AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

SOFTWARE ENGINEERING PRINCIPLES SOFTWARE ARCHITECTURE

STEFAN HALLESTEDE
PETER GORM LARSEN
CARL SCHULTZ

VERSITET UNI

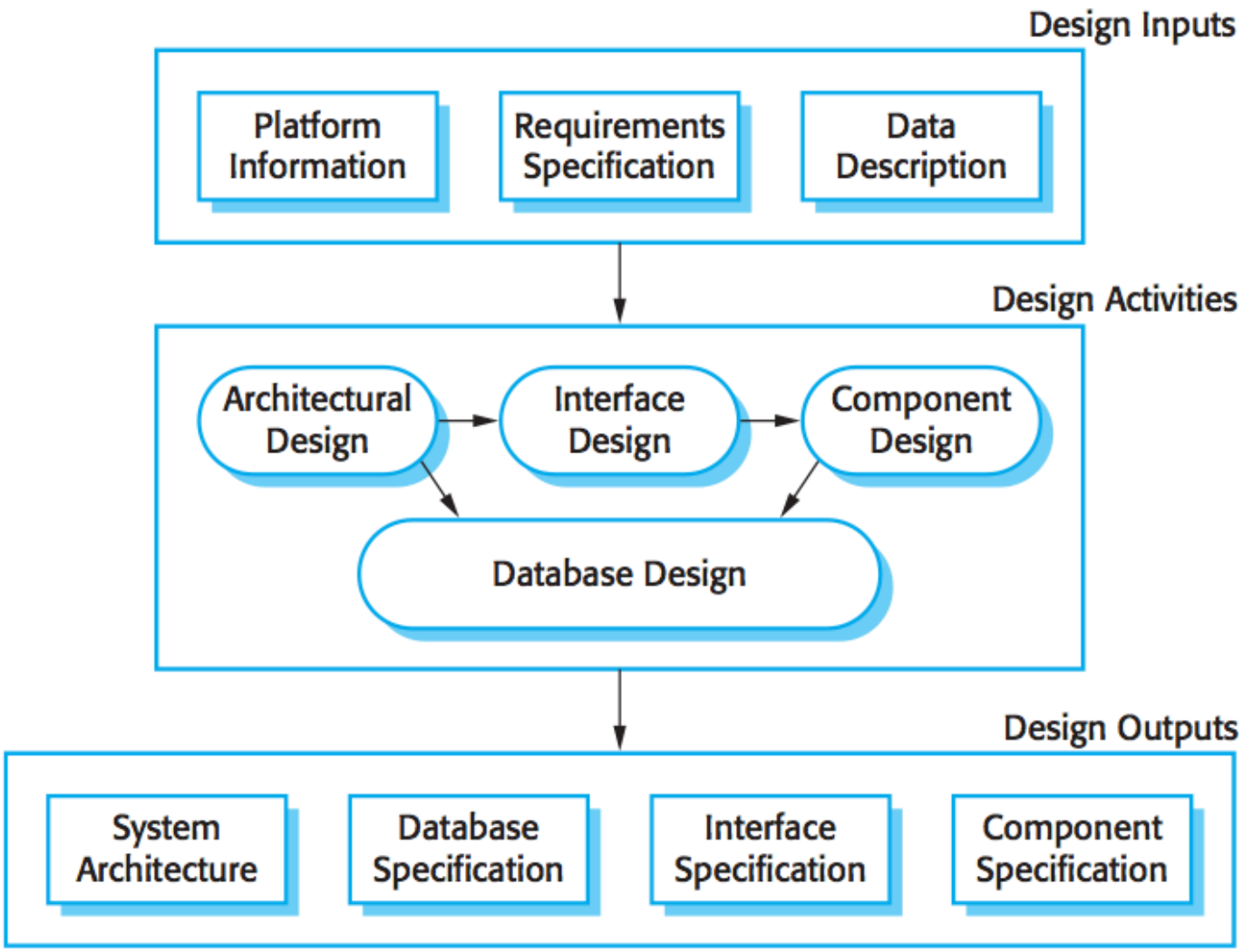


Figure 2.5 A general model of the design process

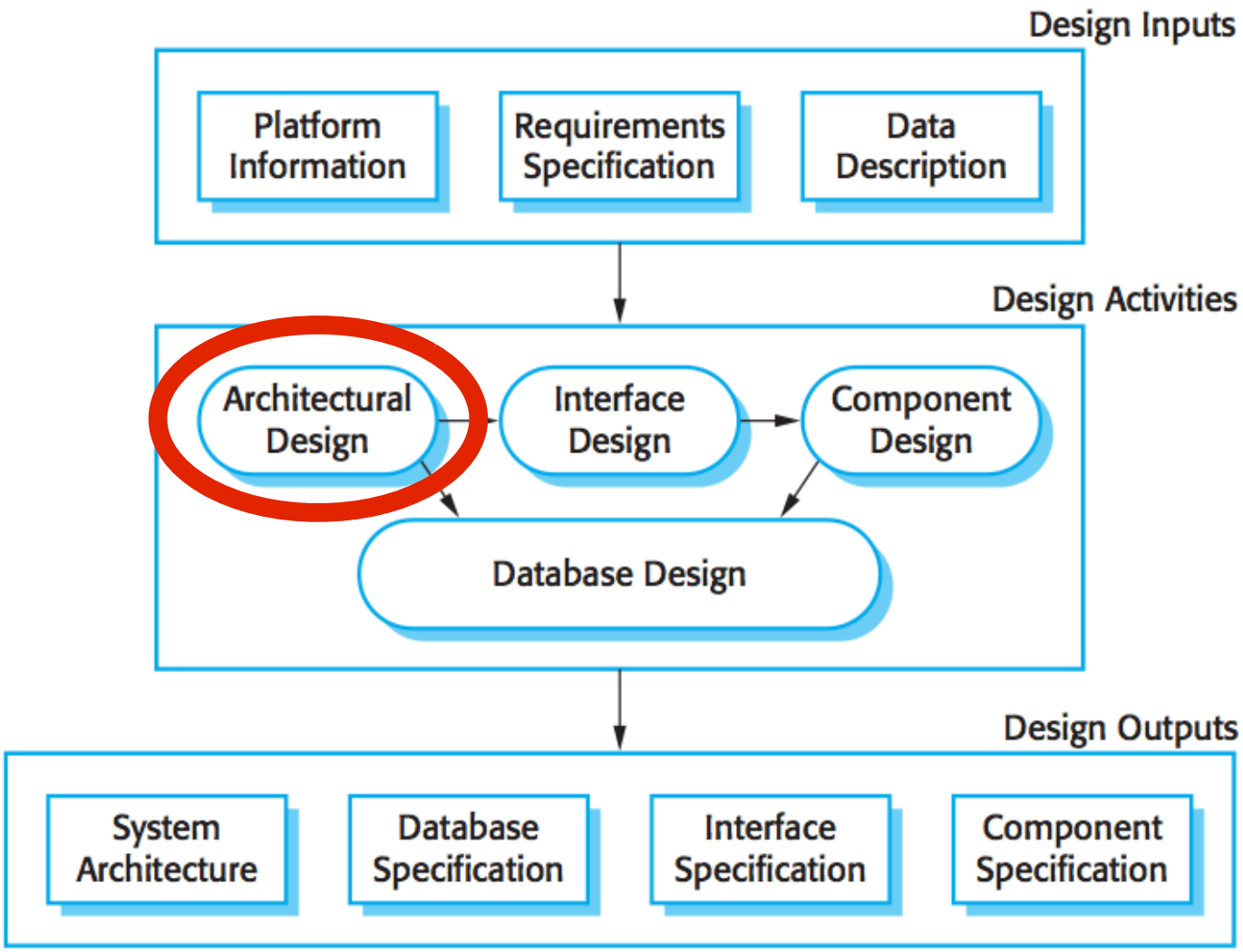
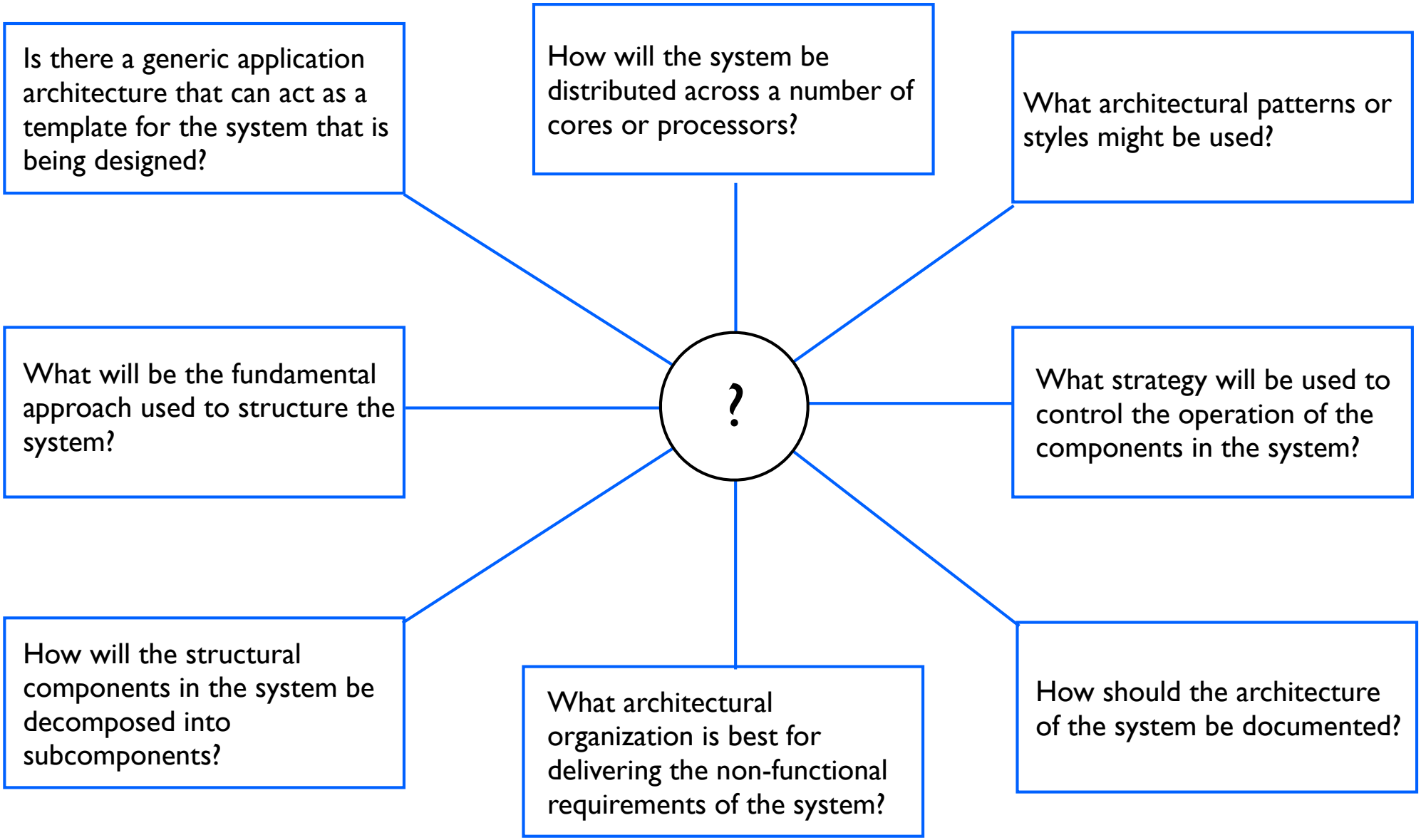


Figure 2.5 A general model of the design process

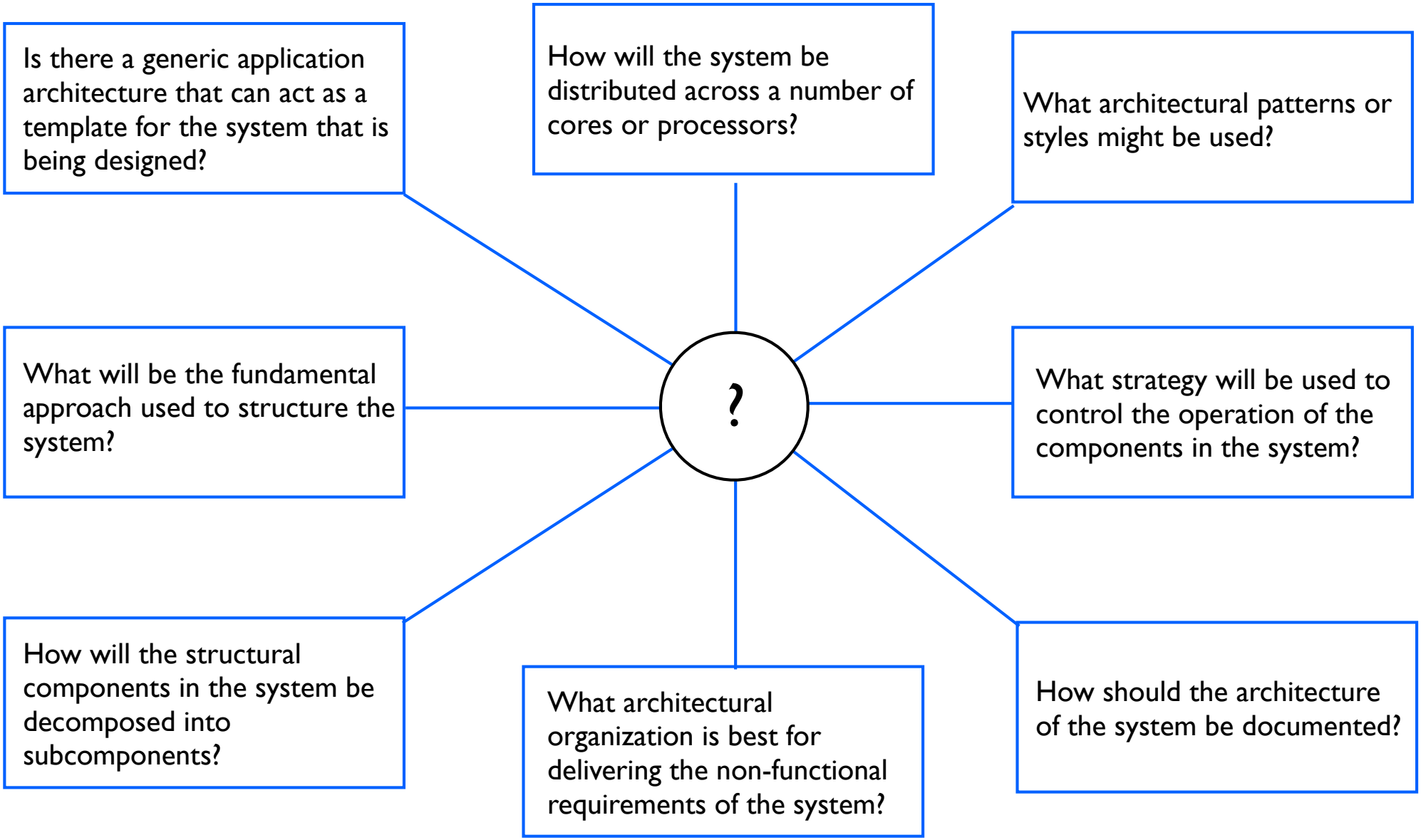
architectural design

- understanding how system should be organised
- designing:
  - overall structure
  - principle components
  - relationships
  - how distributed

