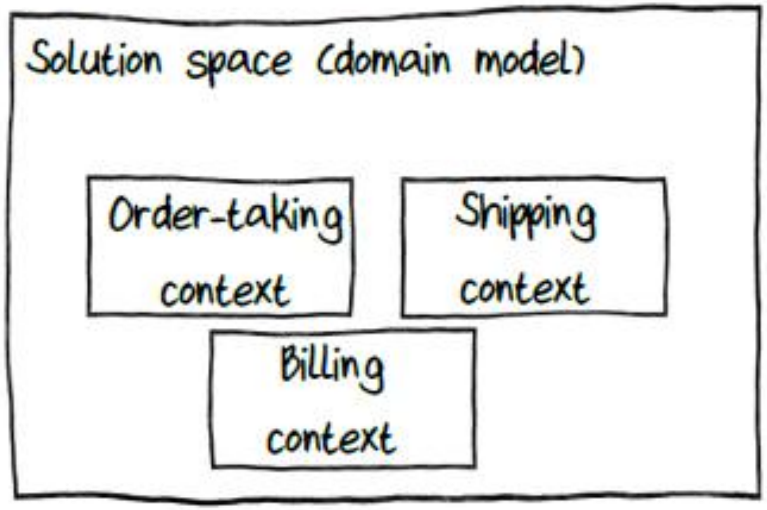
Functional Architecture

Bounded contexts are autonomous software components

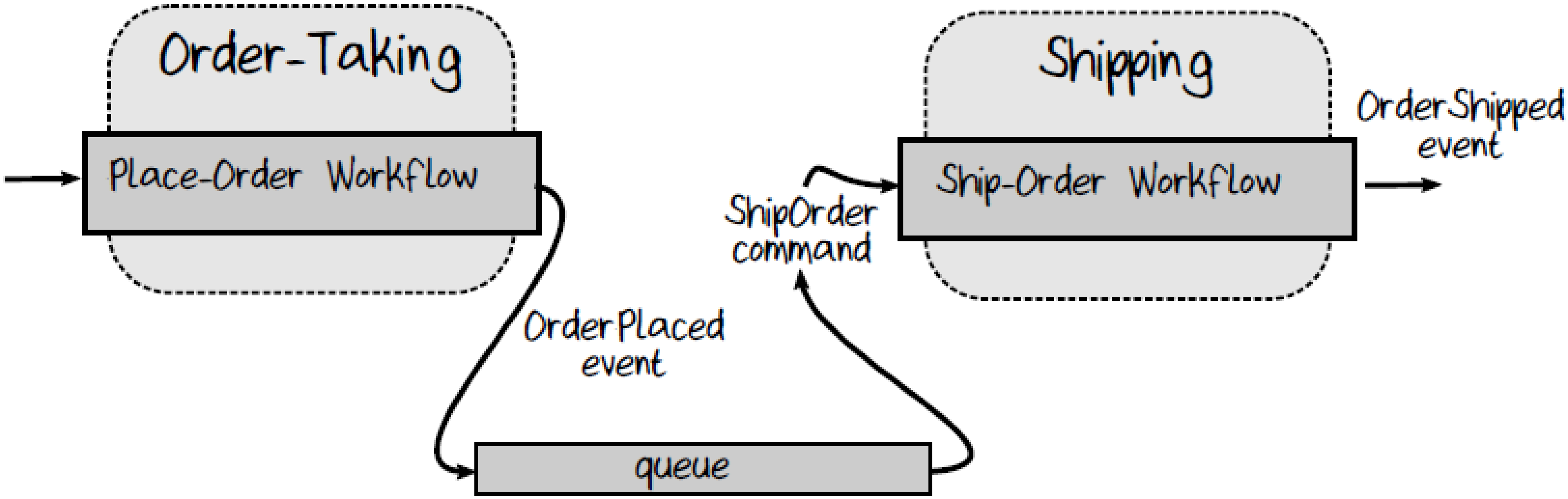


3 different architectures…

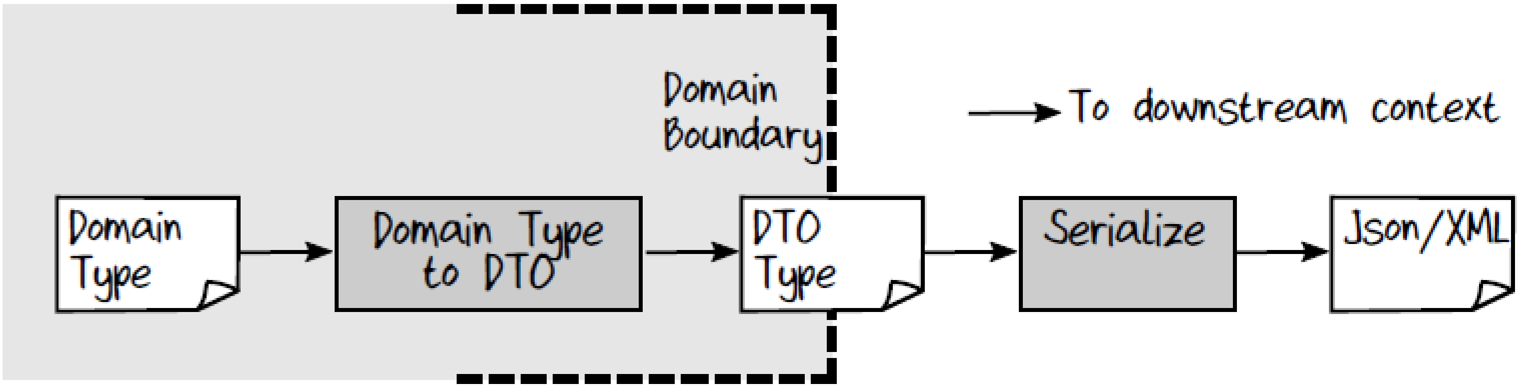
- For monoliths, each bounded context could be
  - a separate module with a well-defined interface, or
  - a .NET assembly. Alternatively, each
- For service-oriented architecture:
  - each bounded context is a separate container
- For microservices:
  - each individual workflow is deployed separately

How to communicate between bounded contexts?

