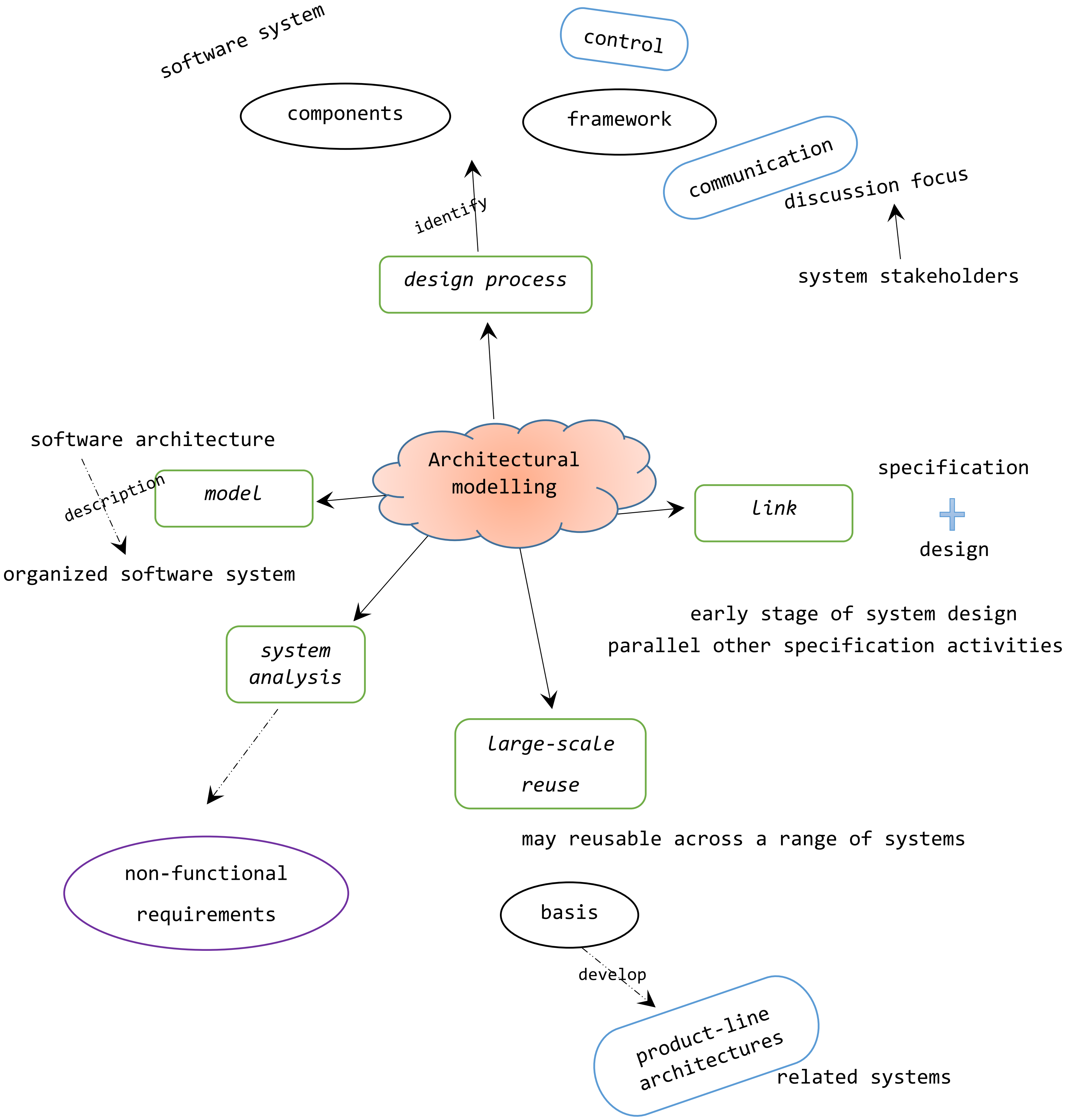


software system

components

control

framework

communication

discussion focus

identify

system stakeholders

design process

software architecture

description

model

Architectural modelling

link

specification

design

organized software system

early stage of system design

parallel other specification activities

system analysis