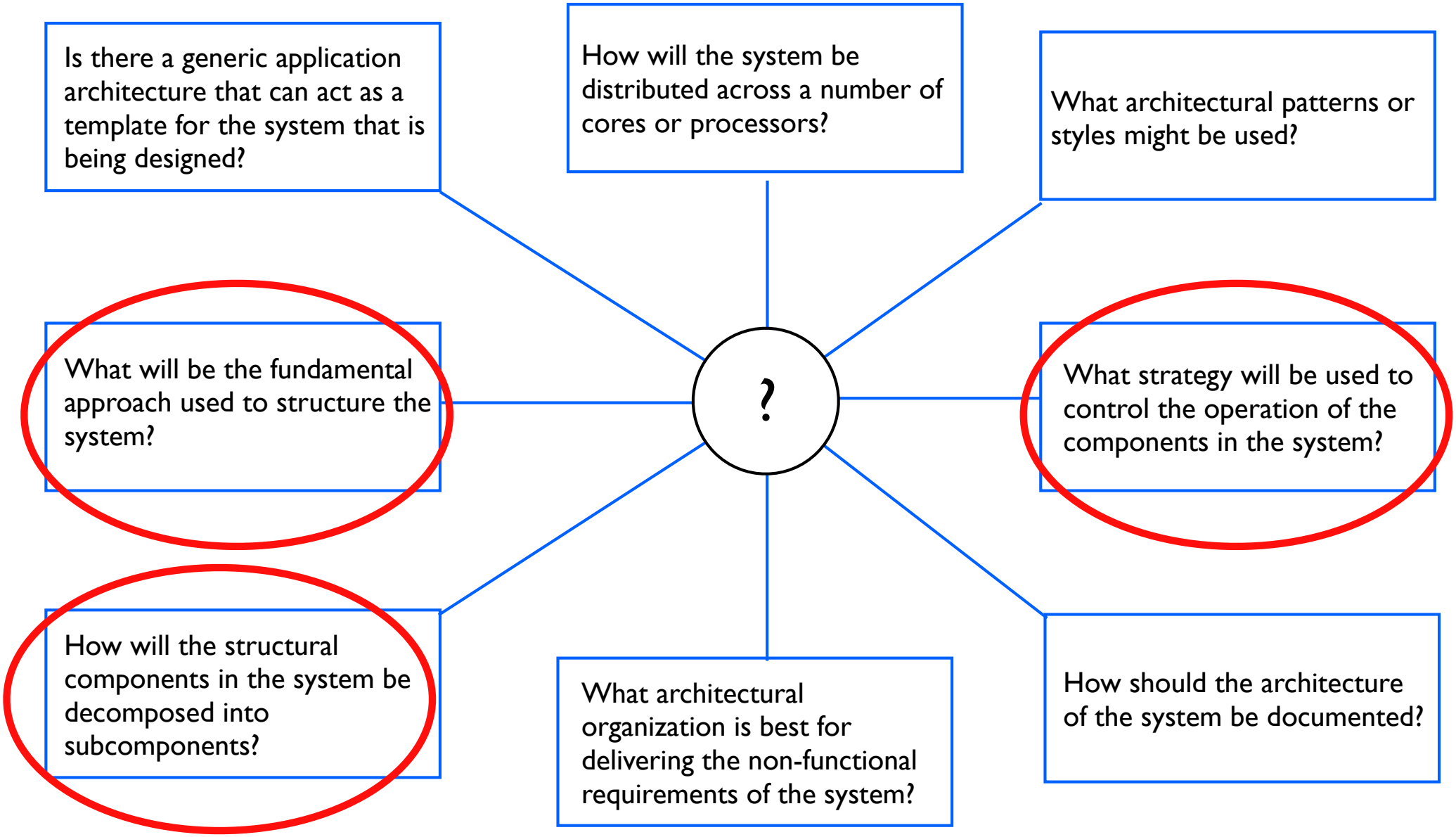
Architectural design vs architectural patterns



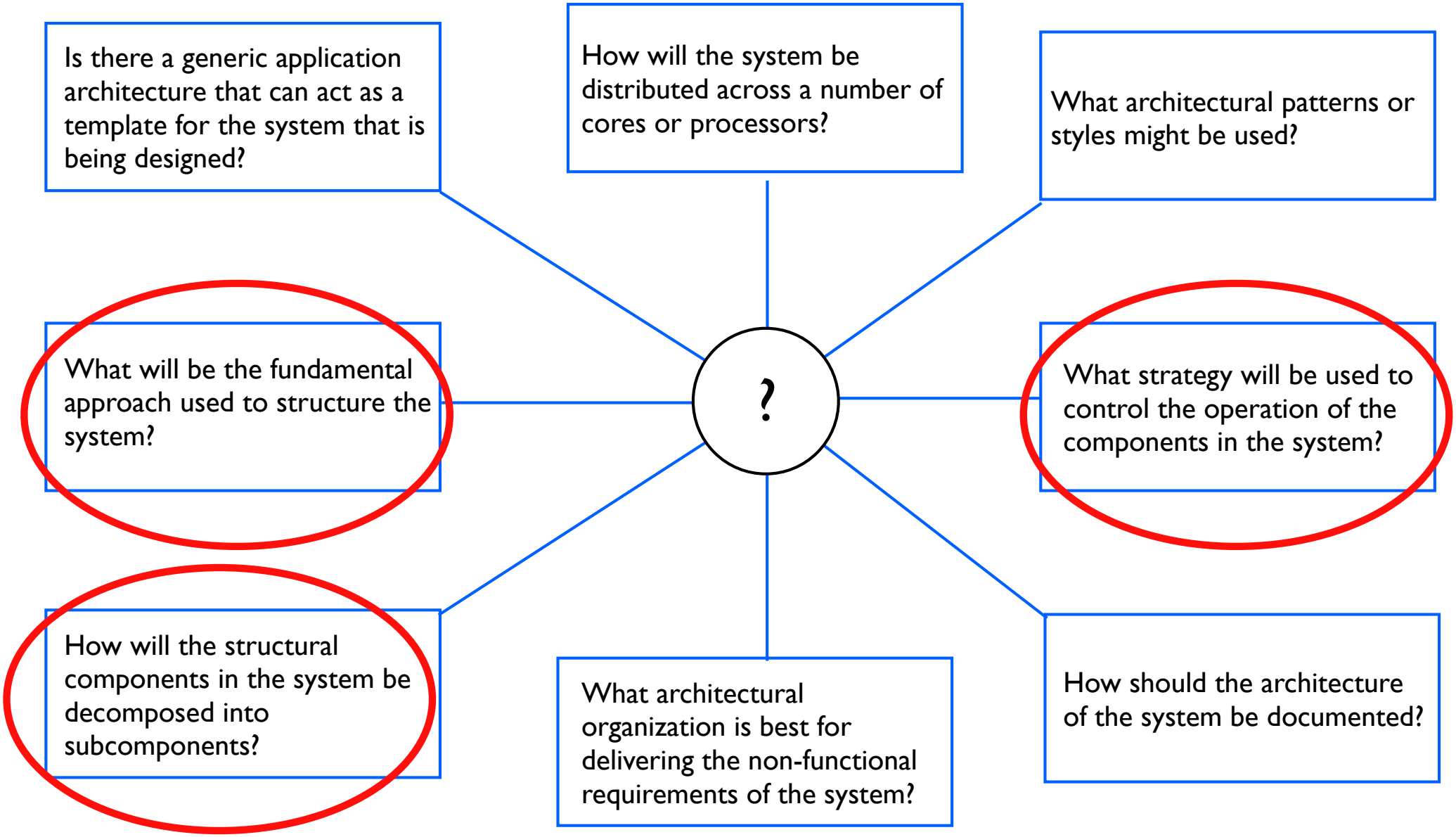
Which of these are "architecture design" decisions?



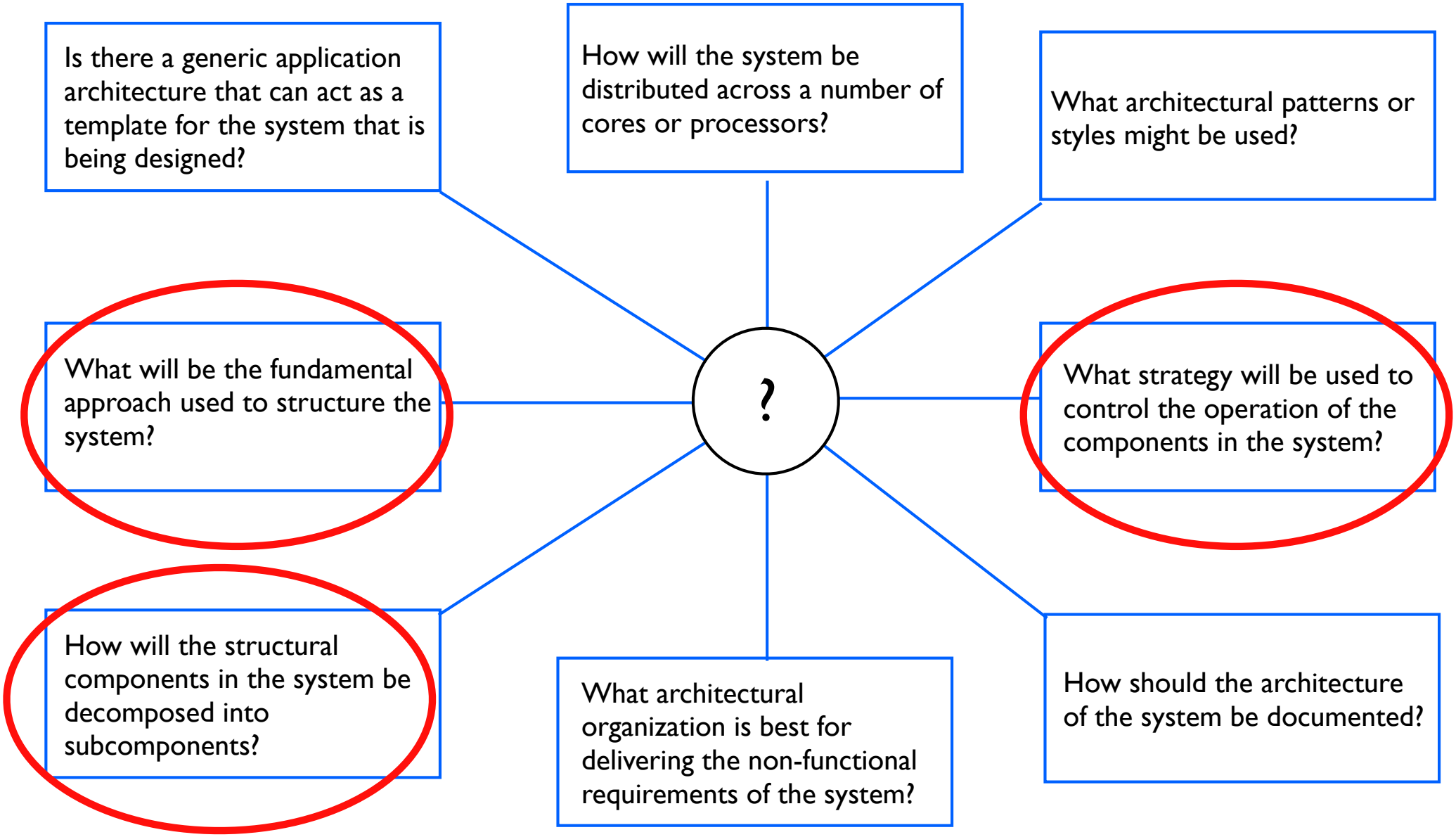
Which of these do architecture **patterns** address?



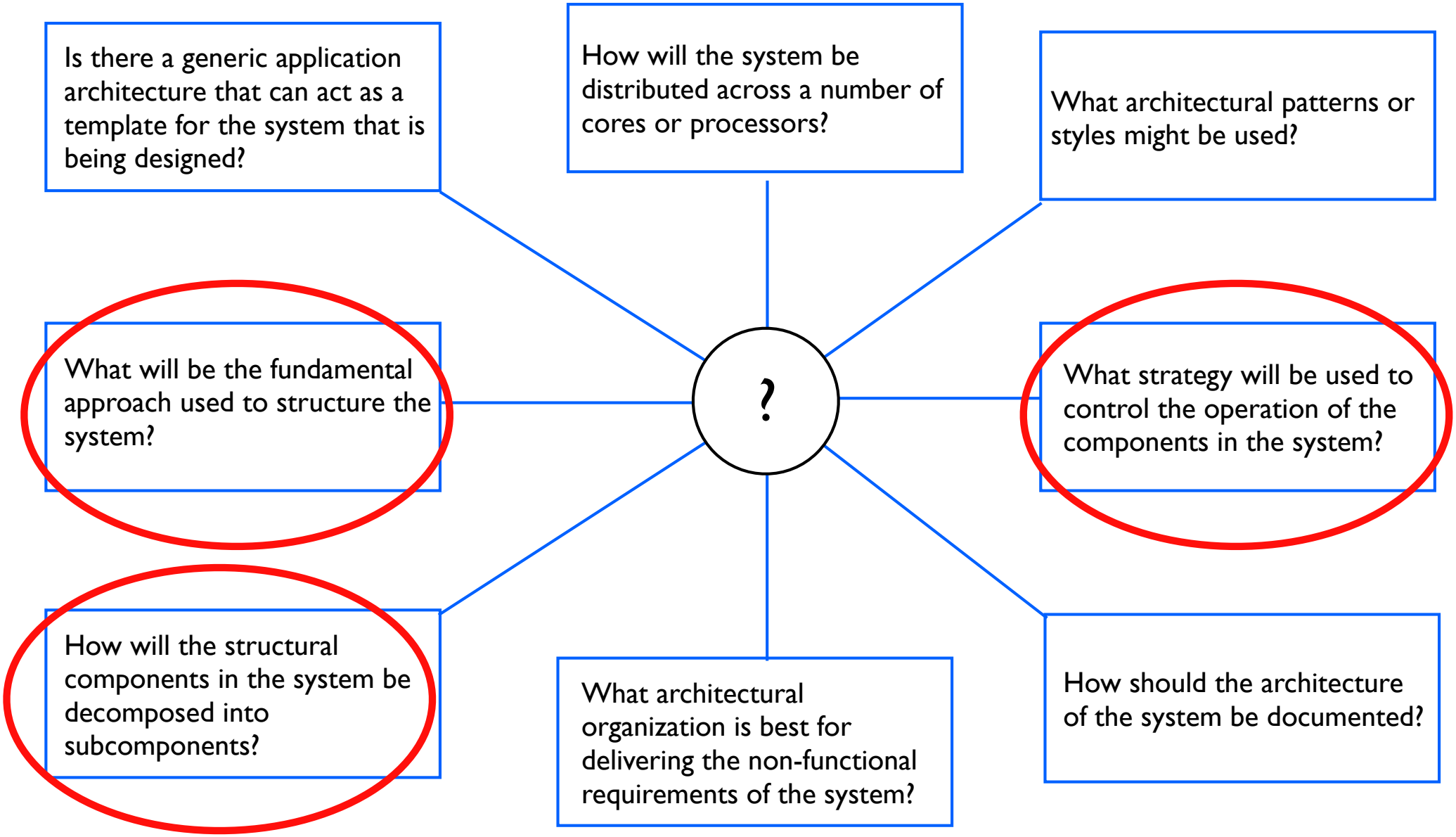
Which of these do architecture **patterns** address?