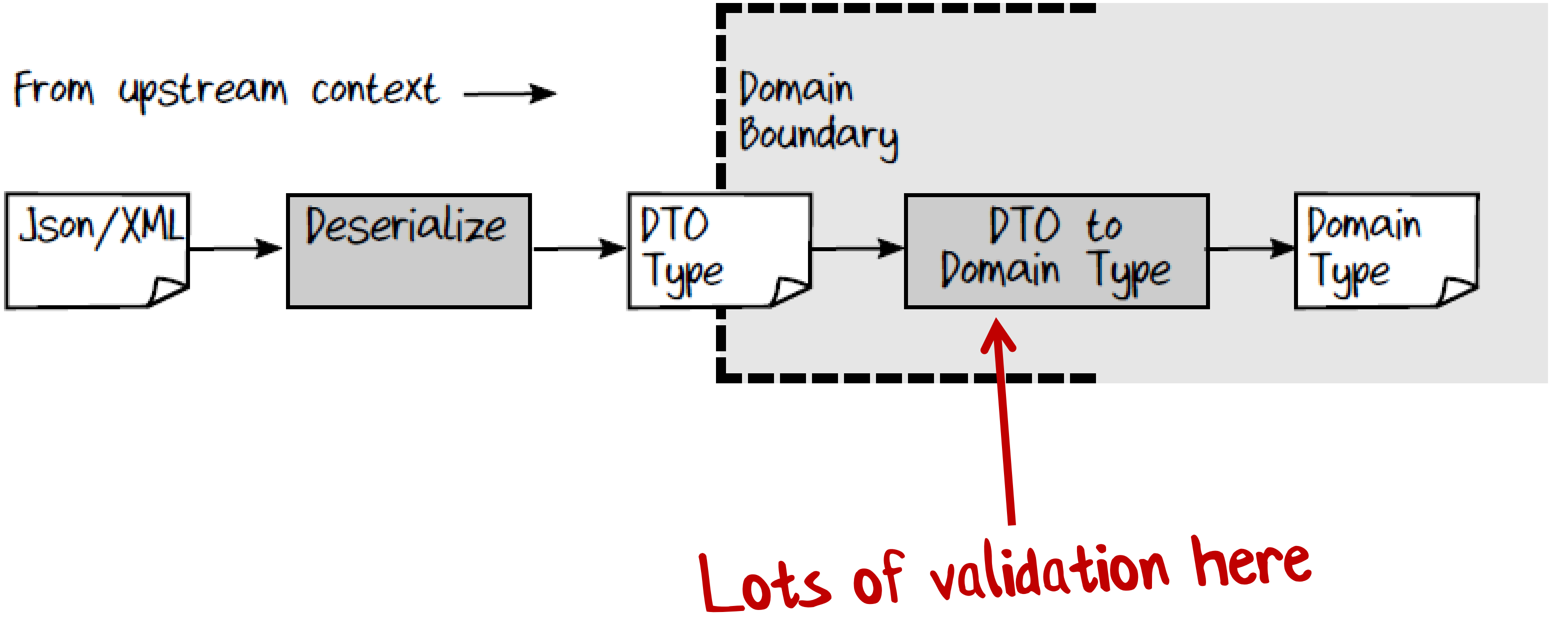
Answer: queues



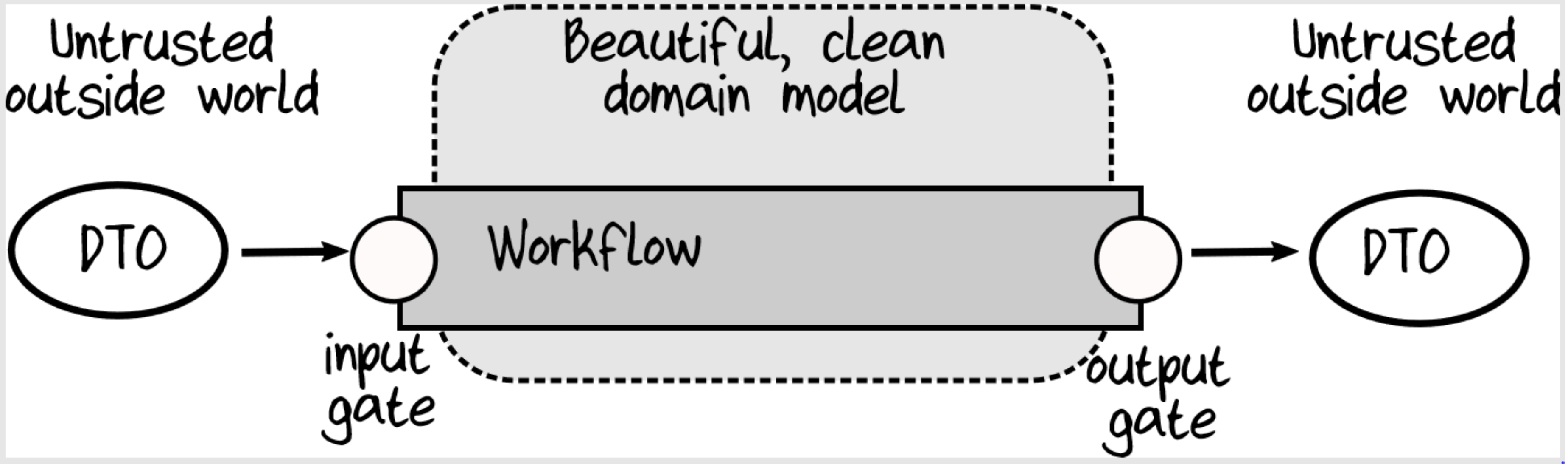
On the way out,
domain objects become DTOs