large-scale reuse

may reusable across a range of systems

non-functional requirements

basis

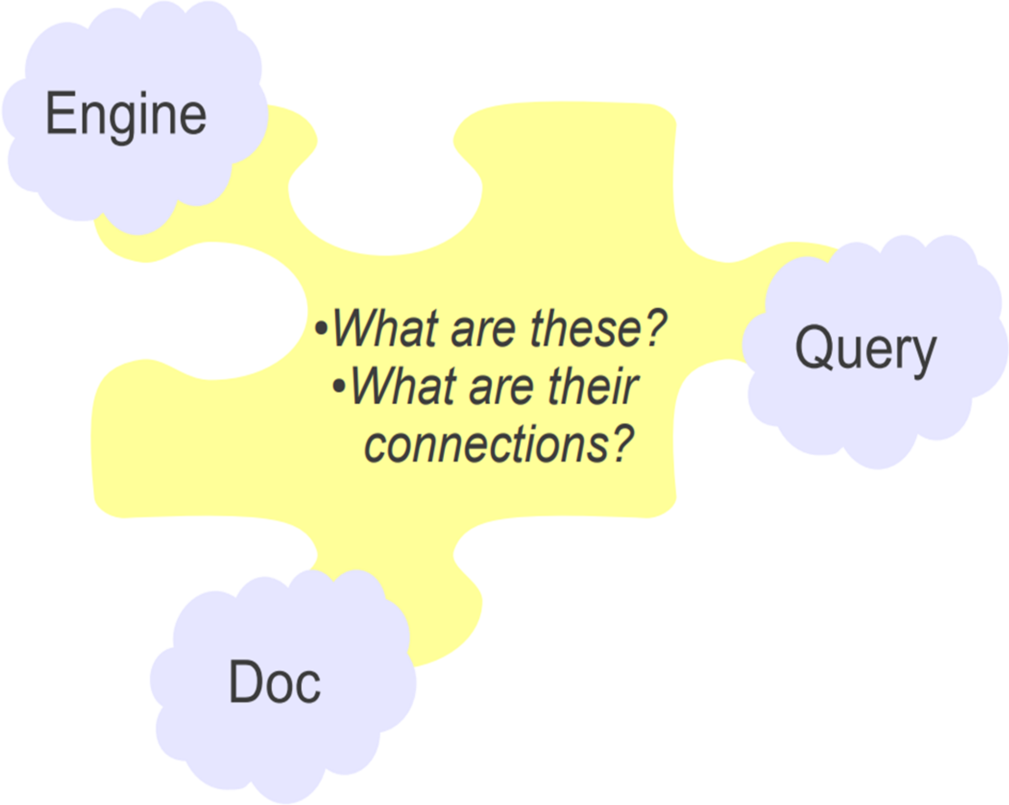
develop

product-line architectures

related systems

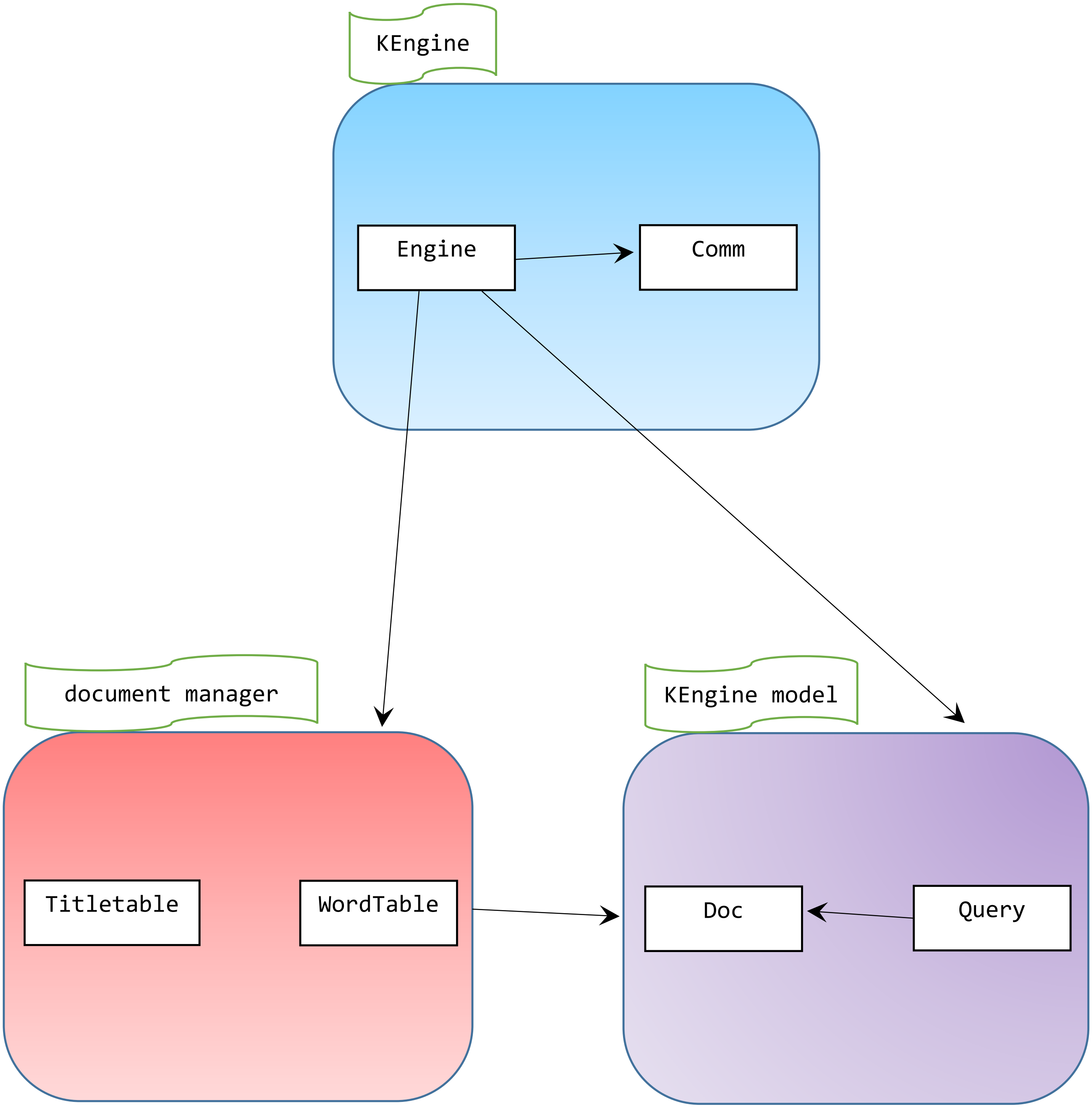
Example: KEngine

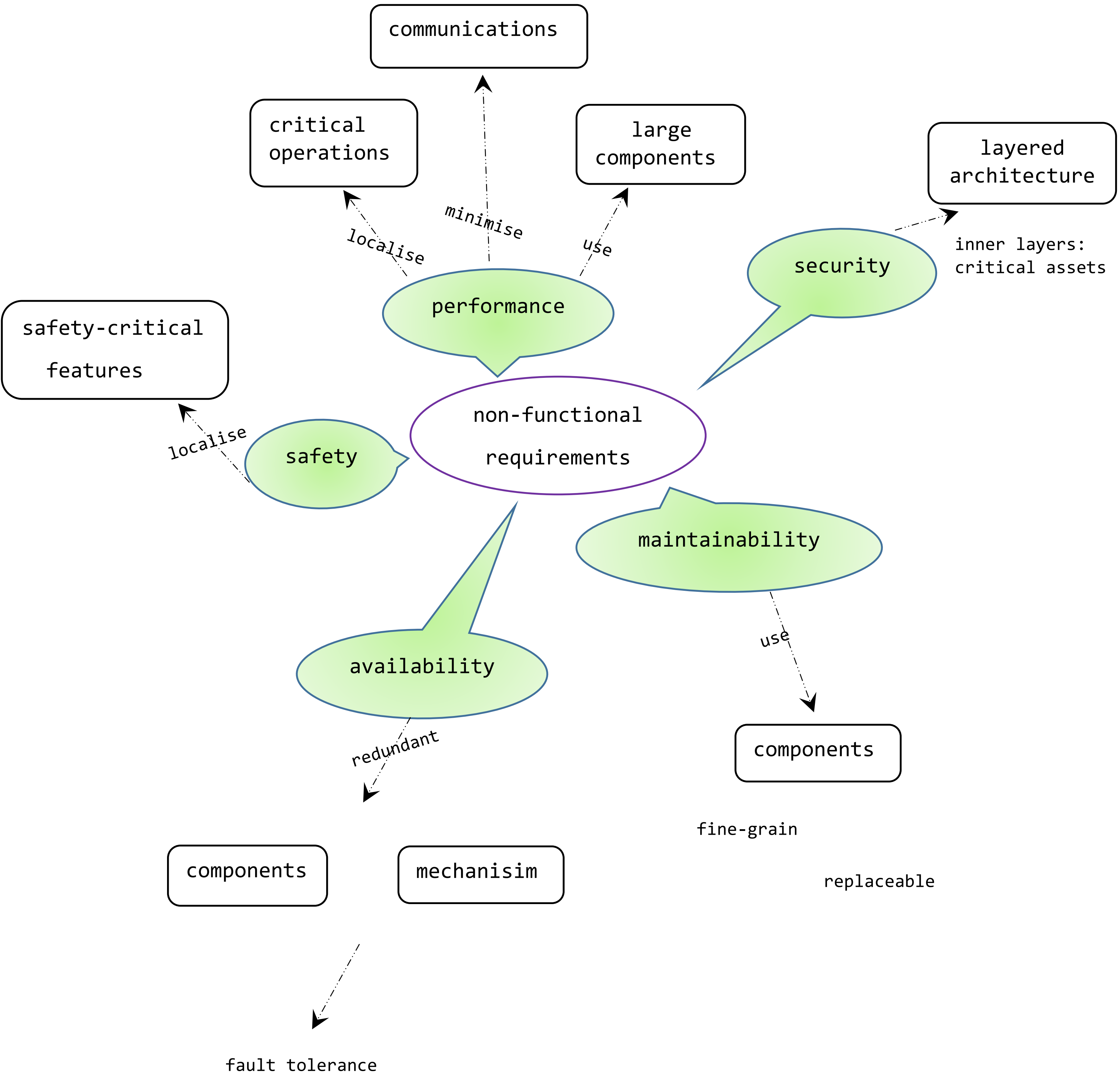
◊ KEngine architectural concerns:

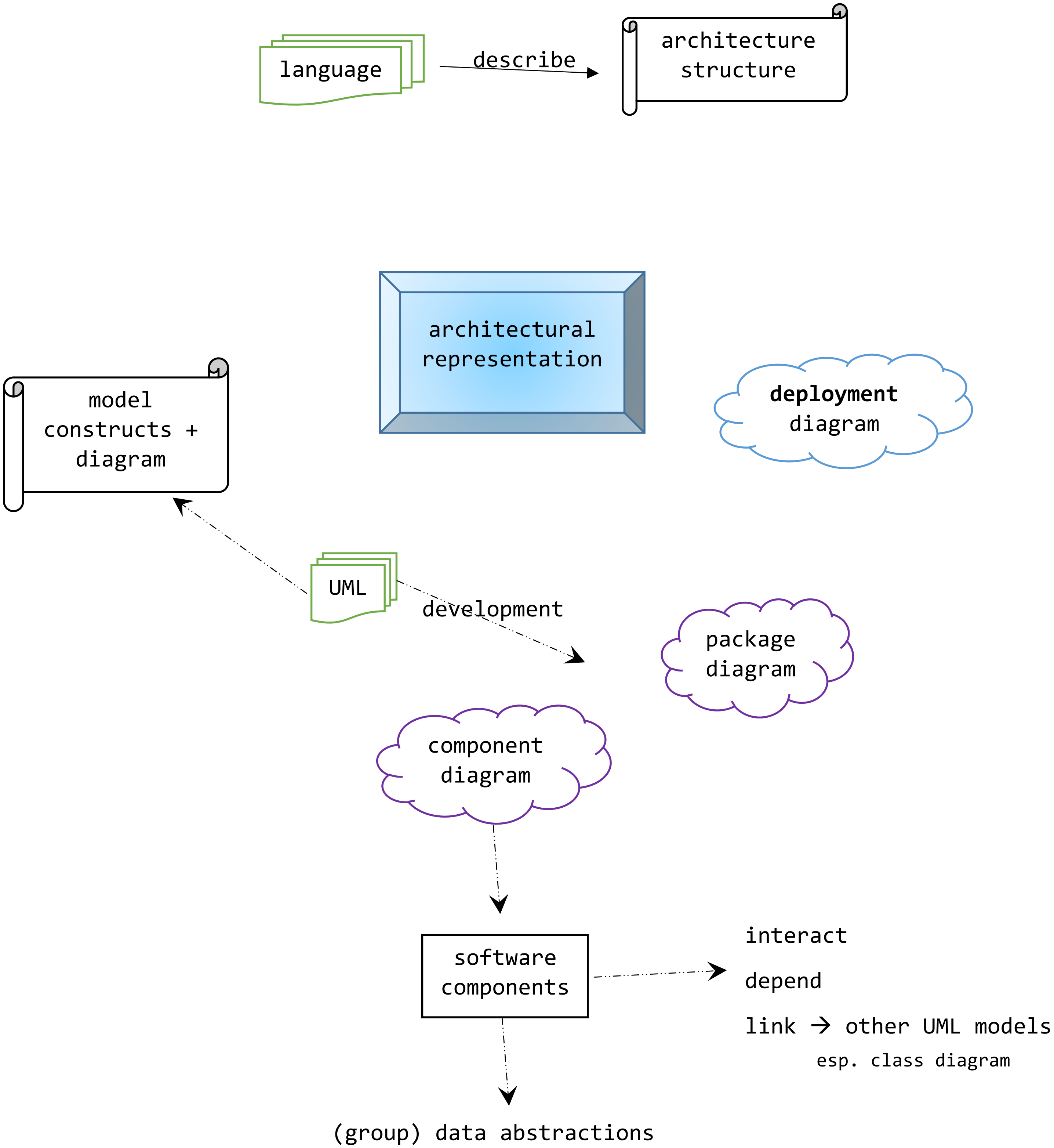


- *What are these?*
- *What are their connections?*

KEngine architecture




communications
critical operations
large components
layered architecture
minimise
localise
use
inner layers: critical assets
security
performance
safety-critical features
localise
safety
non-functional requirements
maintainability
availability
use
redundant
components
components
mechanisim
fine-grain
replaceable
fault tolerance