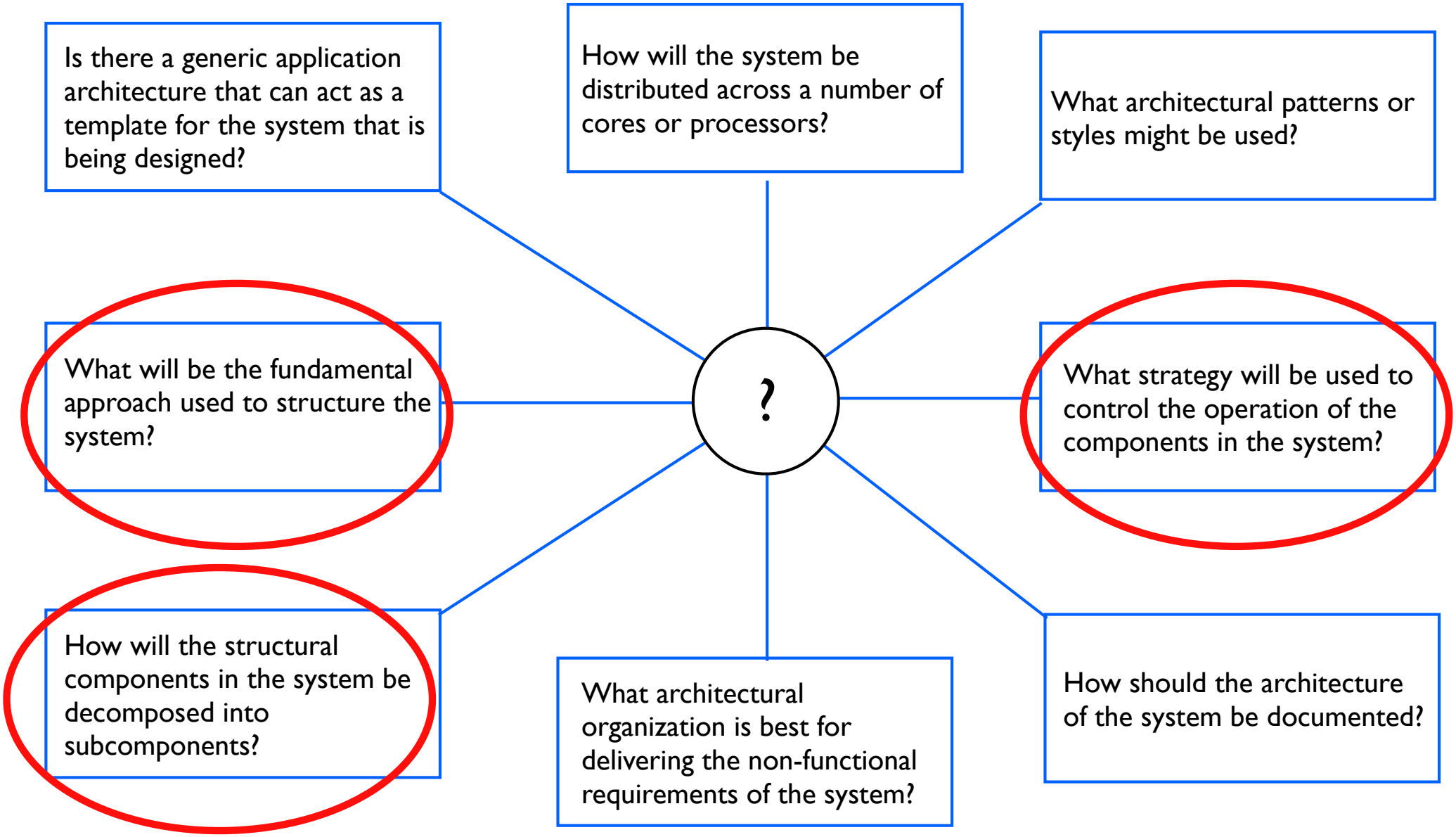
How do you **motivate** architecture design decisions?



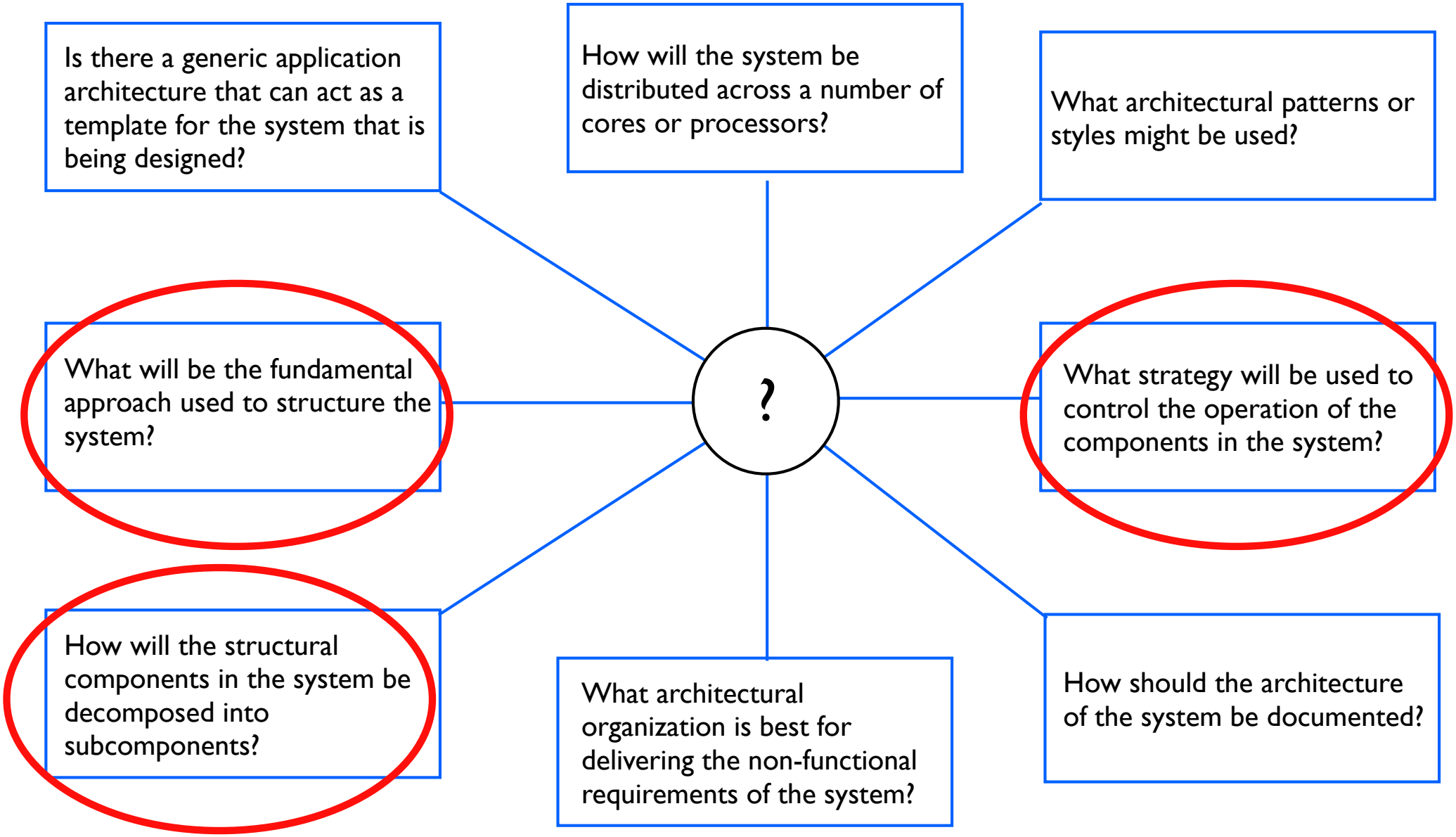
Architecture addresses non-functional requirements



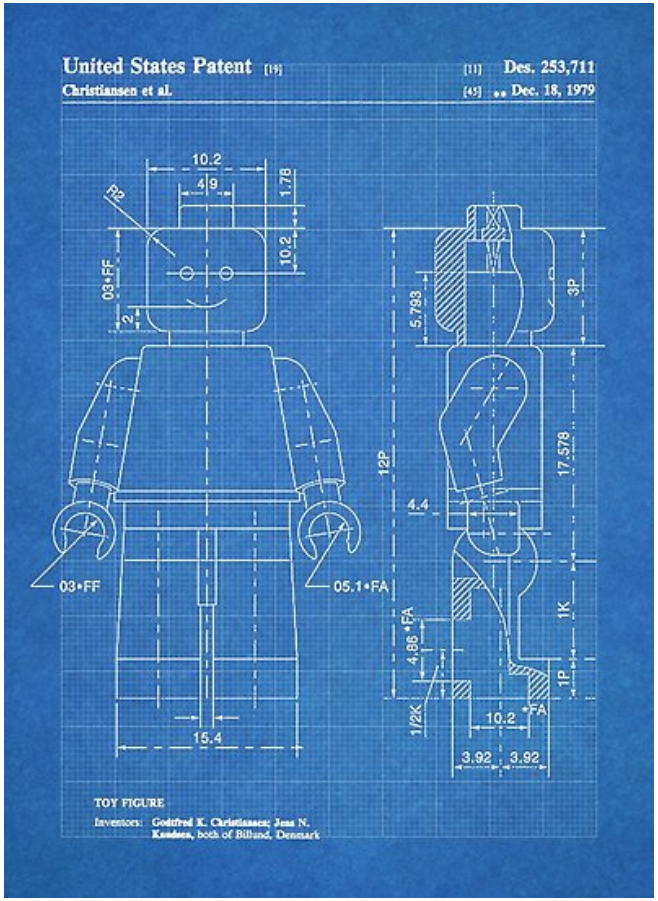
Components address functional requirements



Does architectural design only address structural aspects of a system, i.e. how components are organised?



Architectural design addresses structure and behaviour



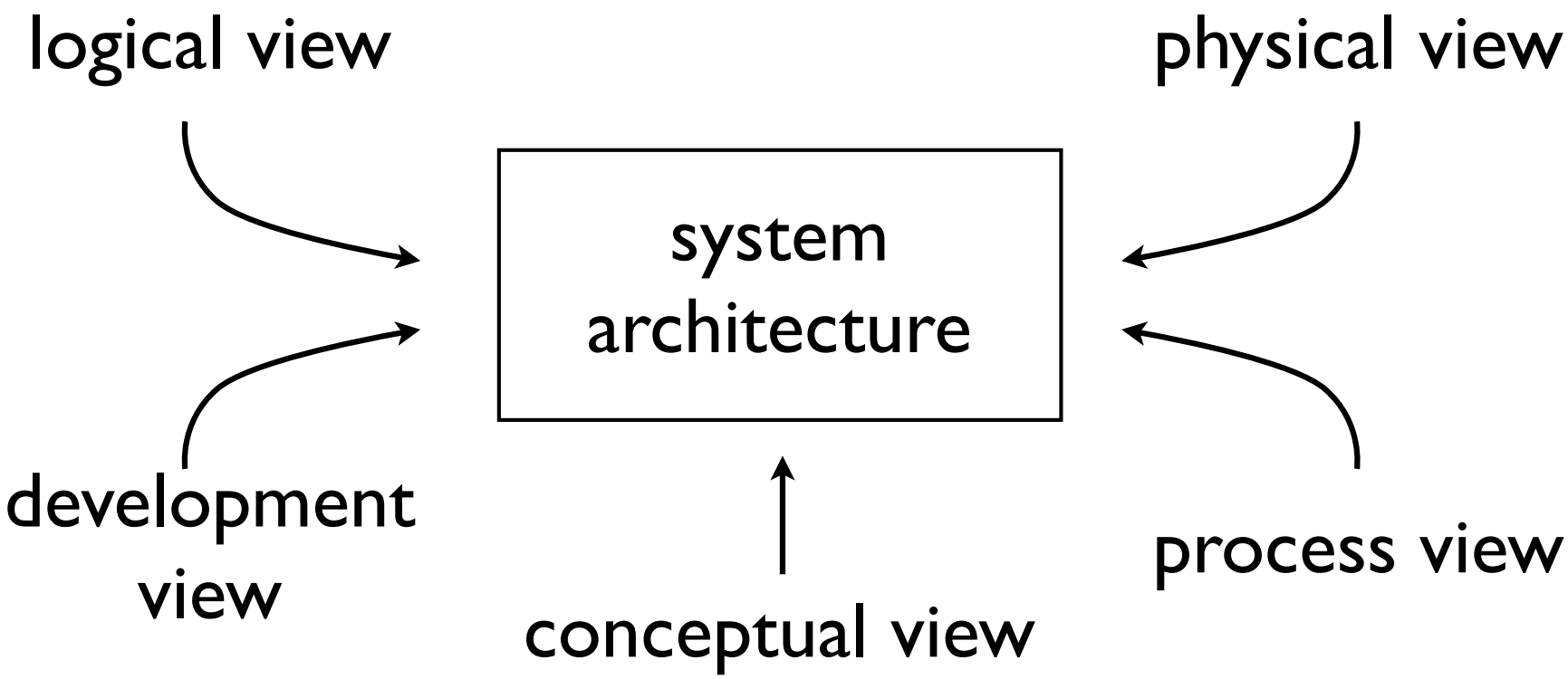
Architecture design - **viewpoints**

Architecture designs are like blueprints - “look” at the system from a particular perspective