On the way in,
DTOs become domain objects

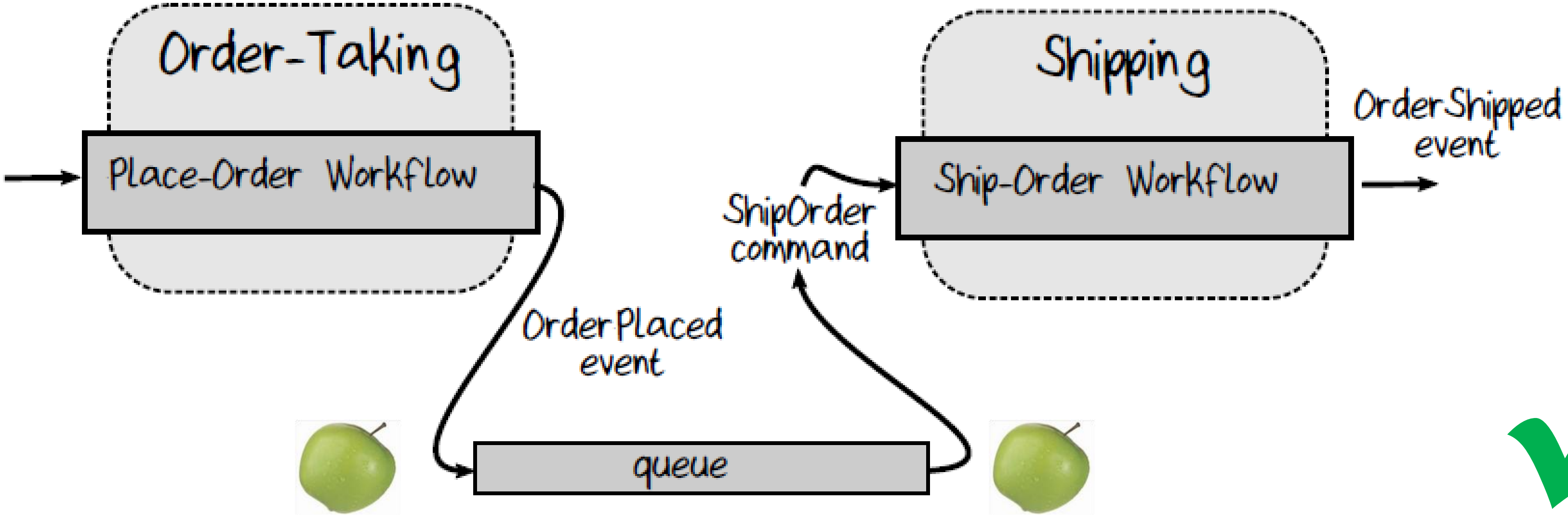


Do not trust the outside world



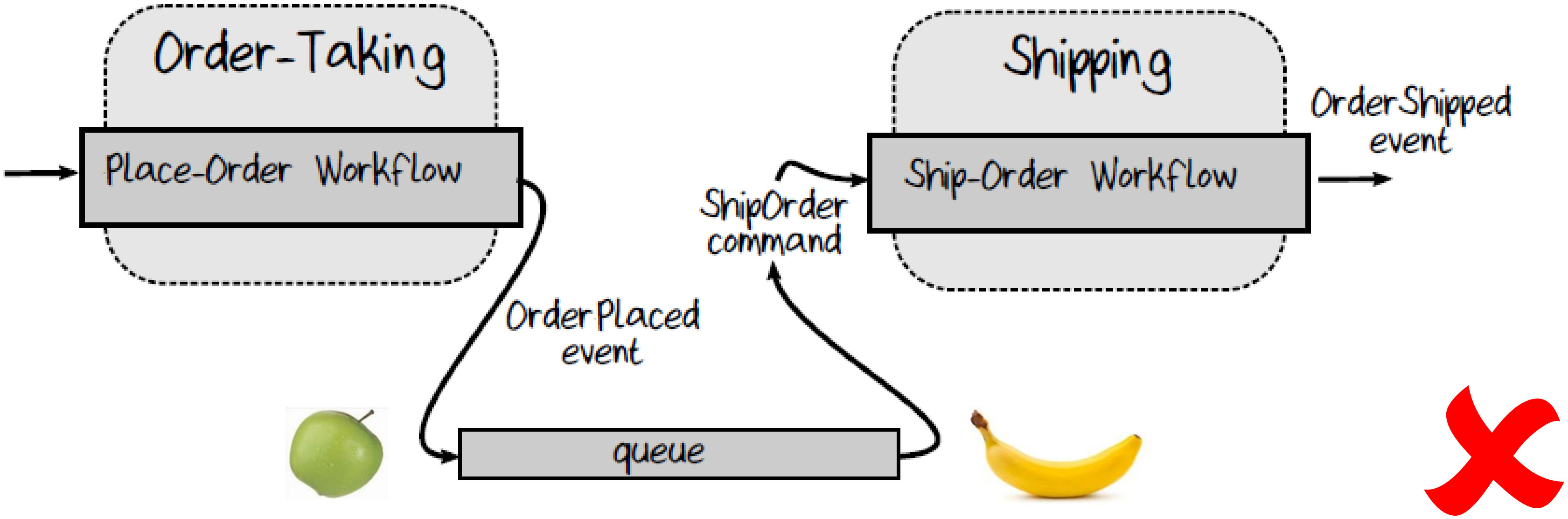
DTOs are contracts
between bounded contexts