language

describe

architecture structure

architectural representation

model constructs + diagram

deployment diagram

UML

development

package diagram

component diagram

software components

interact

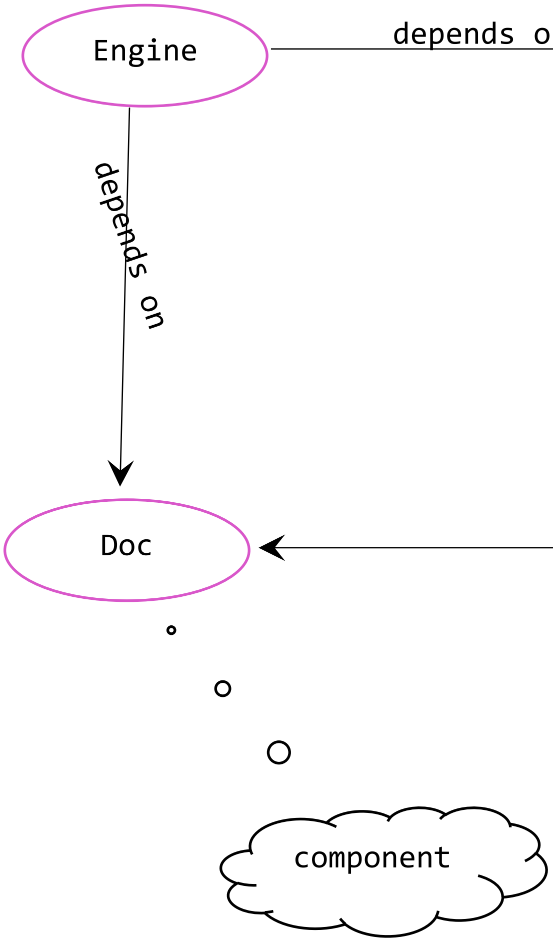
depend

link → other UML models

esp. class diagram

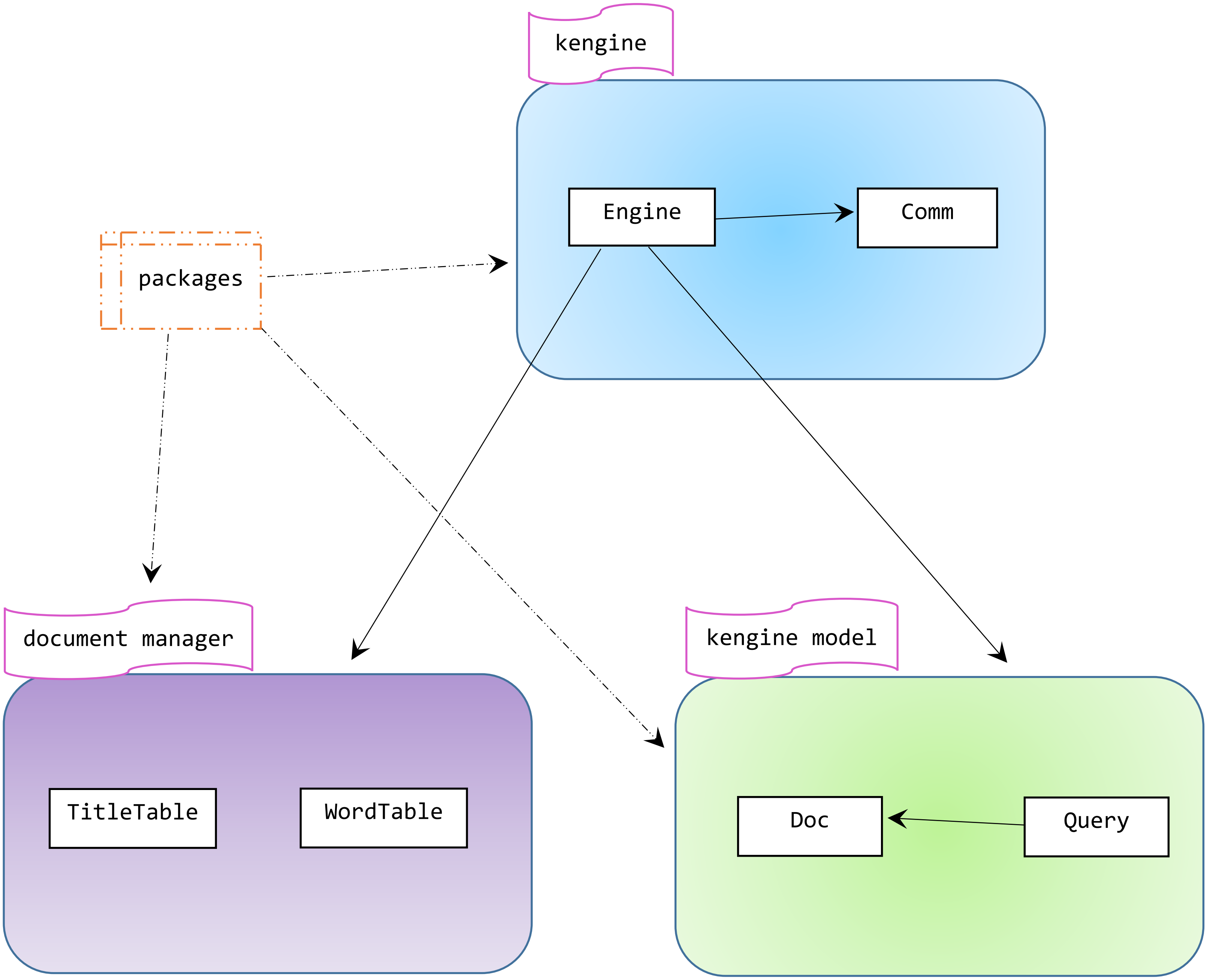
(group) data abstractions