Can’t represent all aspect of system in one diagram

Different stakeholders need different info

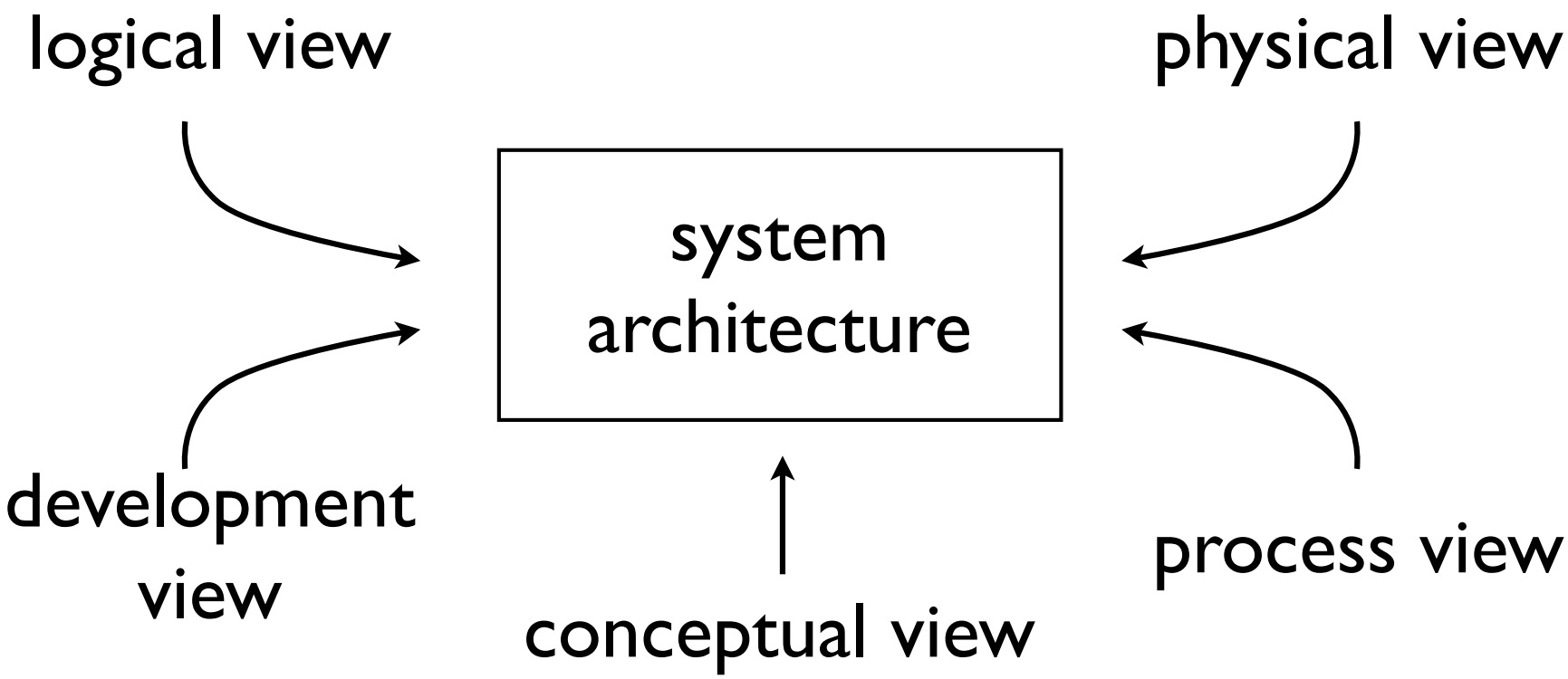
Krutchen 4+1



we'll match definitions to views...

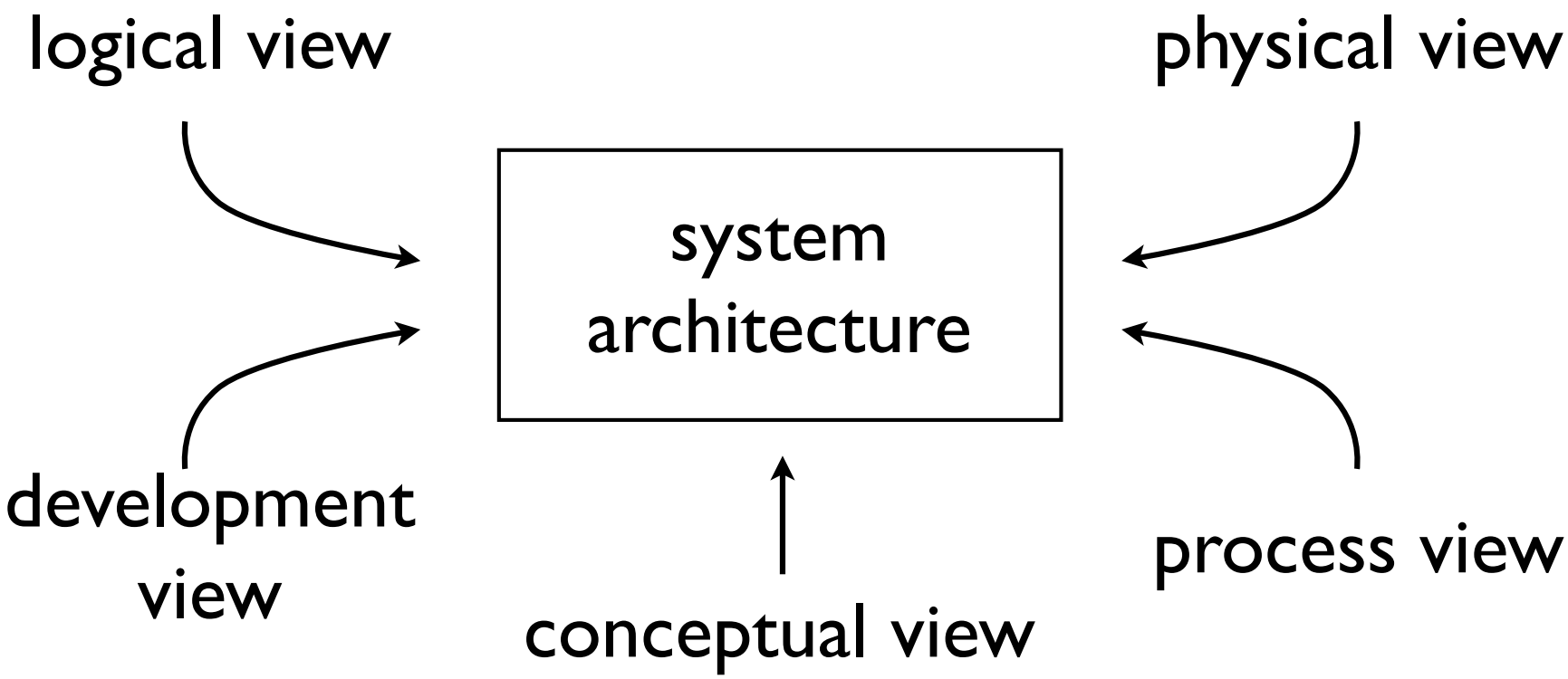
for each view, consider what UML diagrams are related

Krutchen 4+1



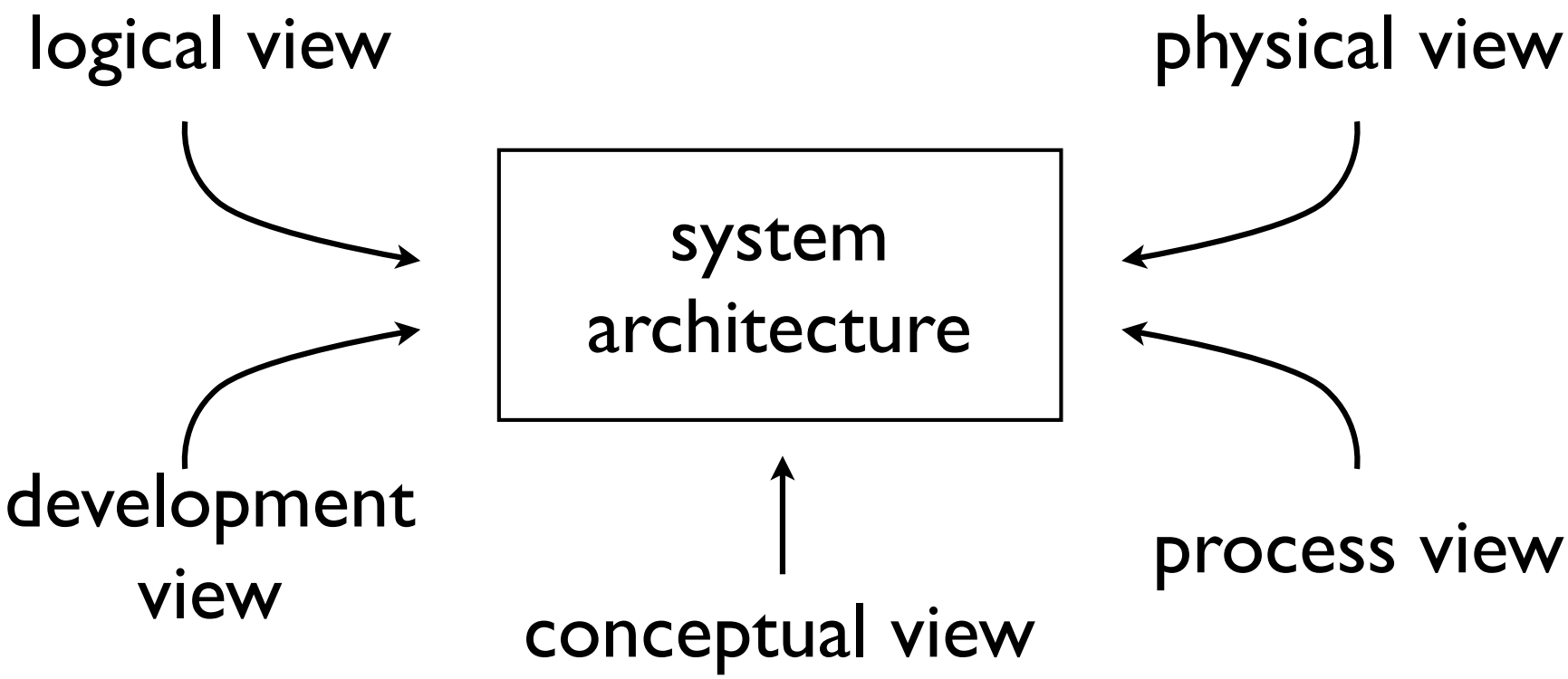
shows key abstractions in system as objects and object classes (in OO programming context)

Krutchen 4+1



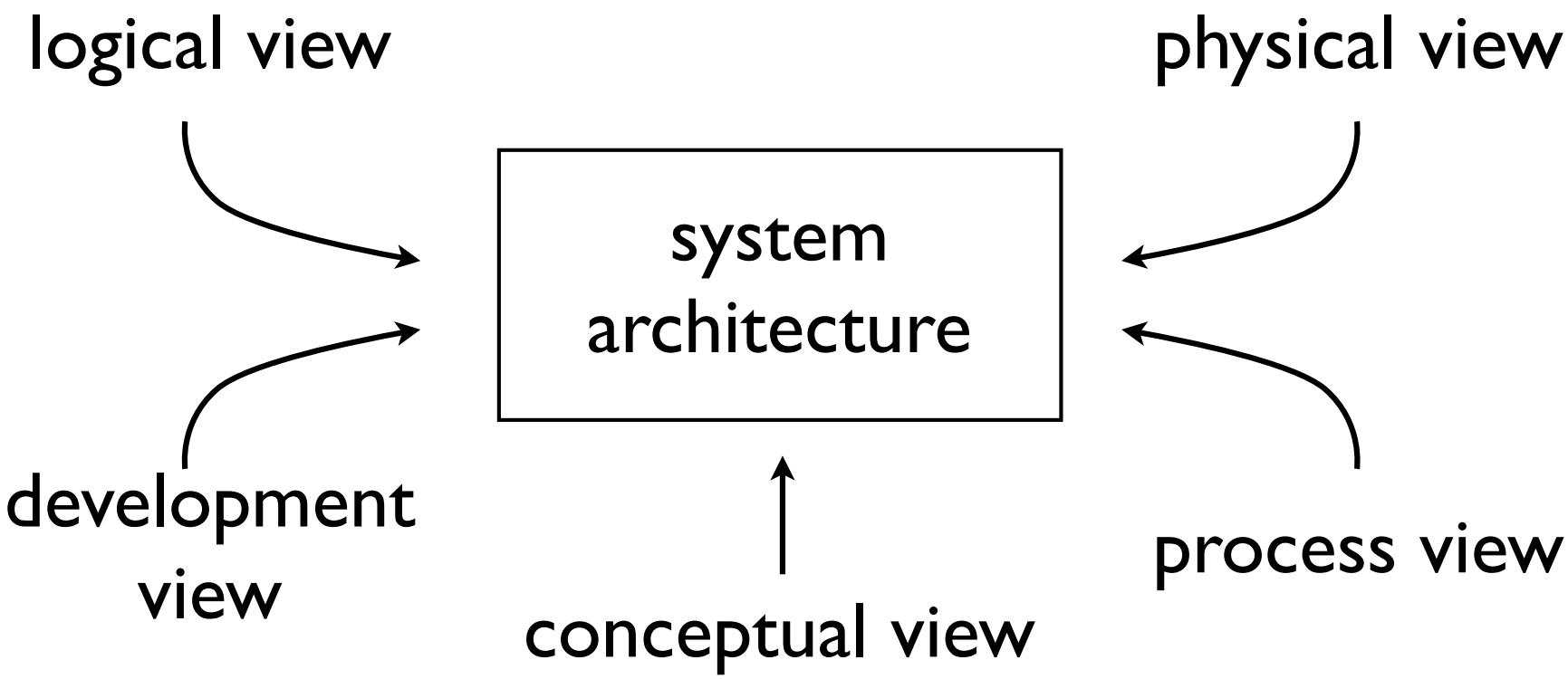
shows interacting processes during runtime

Krutchen 4+1



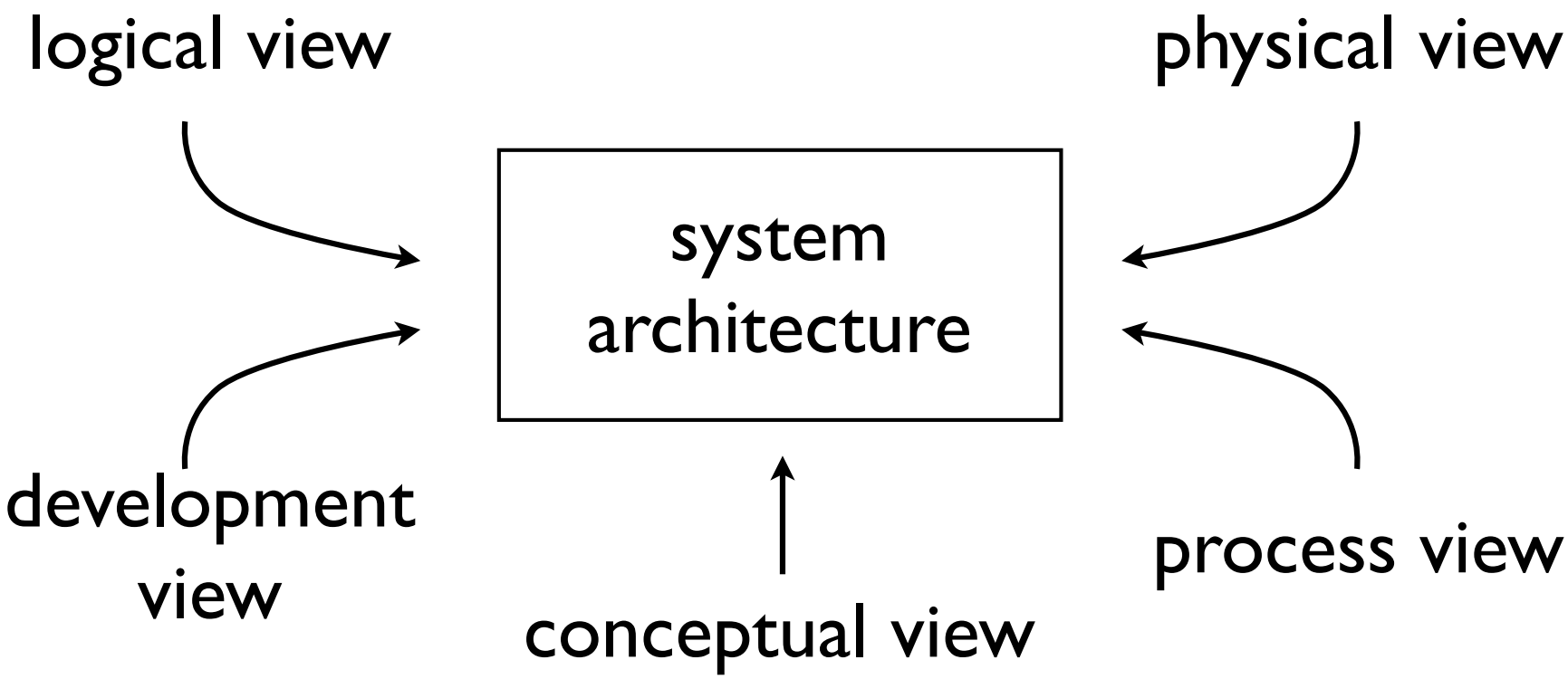
shows breakdown of software into components, implemented by single developer/team