DTOs are contracts



✓

DTOs are contracts



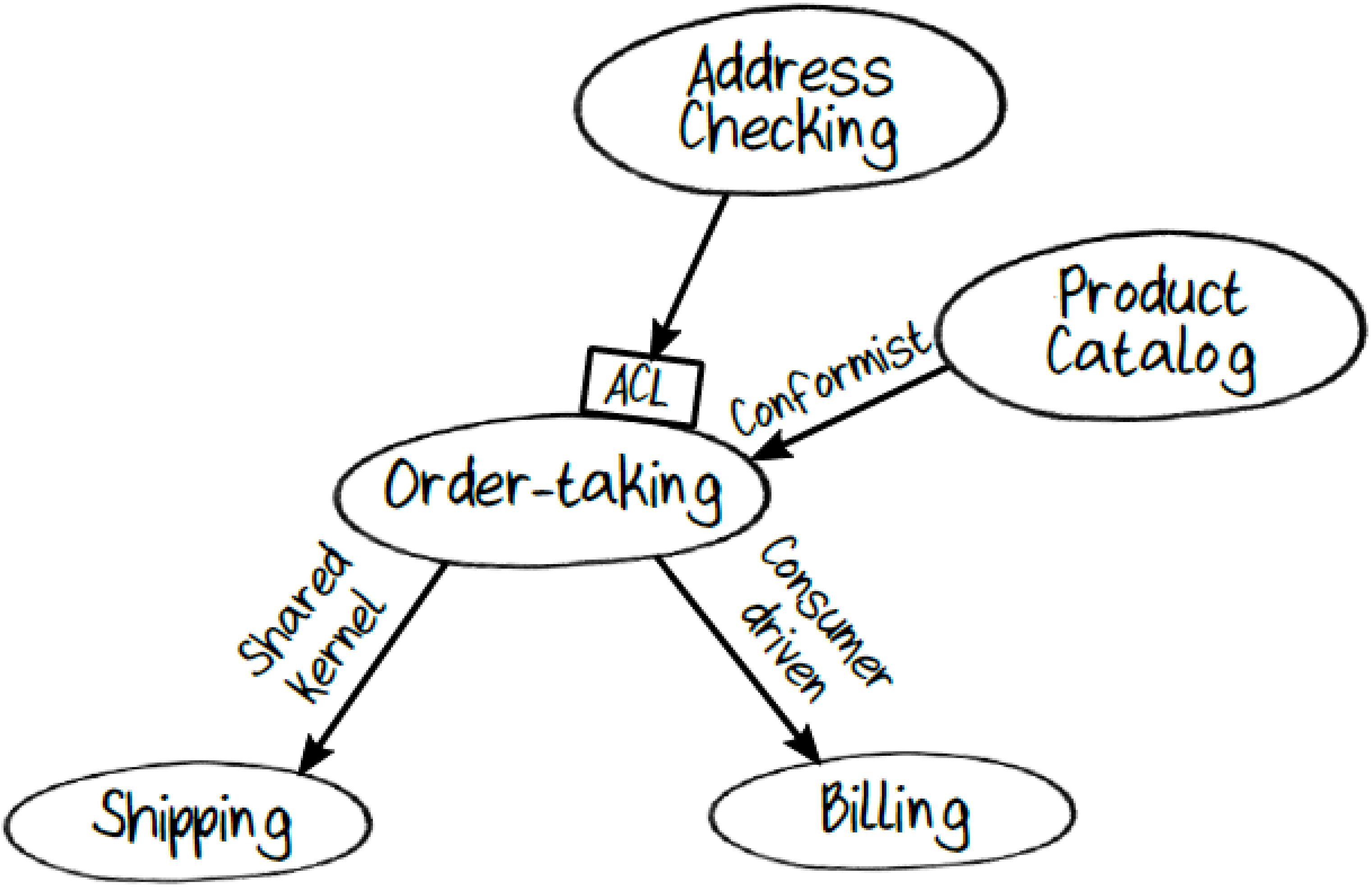
Relationships between contexts

- *Shared Kernel*
  - Two contexts share some common domain design, so the teams involved must collaborate.
- *Consumer Driven*
  - The downstream context defines the contract
- *Conformist*
  - The downstream context accepts the contract provided by the upstream context