Basic KEngine architecture



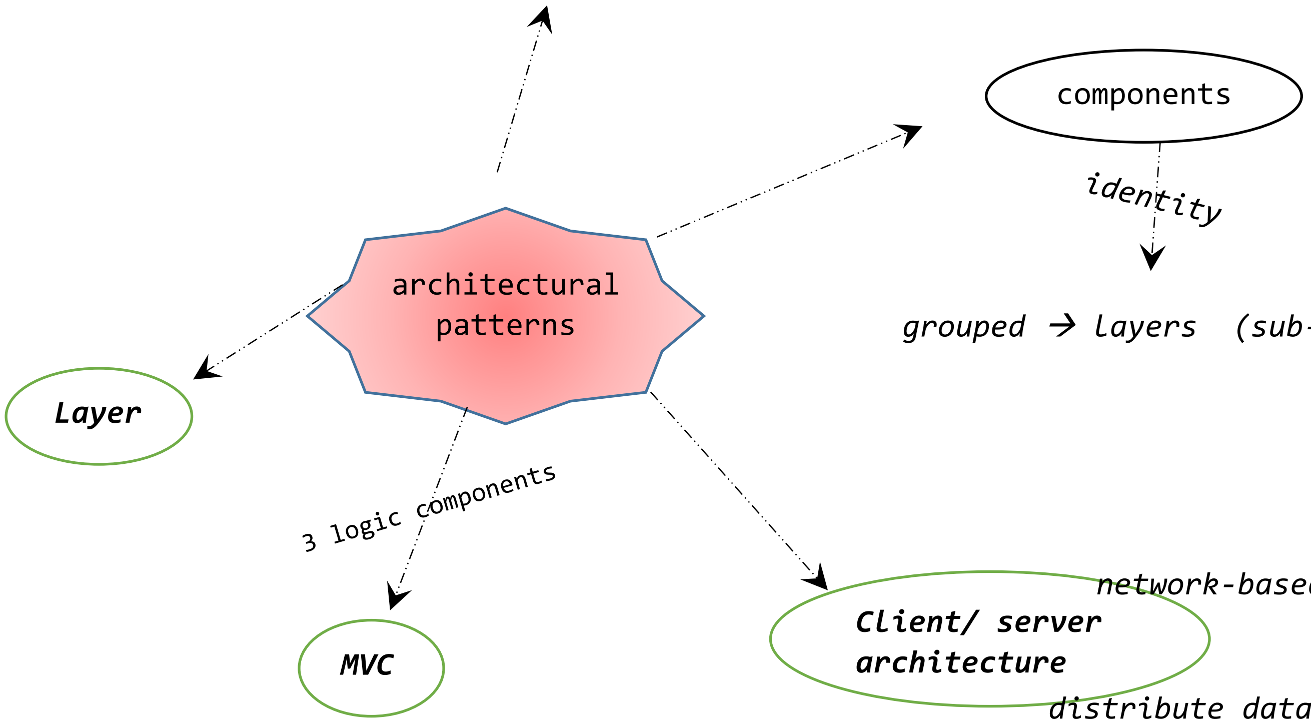
Detailed KEngine architecture

KEngine architectural model → show some components → packages



stylized description → good design practice

means → reusing knowledge → generic system architectures



architectural patterns

components

identity

grouped → layers (sub-systems)