Krutchen 4+1



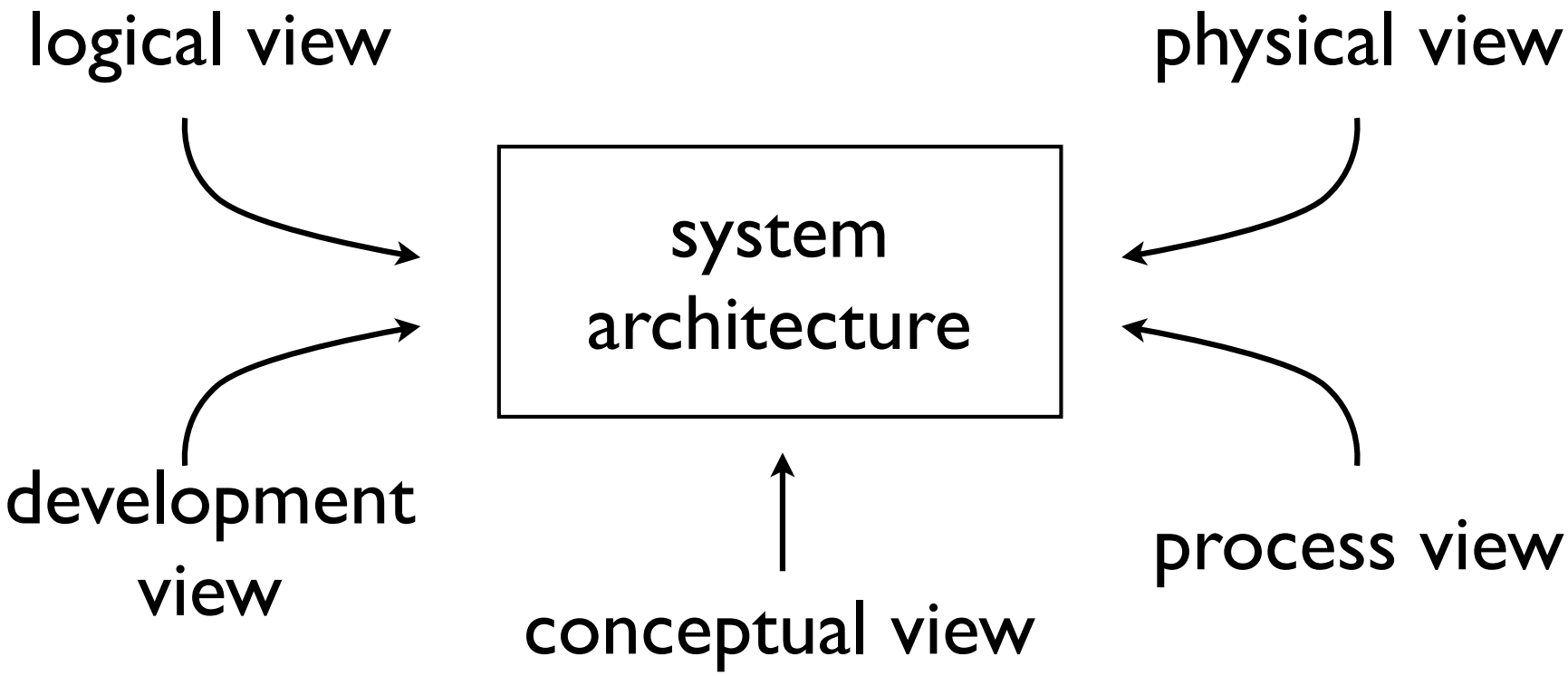
shows how software components distributed across different processors

Krutchen 4+1



abstract view, basis for decomposing high-level requirements into more detailed specification

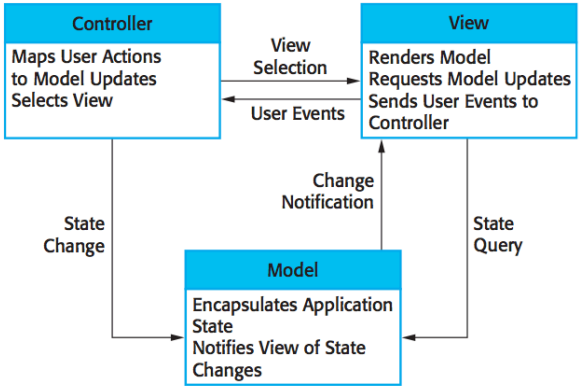
Krutchen 4+1

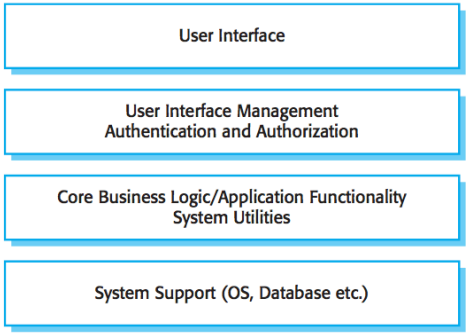


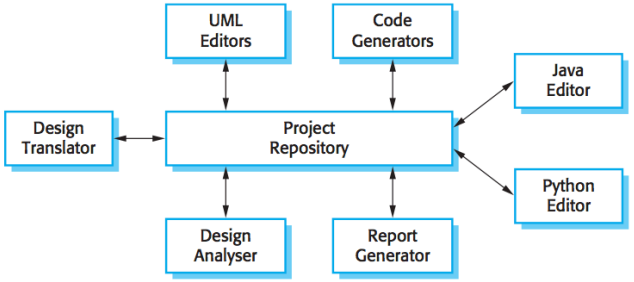
https://en.wikipedia.org/wiki/4%2B1_architectural_view_model

Note: "Conceptual view" is sometimes replaced with "Scenario view"

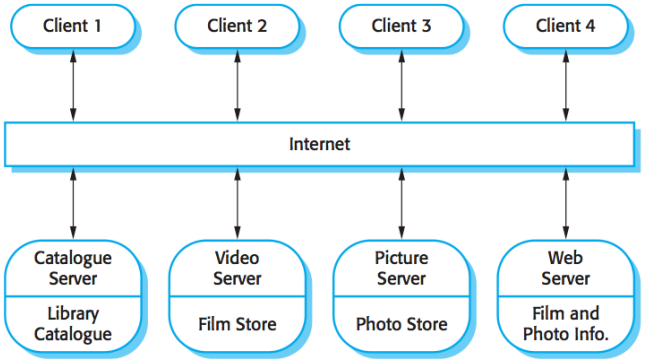
architecture patterns

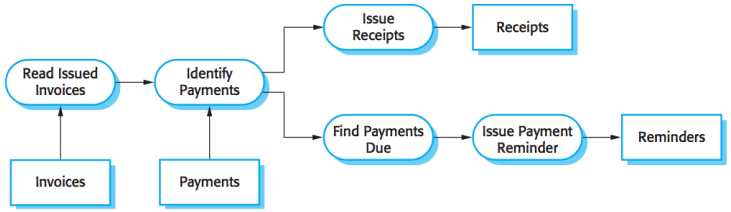




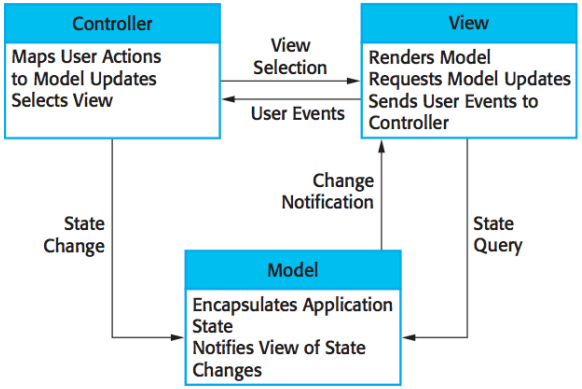


names of architectural styles?

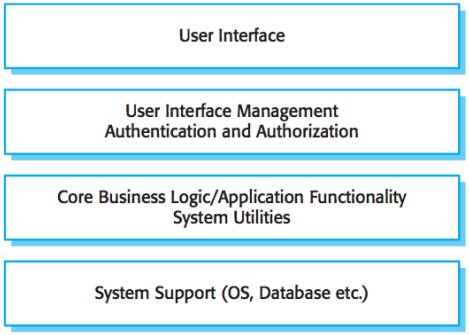




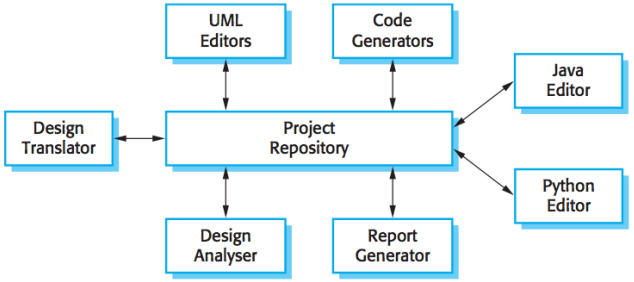
architecture patterns



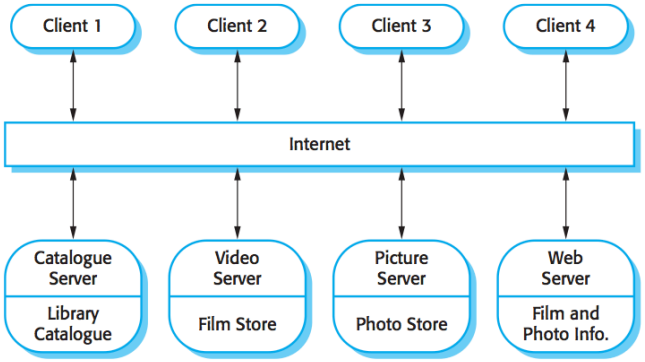
Model-View-Controller



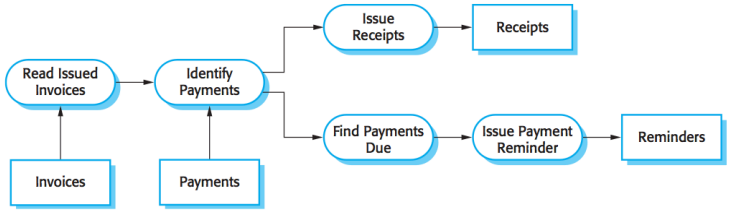
Layers



Repository



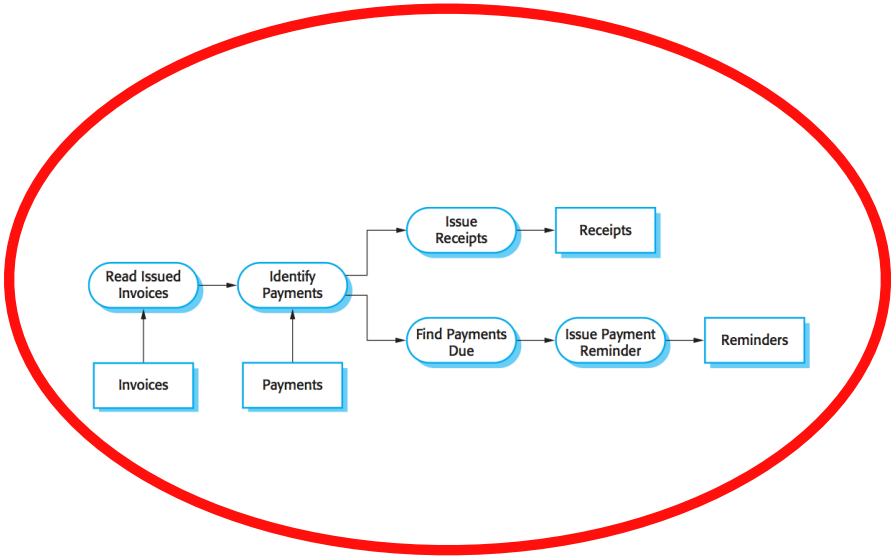
Client-Server (resp. 2-tier)



Pipe and filter

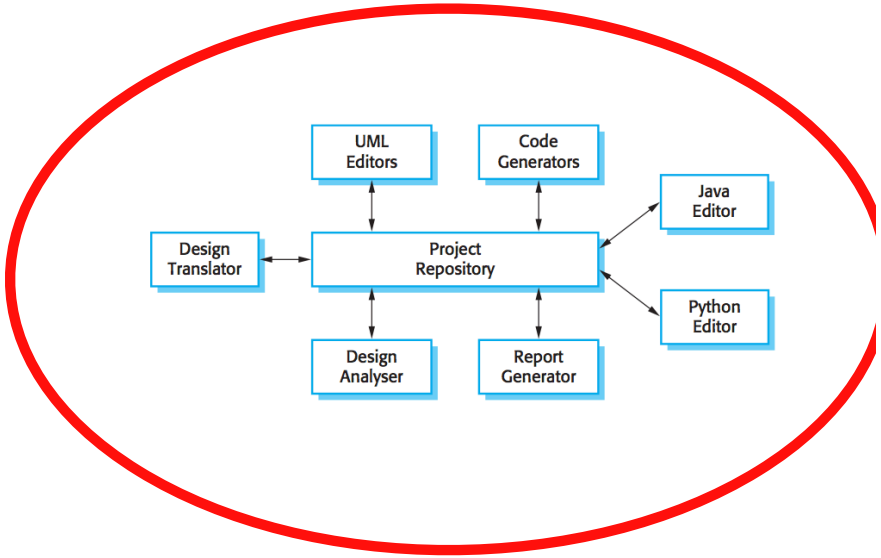
architecture patterns

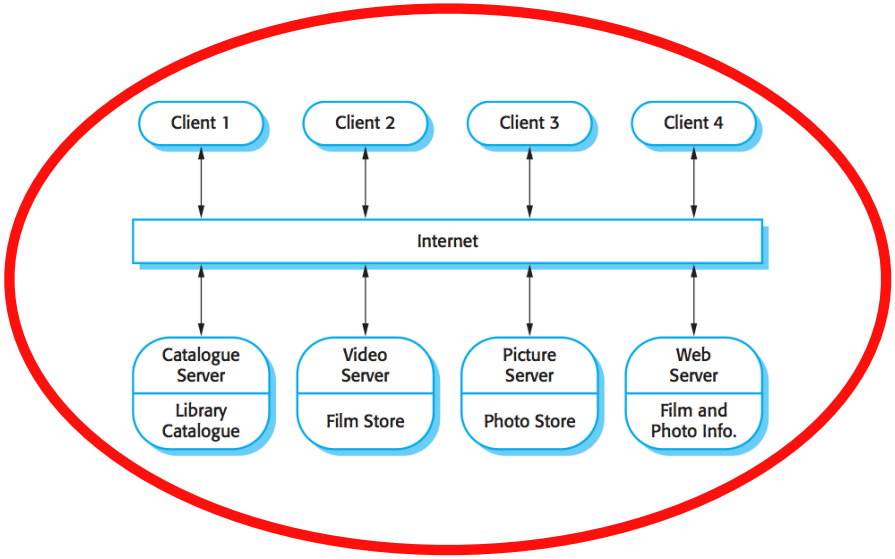
Only for heavy computational processing?