Anti-Corruption Layer

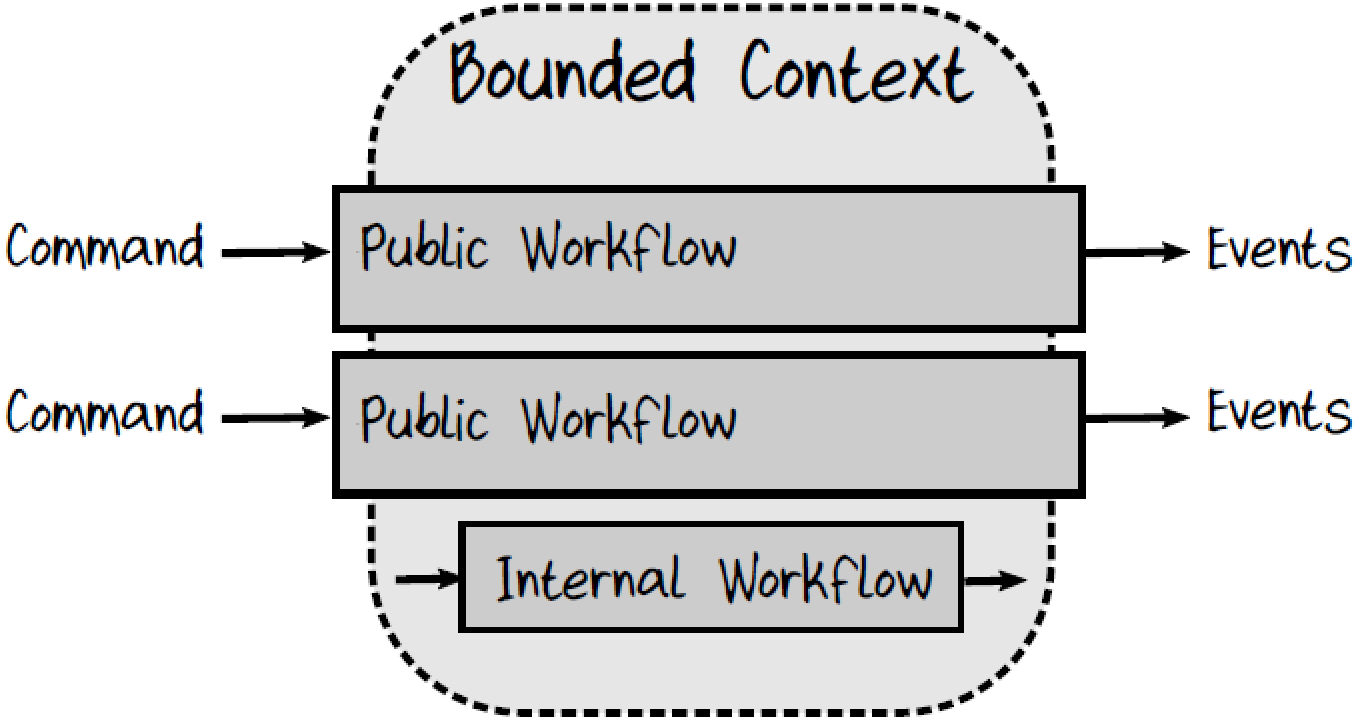
- Acts as a translator between two different languages
  - the language used in the upstream context
  - The language used in the downstream context