Layer

3 logic components

MVC

Client/ server architecture

network-based system

distribute data + processing

***Model:** manage system data + operation*

***View:** manage data presentation → user*

***Controller:** manage users interactions*

via keyboard, mouse, …

multiple ways → interact data

unknown future requirements

data change ←→ representation

support dif pre. – same data

code complexity

***Server:** stand-alone process*

➔ specific service

printing, data management, …

***Client:** process ← servers services*

***Network:** clients ←→ servers*

data shared - range of app. / processes

load sharing → particular service

database look-up

distribute server – network

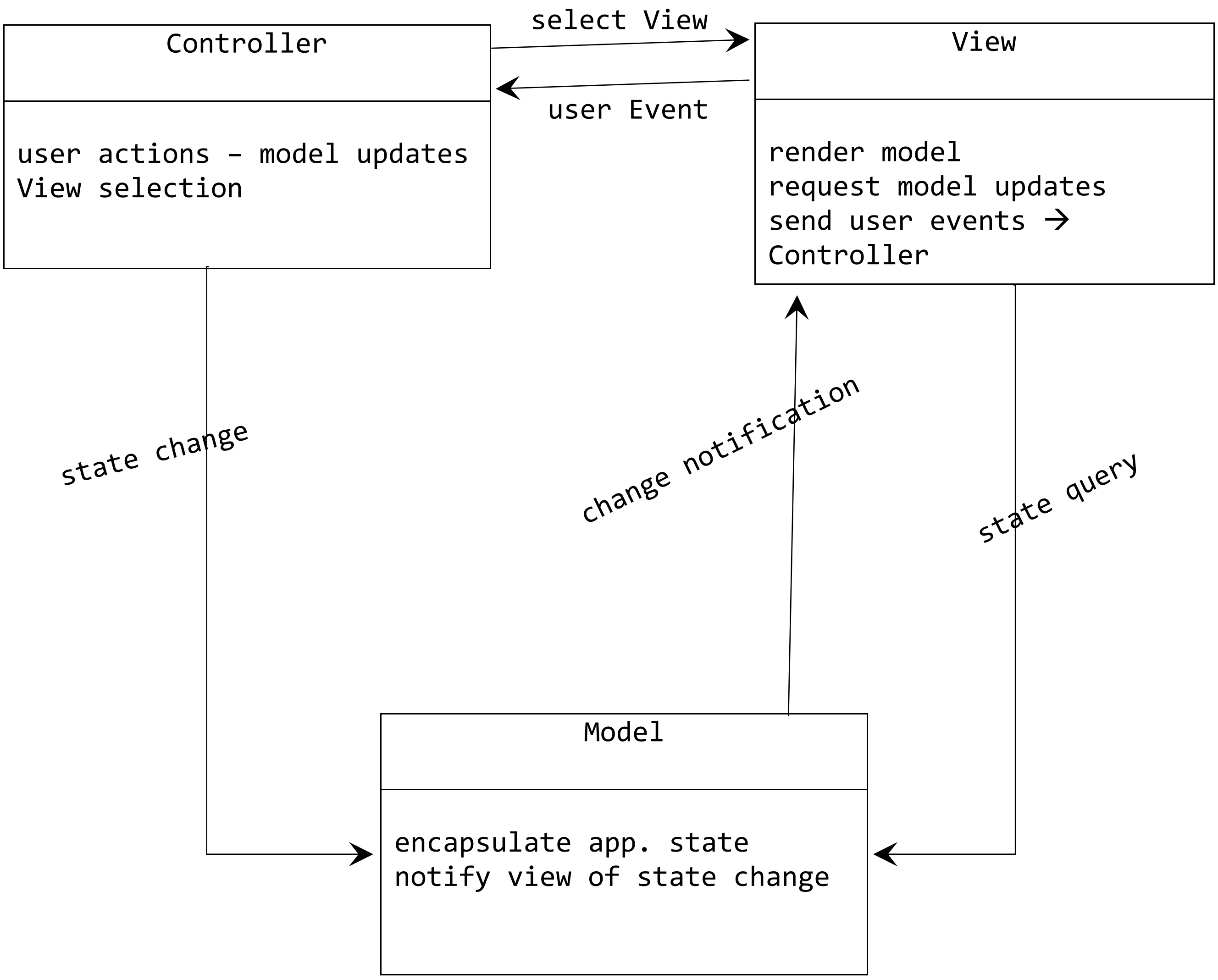