architecture patterns

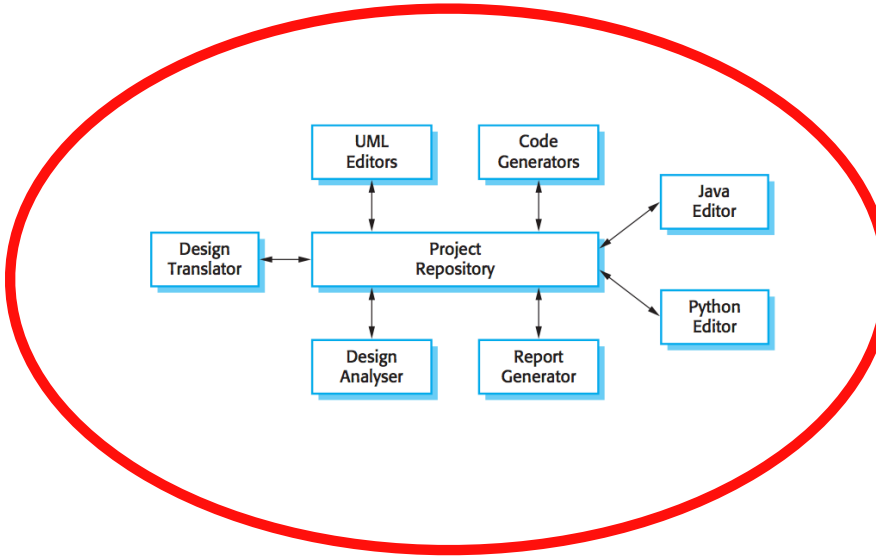
What are key differences?

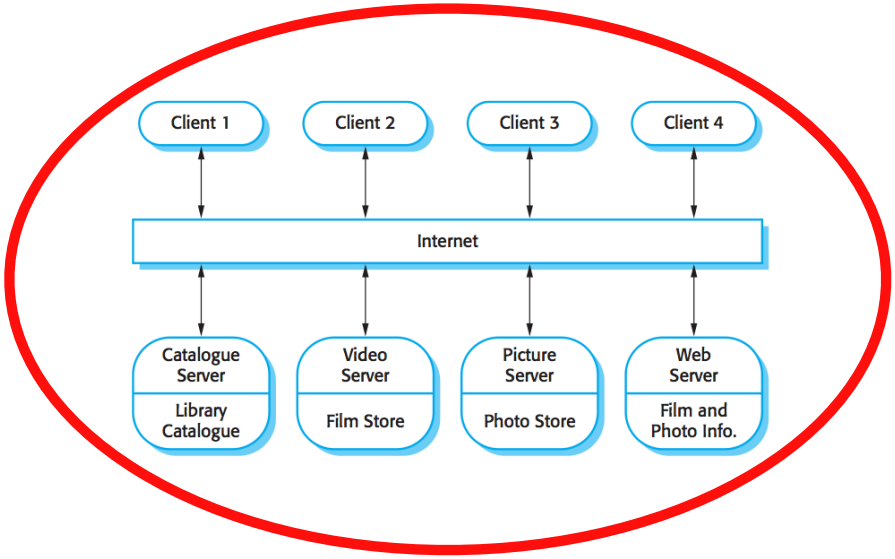




architecture patterns

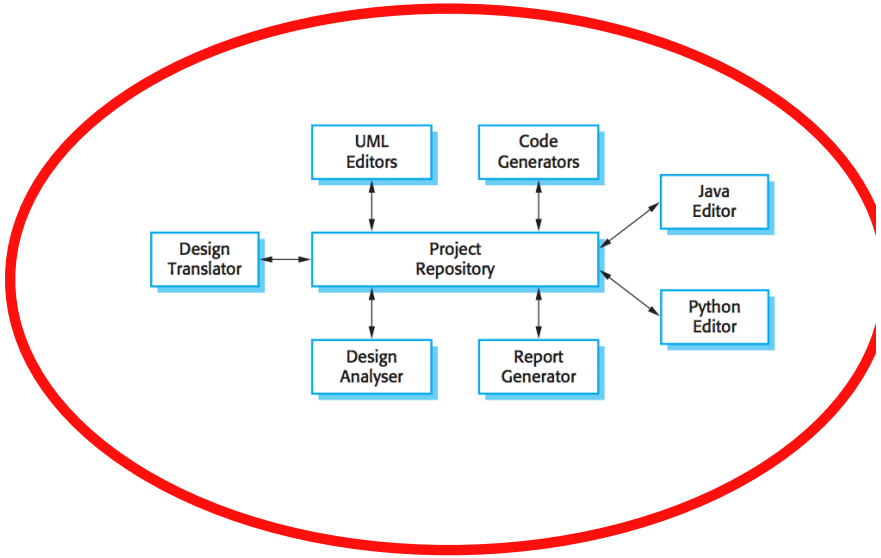
What changes if we want to add a new system feature?

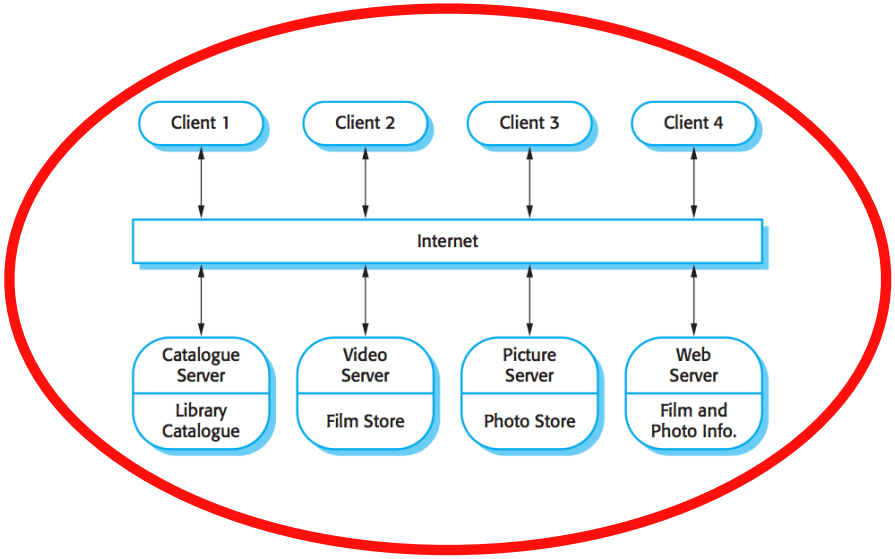




architecture patterns

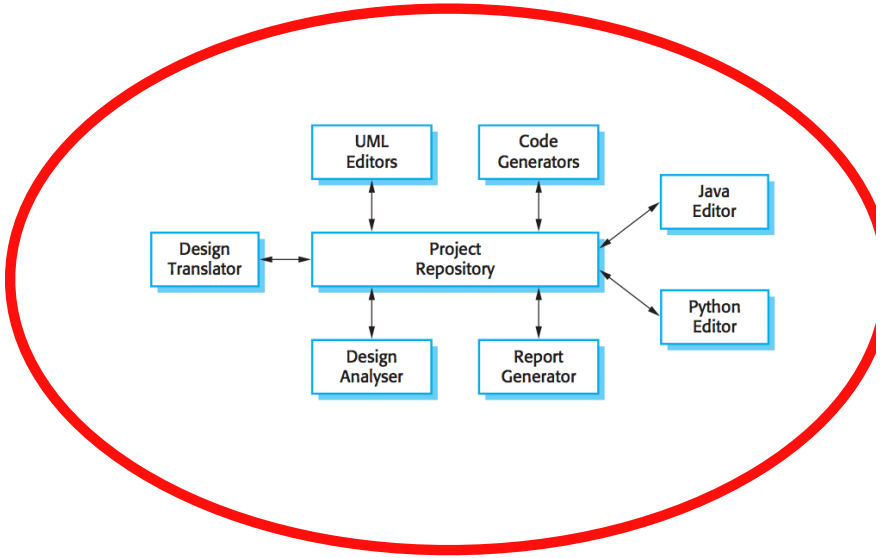
What about during **runtime**: are these architectures communicating the same **type** of system information?

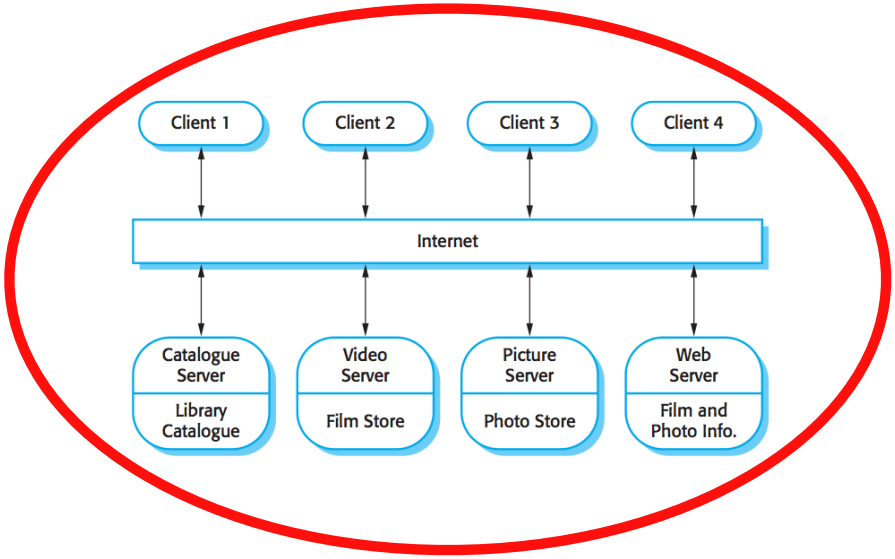




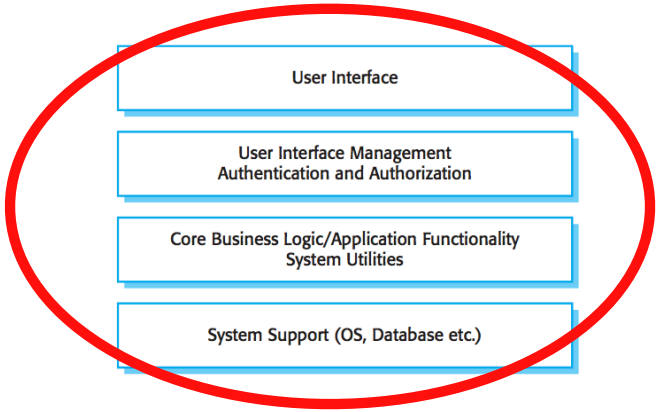
architecture patterns

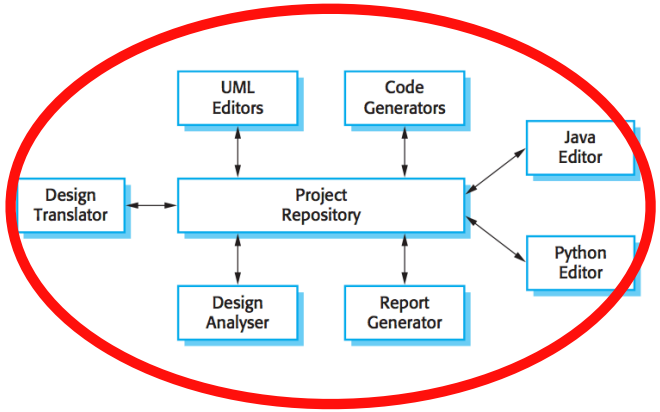
e.g. do components and connections change during runtime?





architecture patterns

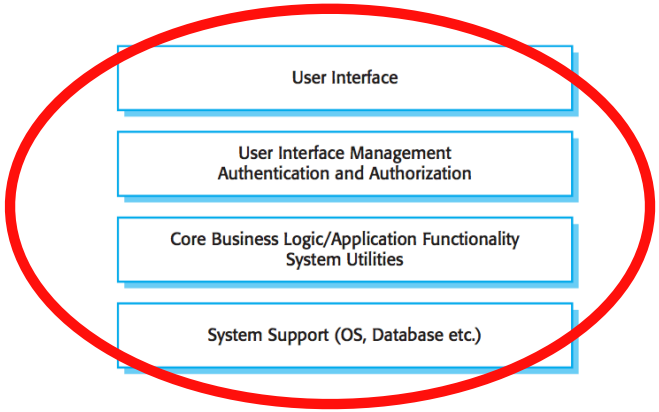


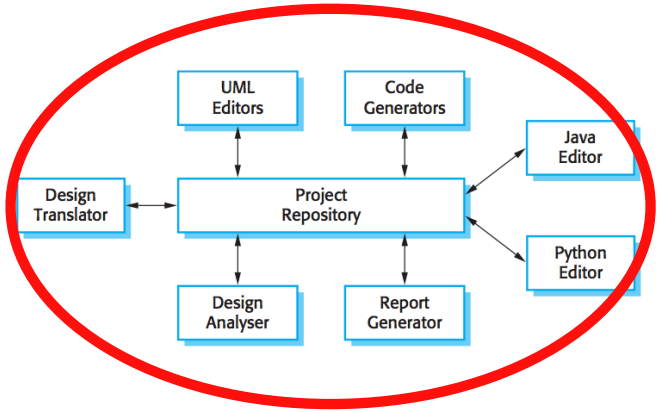


What are some differences between these patterns?

Do they differ in the non-functional requirements that they support?

architecture patterns



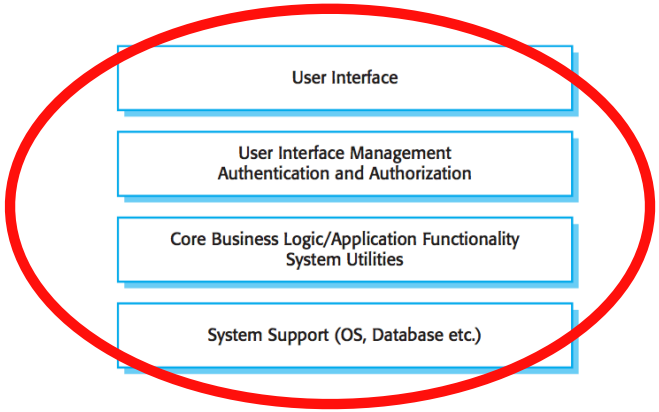


extensibility?
maintainability?
reusability?
availability?
security?
performance?