Example of different contracts

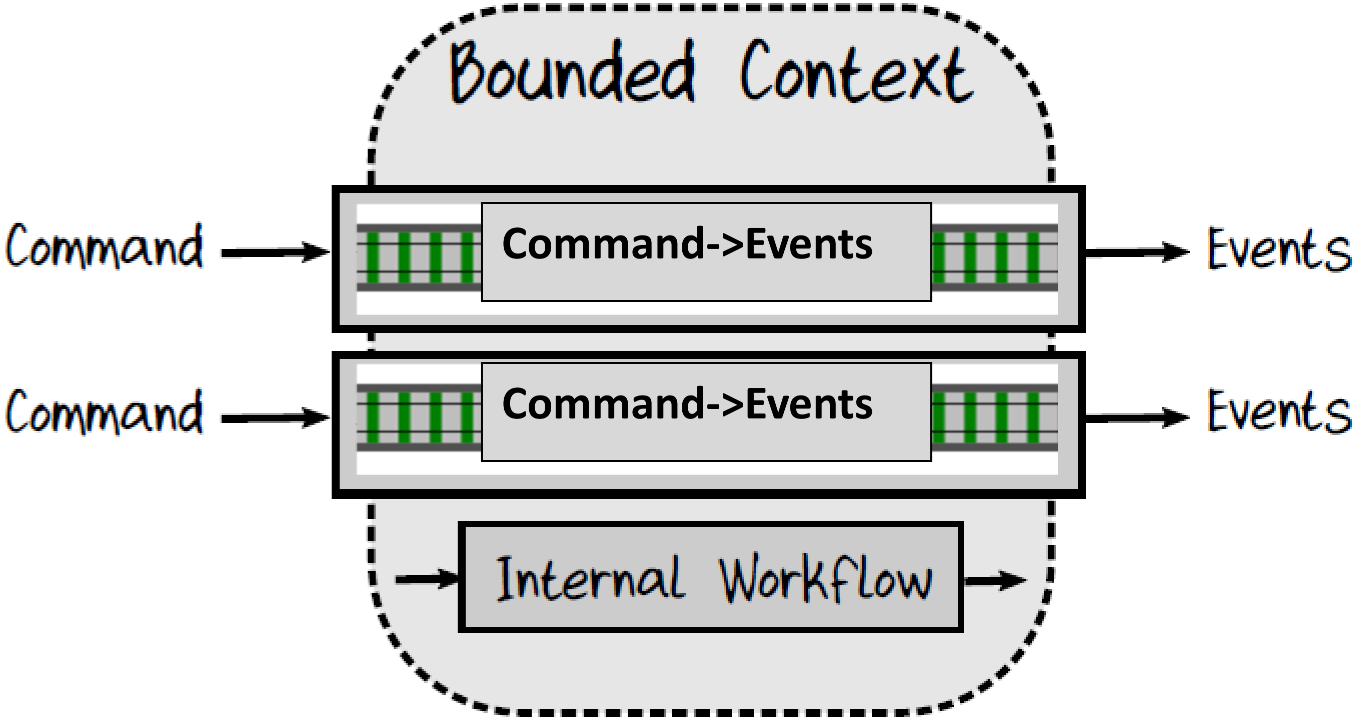


Workflows within bounded contexts

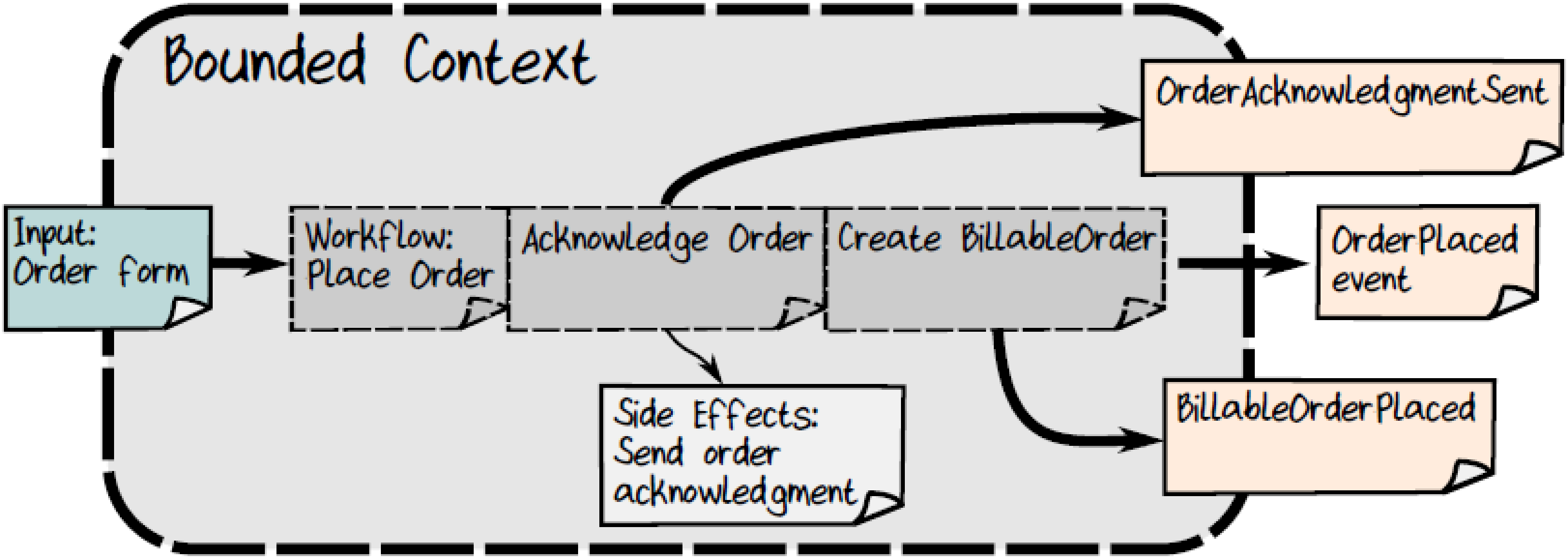
Workflow inputs and outputs



Workflows are functions!