available shared functionality

server: failure point

unpredictable performance

increased system management overhead

MVC diagram



Controller

user actions - model updates
View selection

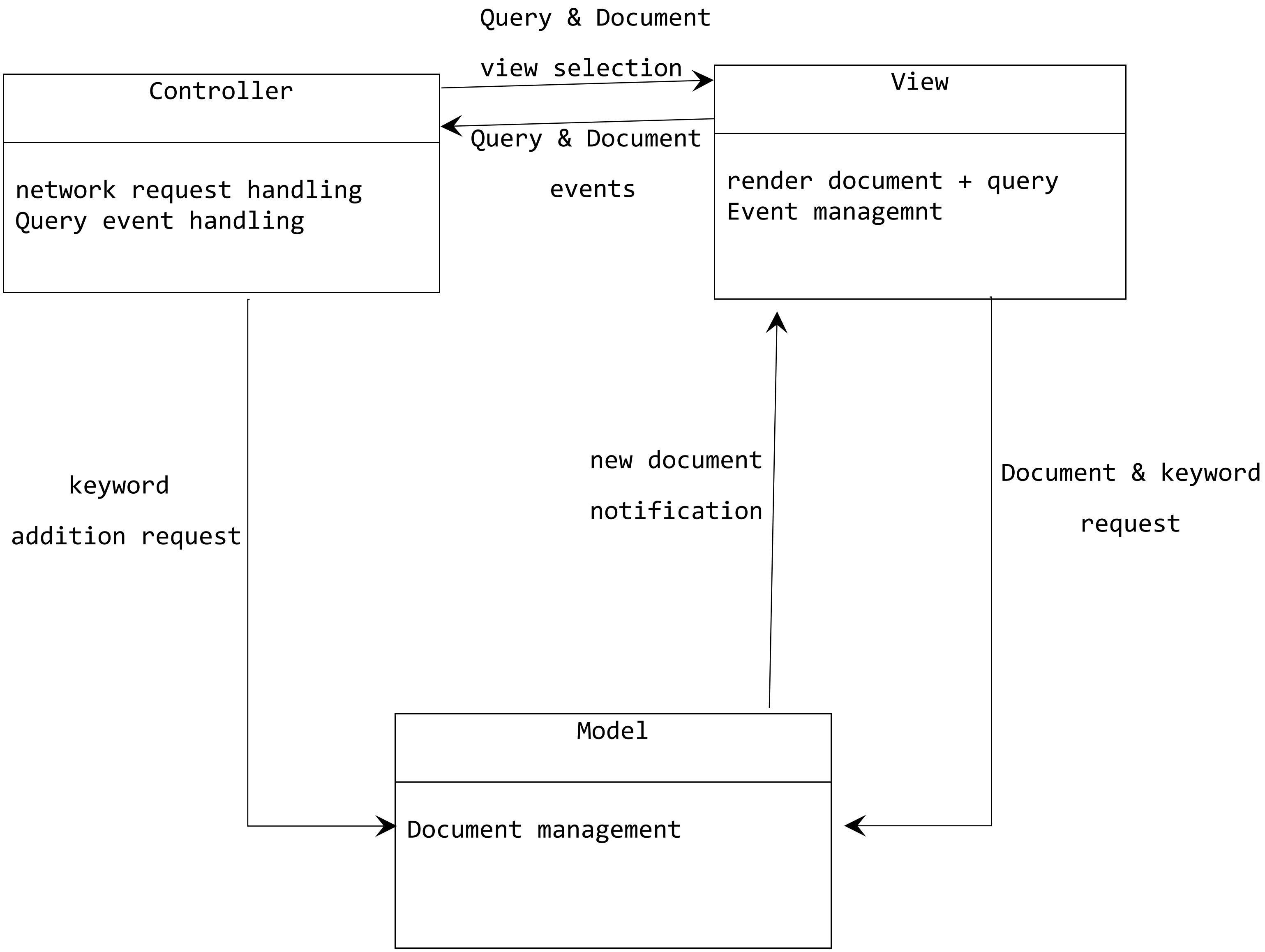
View

render model
request model updates
send user events → Controller

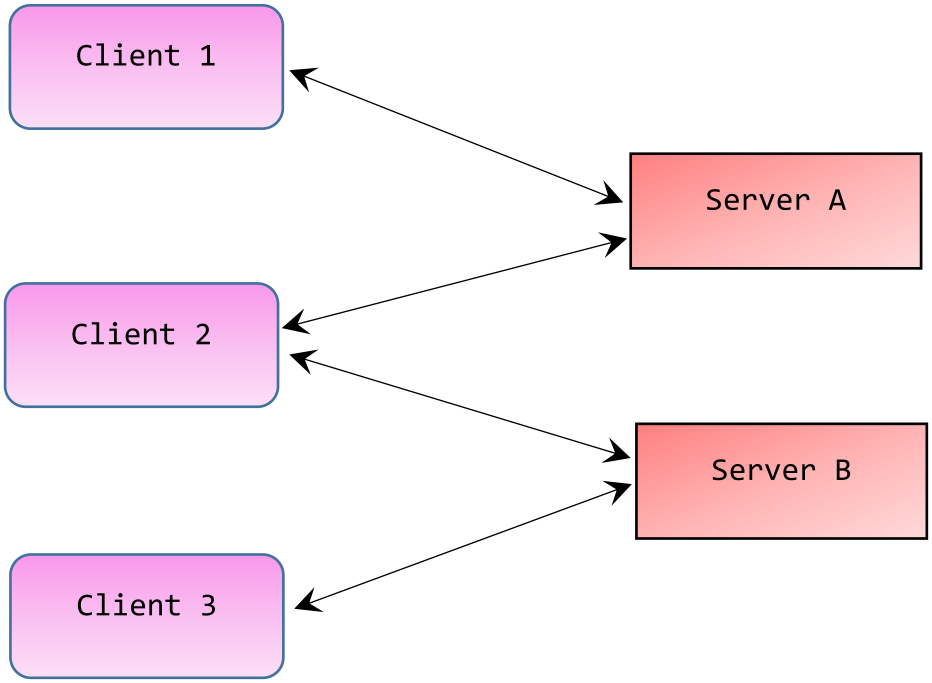
Model

encapsulate app. state
notify view of state change

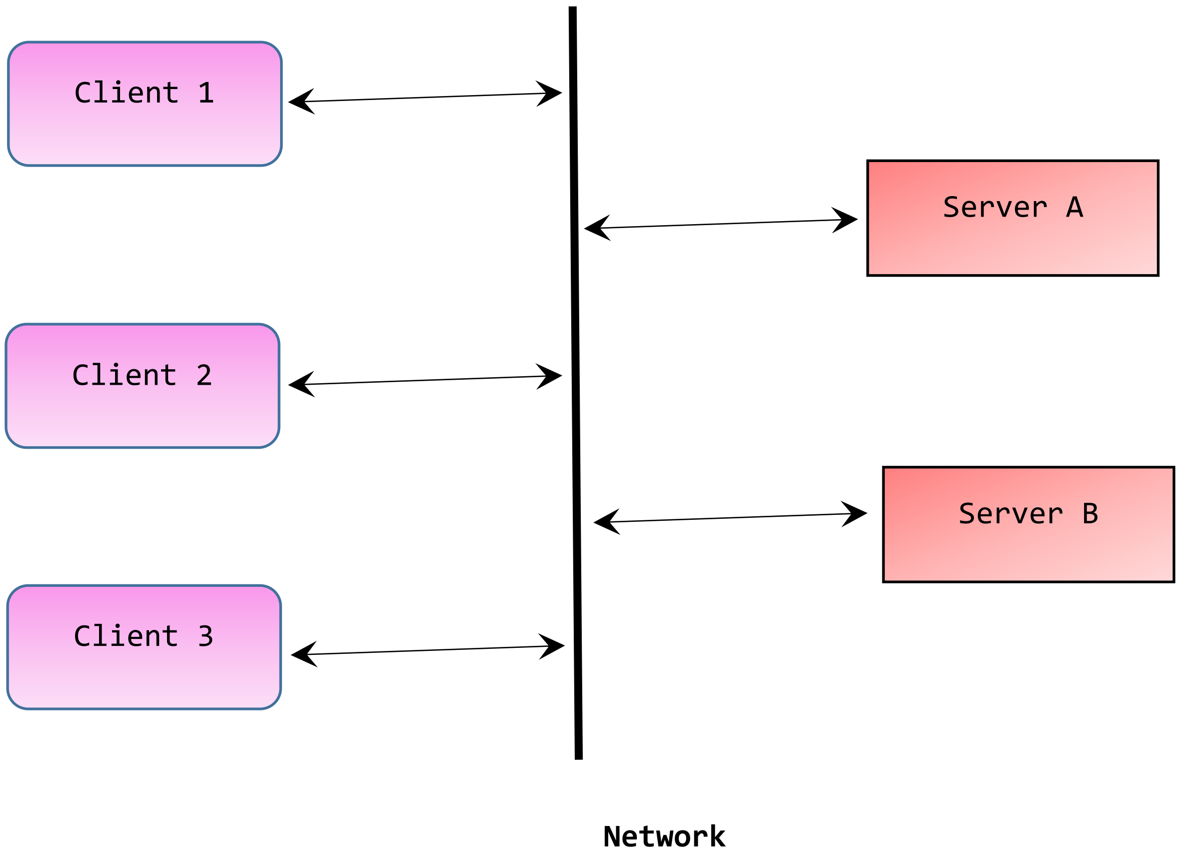
KEngine


Controller
network request handling
Query event handling
View
render document + query
Event managemnt
Model
Document management
Query & Document
view selection
Query & Document
events
keyword
addition request
new document
notification
Document & keyword
request