What are some differences between these patterns?

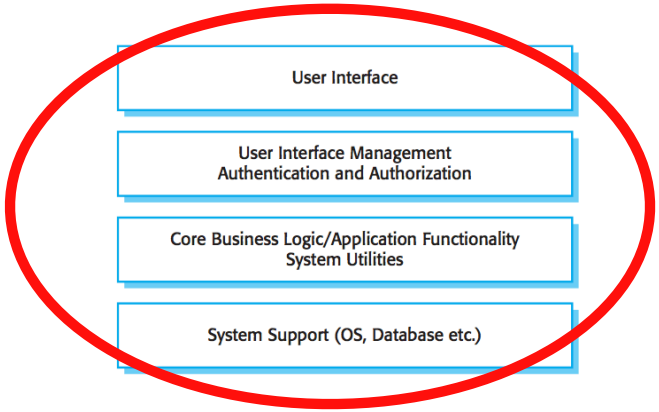
Do they differ in the non-functional requirements that they support?

architecture patterns

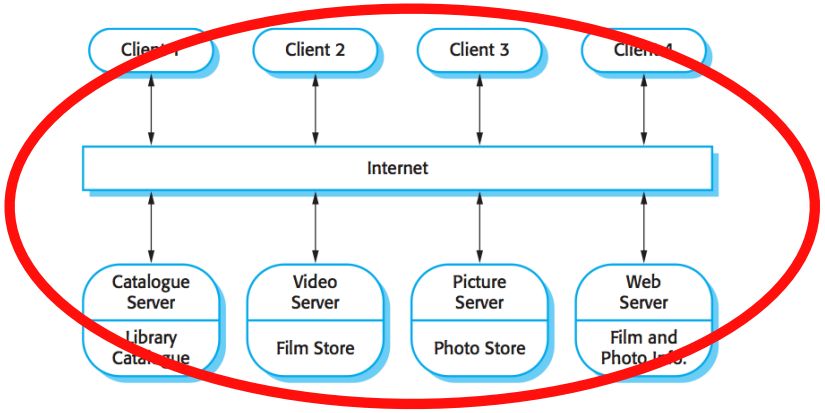


Layers vs
N-tier architecture?

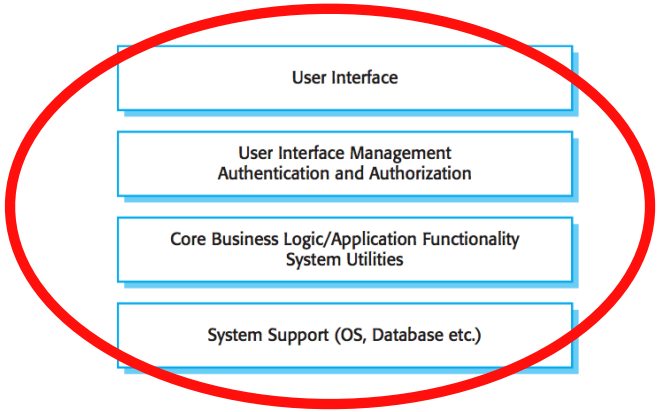
architecture patterns



What are some differences between these?

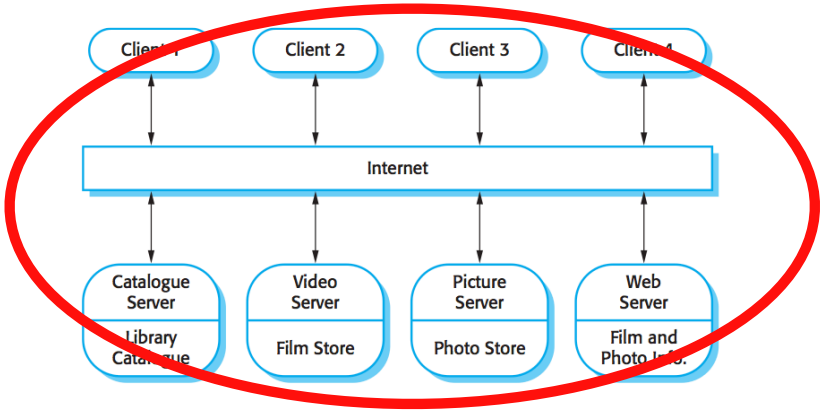


architecture patterns

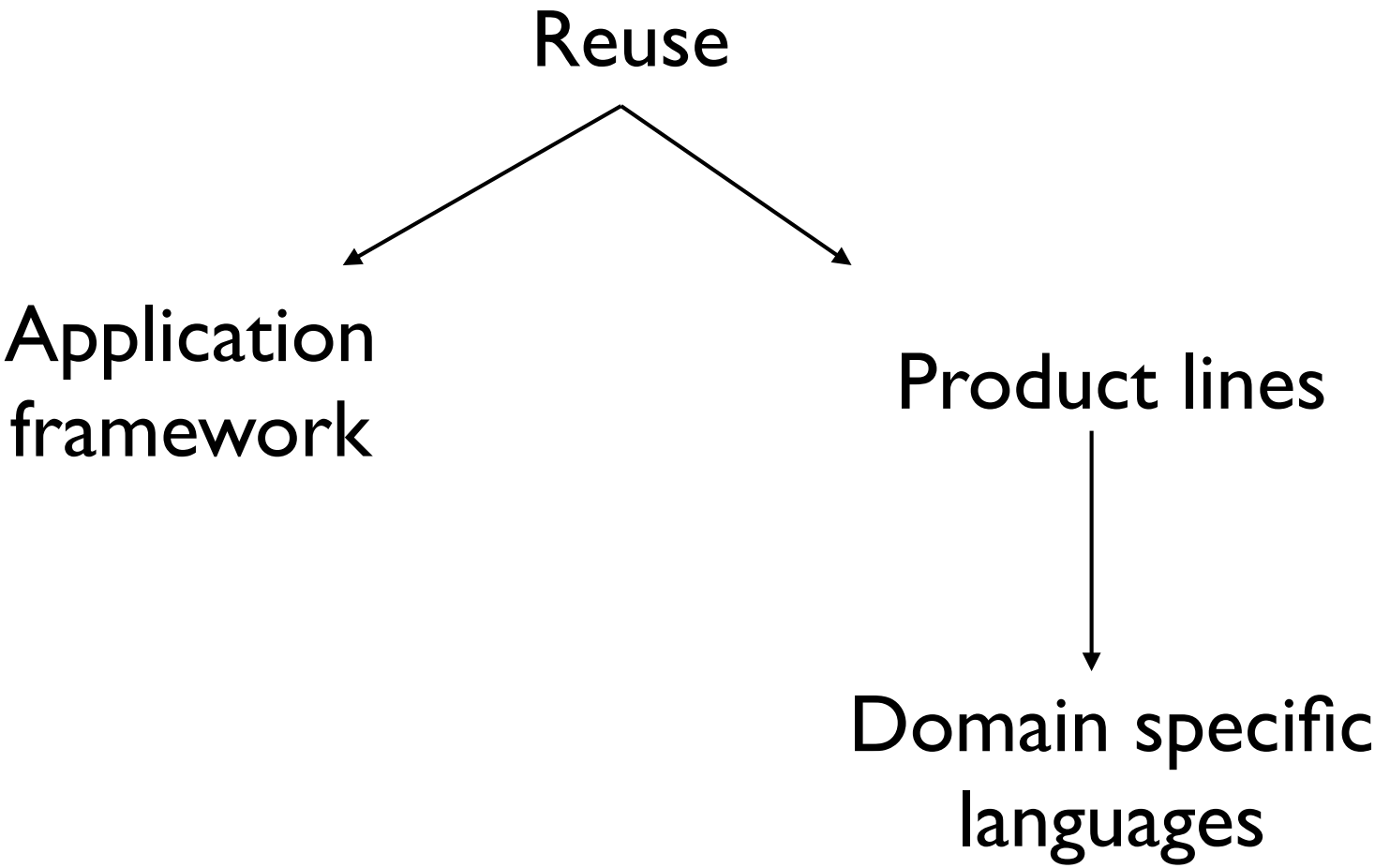


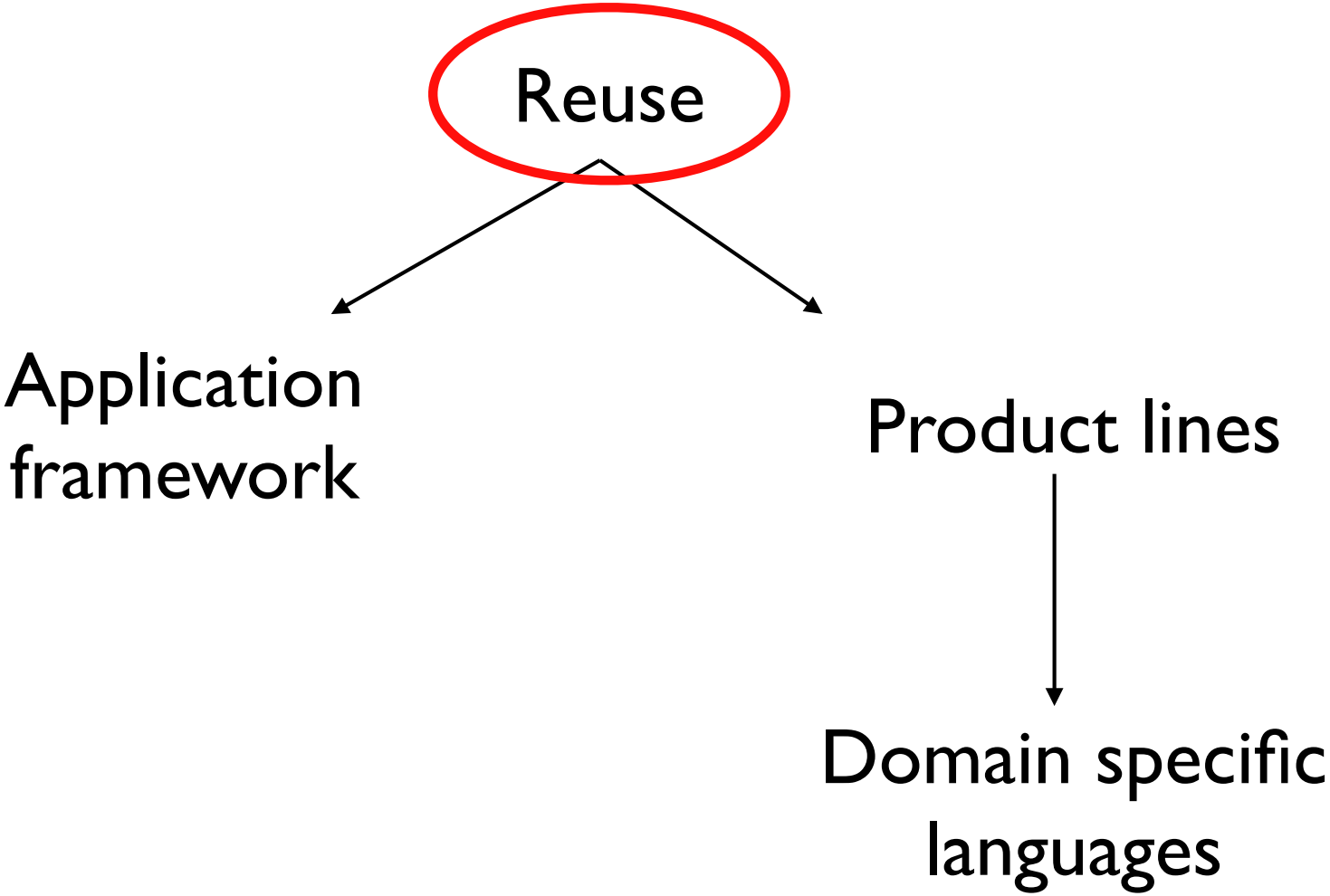
E.g. consider the **sequence** of component interactions

E.g. what about adding new features? (extensibility)



E.g. what about maintenance?


Reuse
Application framework
Product lines
Domain specific languages


Reuse
Application framework
Product lines
Domain specific languages

reuse

- reuse-based software engineering - reuse existing software as much as possible
- different scales:
  - system reuse
  - application reuse
  - component reuse ?
  - object, function reuse
  - concept reuse

reuse

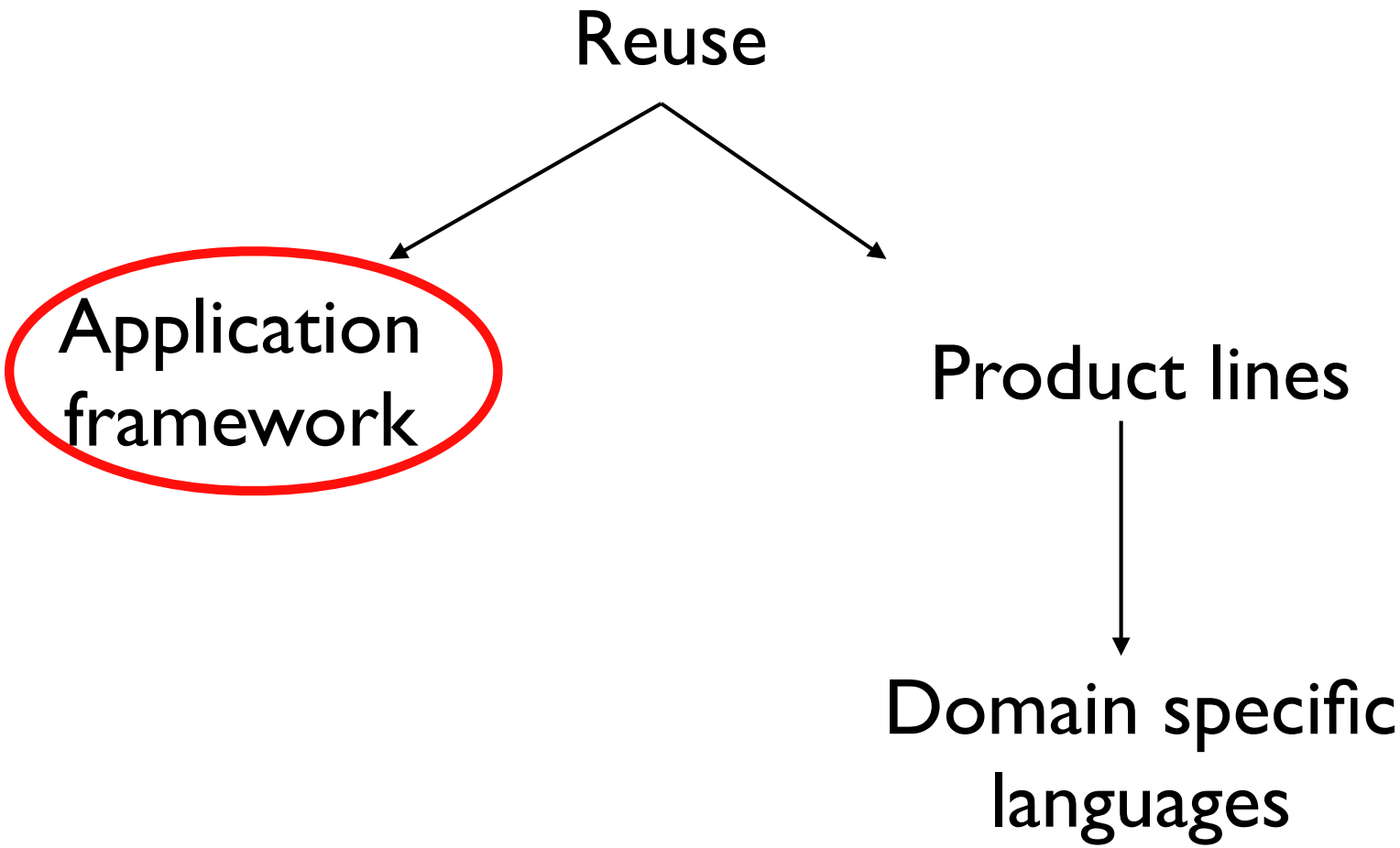
- **system reuse** - number of applications incorporated into larger "system of systems"
- **application reuse** - existing application incorporated into other system (e.g. software product lines)
- **component reuse** - entire subsystems or just small modules
- **object, function reuse** - single functions, standard libraries offer this

reuse

- in practice re-using specific components tricky
- **<u>concept</u> reuse**
  - e.g. “design patterns” (coming in a later lecture...)