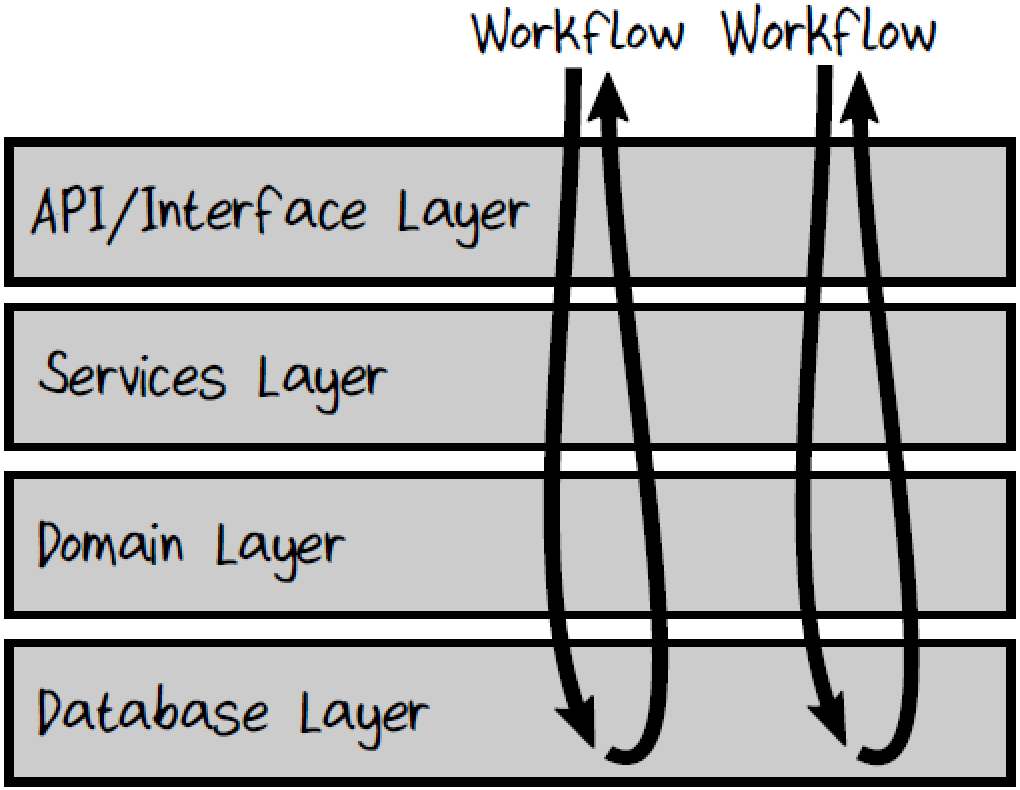


Workflow example



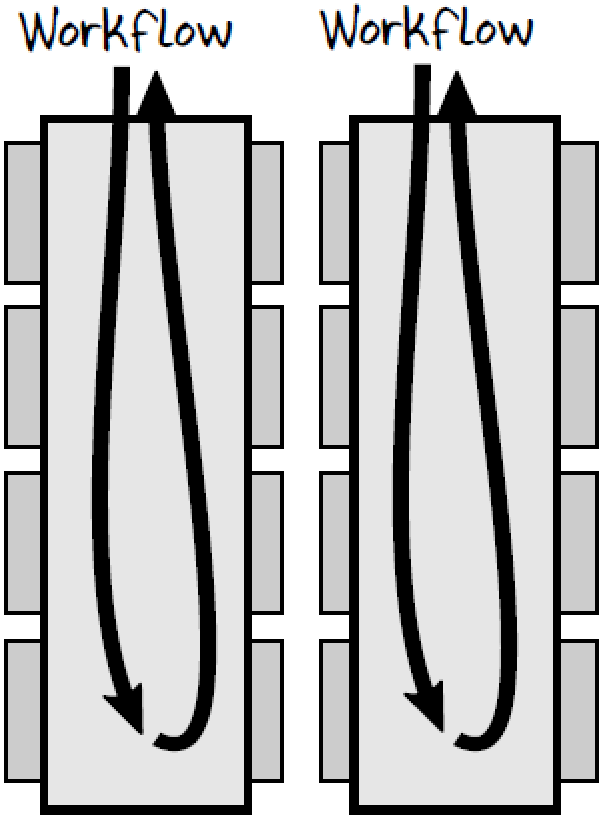
Implementing a workflow functionally

Traditional layered model



A change to the way that a workflow works means that you need to touch every layer.