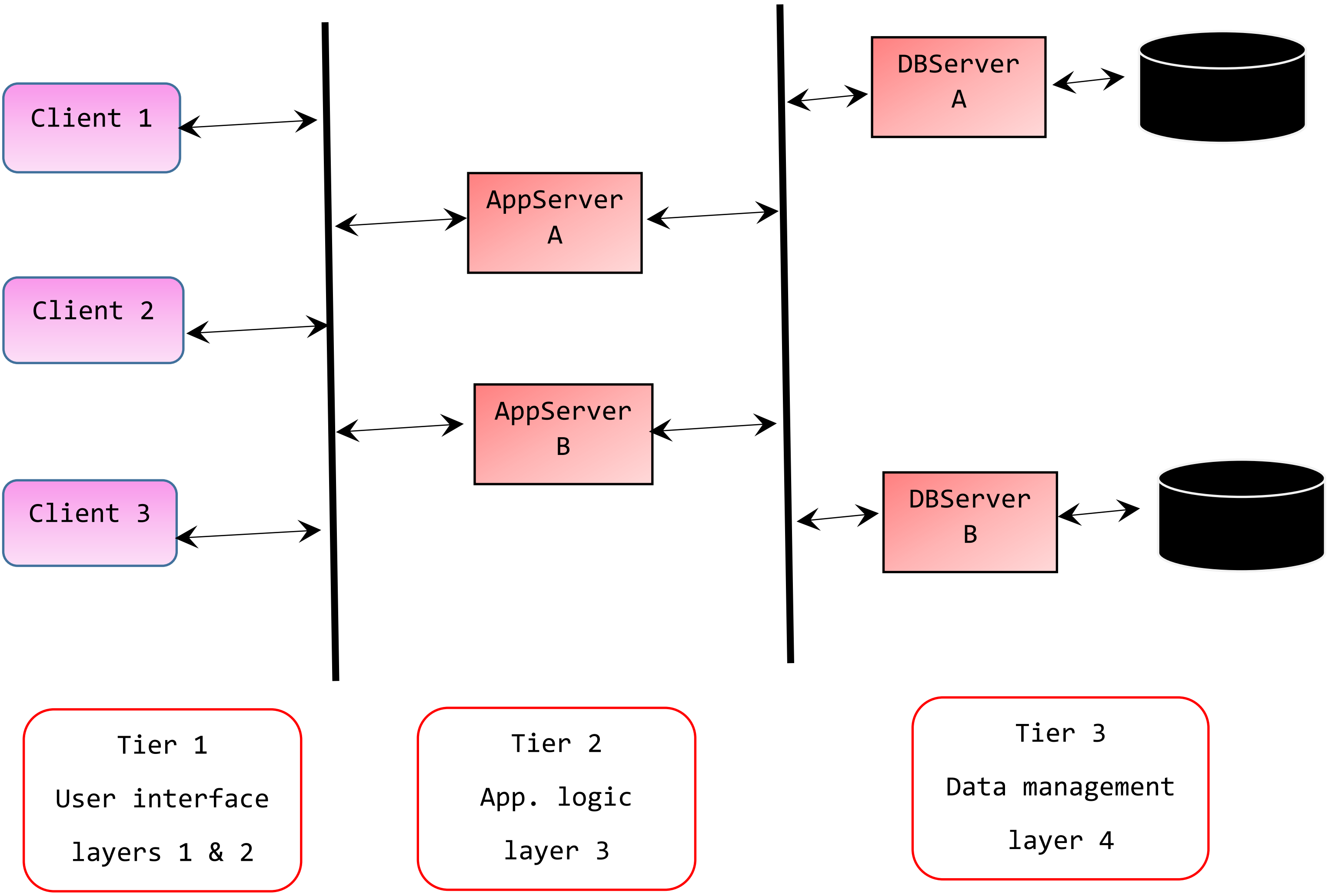
Client/ server architecture diagram



Bus style network



3-tier architecture



Client 1

Client 2

Client 3

AppServer A

AppServer B

DBServer A

DBServer B

Tier 1

User interface

layers 1 & 2

Tier 2

App. logic

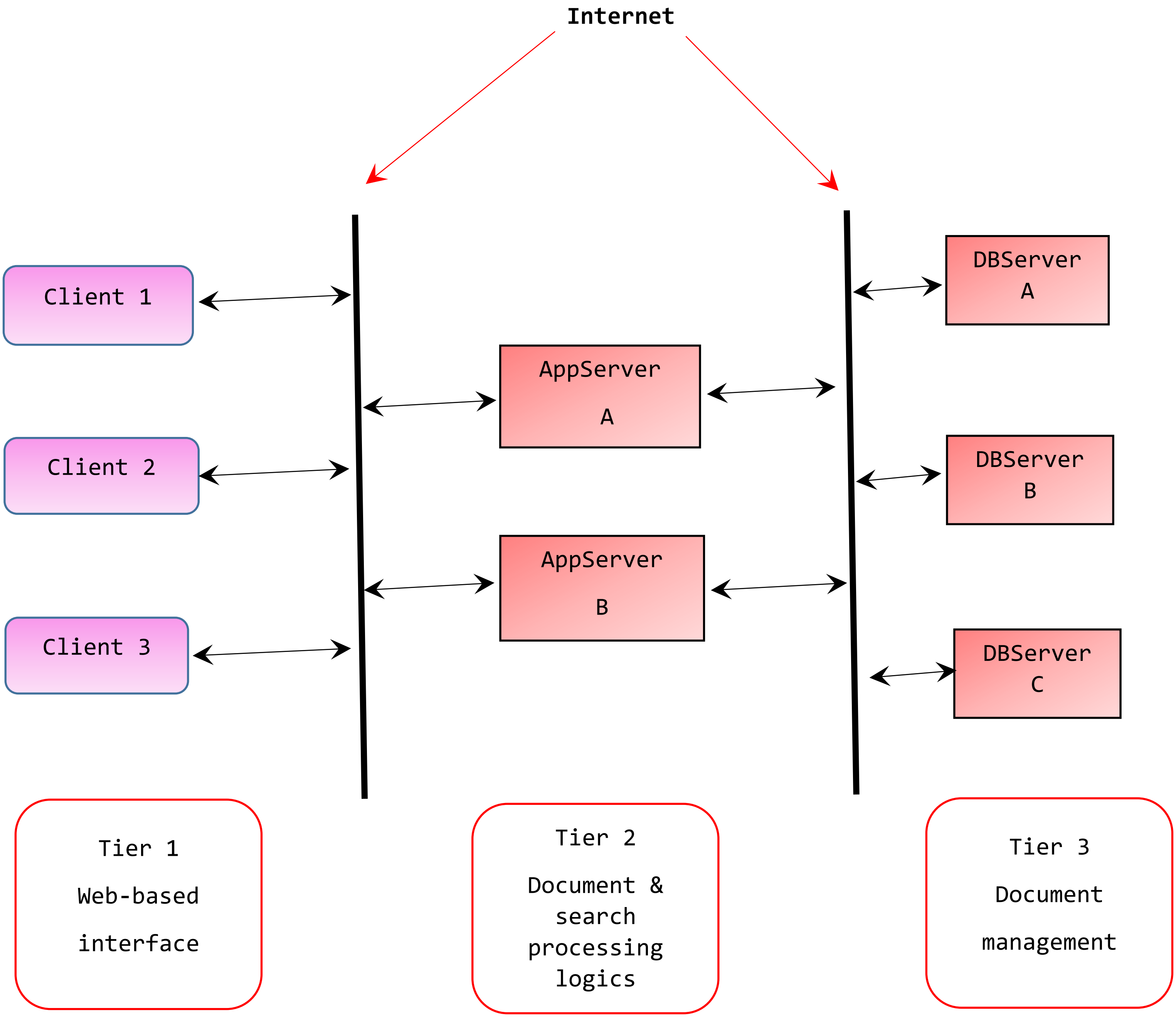
layer 3

Tier 3

Data management

layer 4

KEngine 3-tier architecture



Internet

Client 1

Client 2

Client 3

AppServer A

AppServer B

DBServer A

DBServer B

DBServer C

Tier 1

Web-based interface

Tier 2

Document & search processing logics

Tier 3

Document management