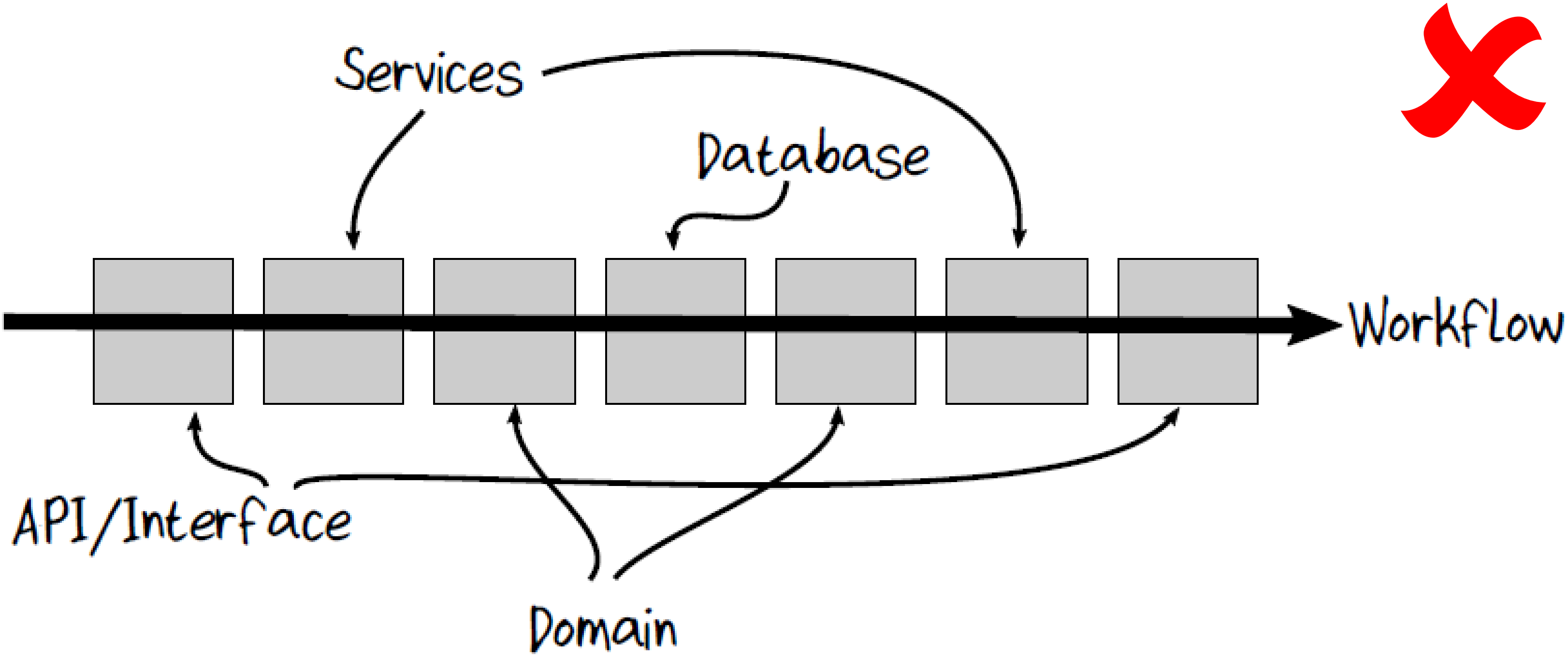
Vertical slices



Each workflow contains all the code it needs to get its job done. When the requirements change for a workflow, only the code in that particular vertical slice needs to change.

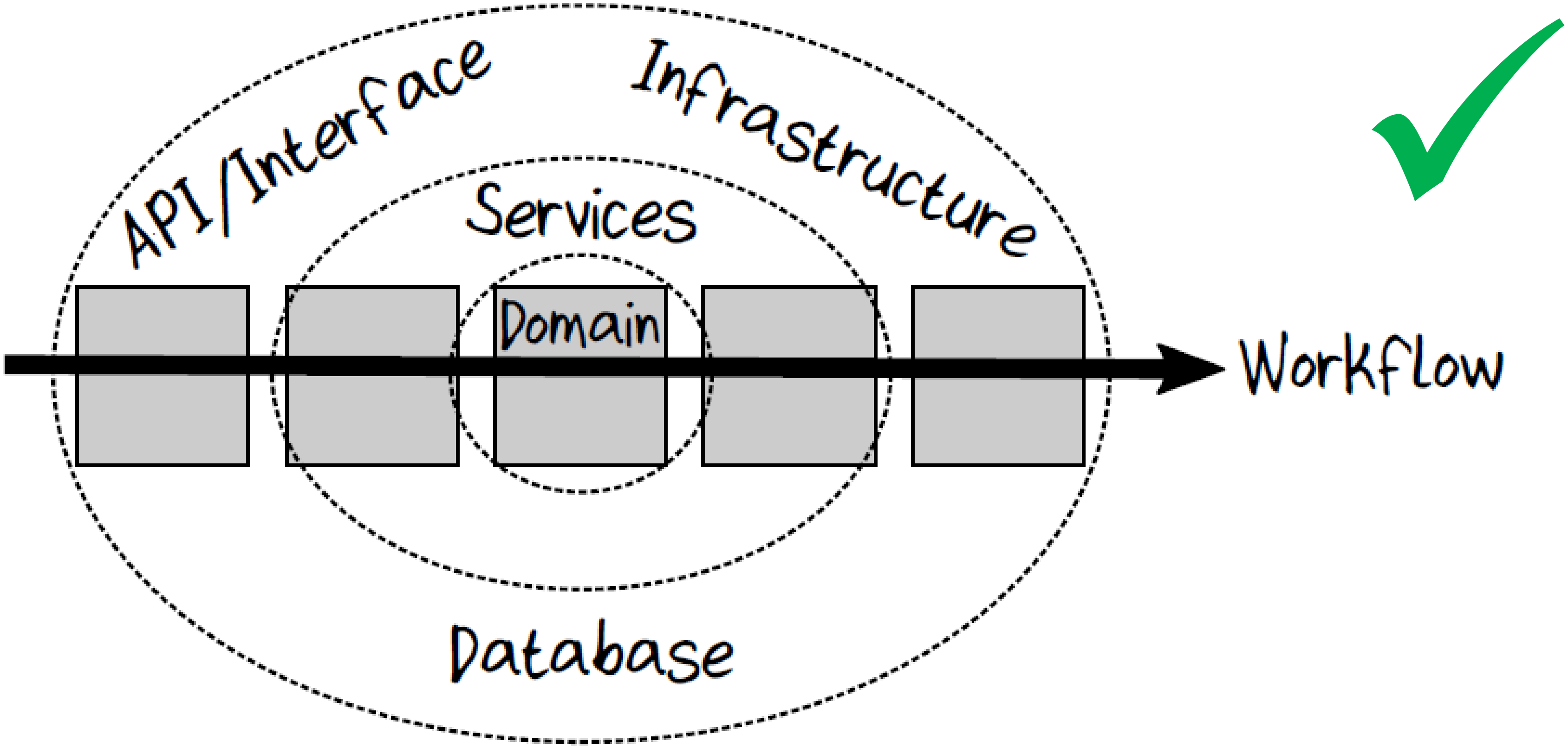
Vertical slices stretched out

✗



Confusing!

The "onion" architecture



Core domain is pure, and all I/O is at the edges

End