early problems with reuse

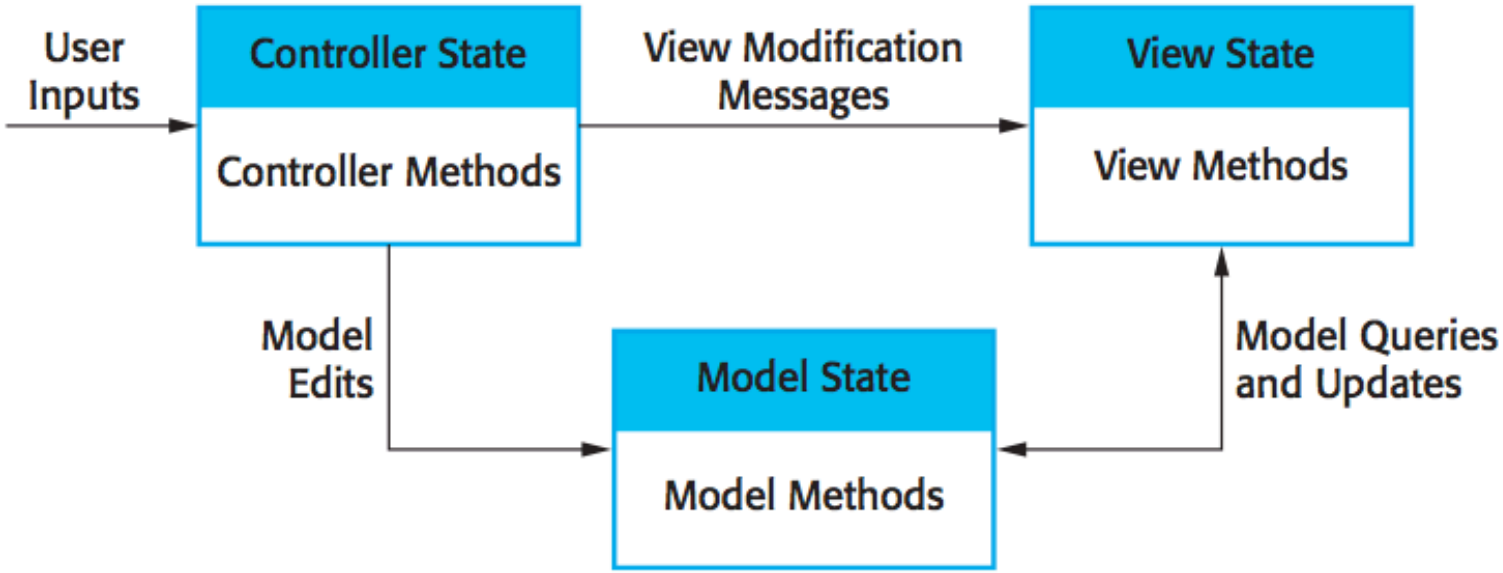
- early hope of OO programming: reuse object classes
- in practice:
  - objects too fine-grained to be reused
  - quicker to just re-implement


Reuse
Application framework
Product lines
Domain specific languages

application framework

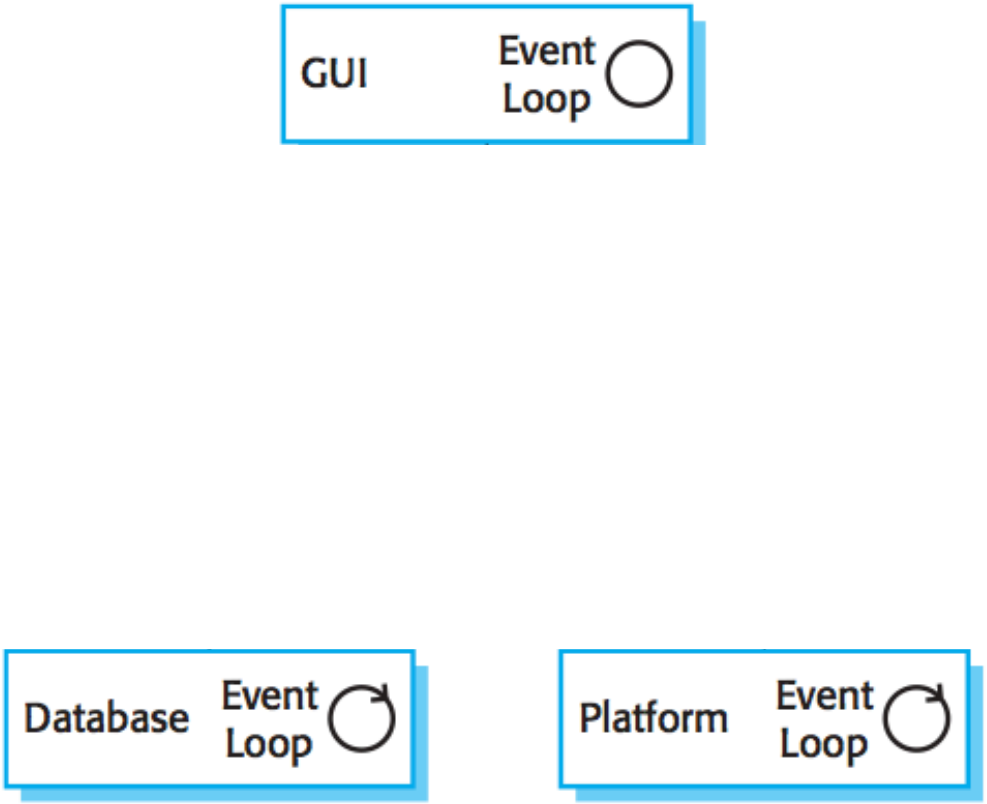
- idea: reuse software at higher level of abstraction
- not fine-grained objects, but larger-grained "framework"
- application framework: provides a generic structure of an application
  - you "extend" (in sense of OO) to implement your functionality
  - it is software - a collection of abstract and concrete classes (if it's an OO framework)

application framework



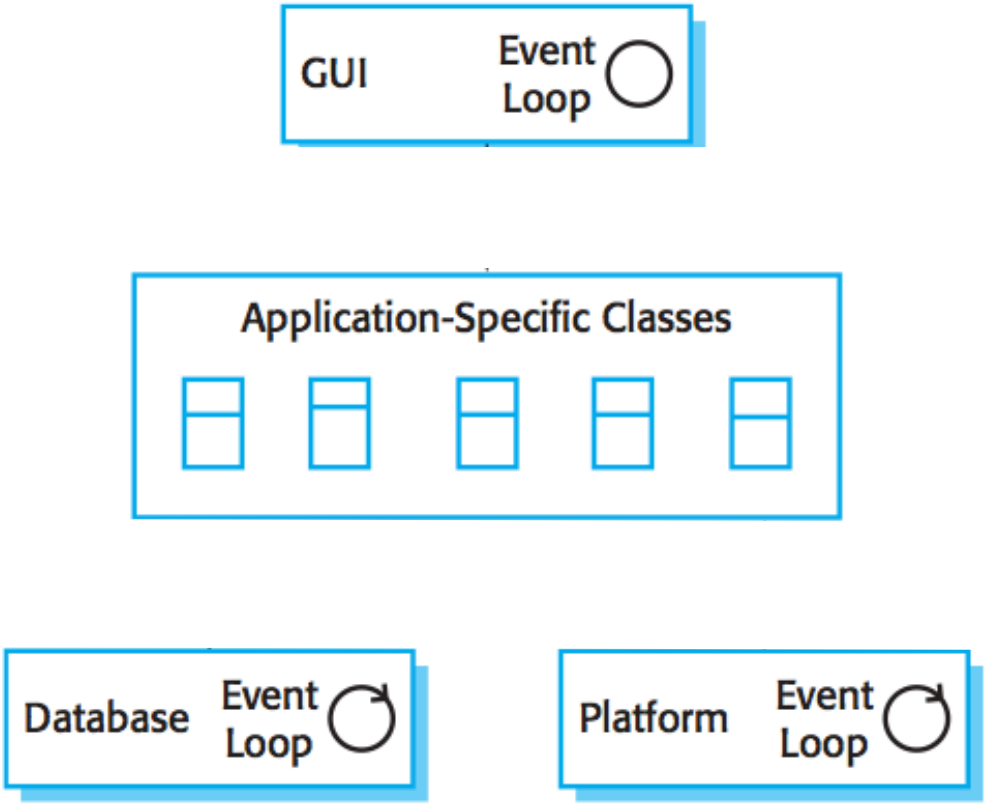
e.g. framework could be "bare-bones" MVC software - you extend certain classes to implement particular model etc.

application framework



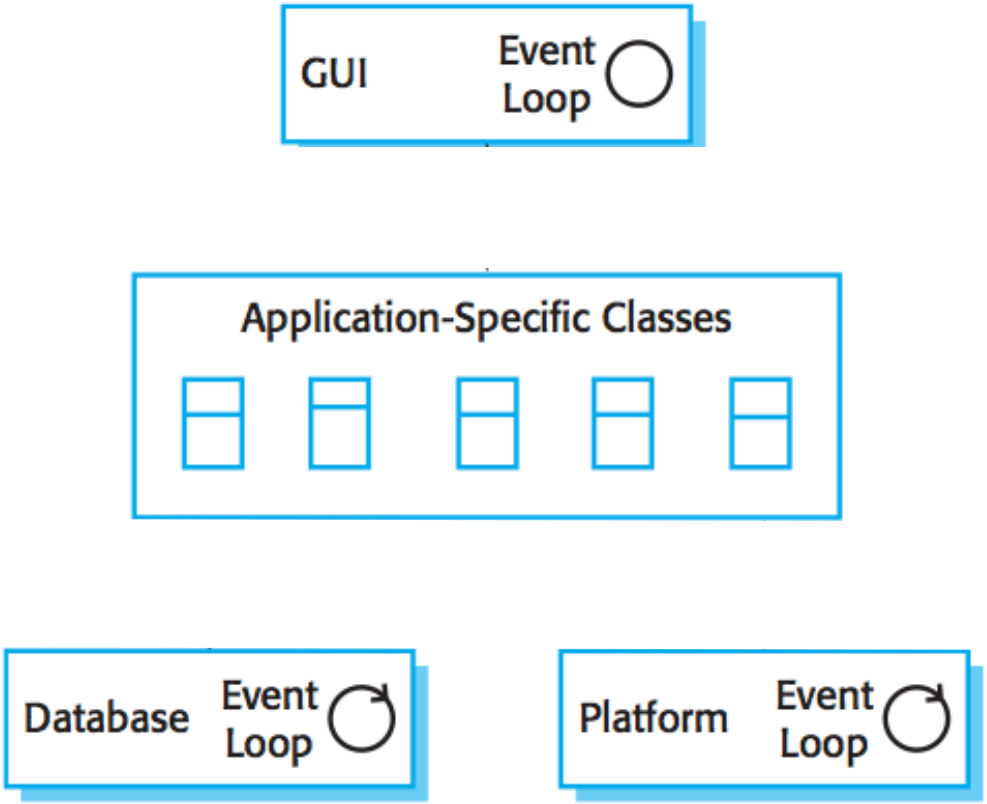
e.g. here's a framework: implements some generic components

application framework



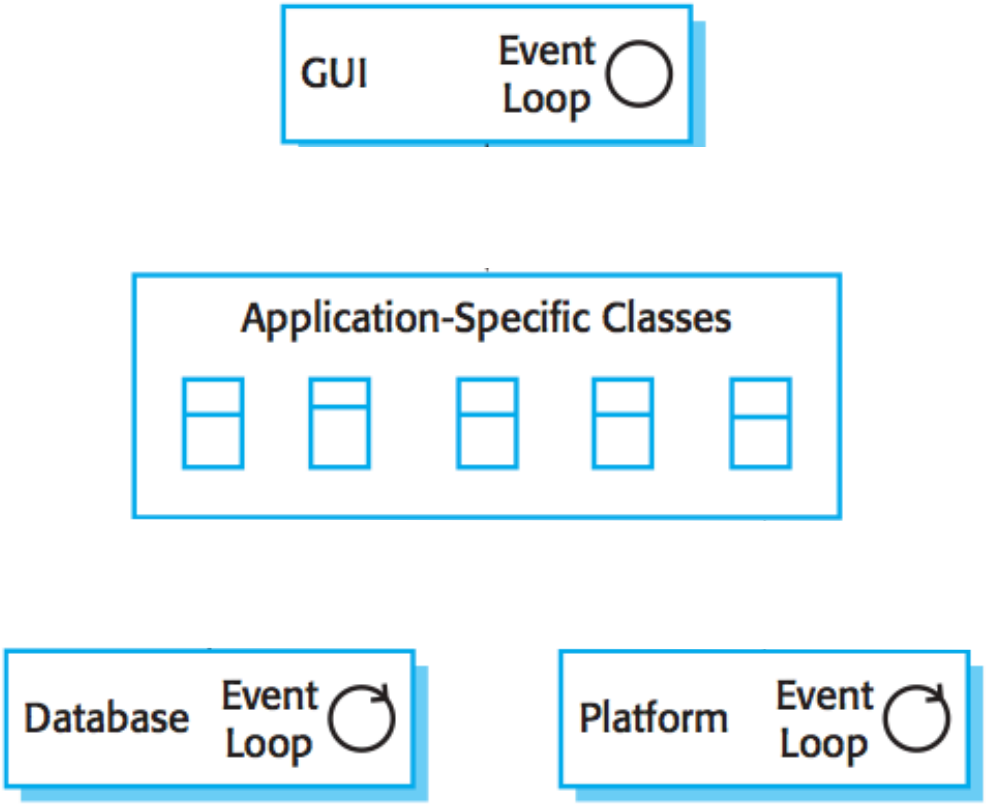
you create your own classes, typically you will extend certain framework classes

application framework



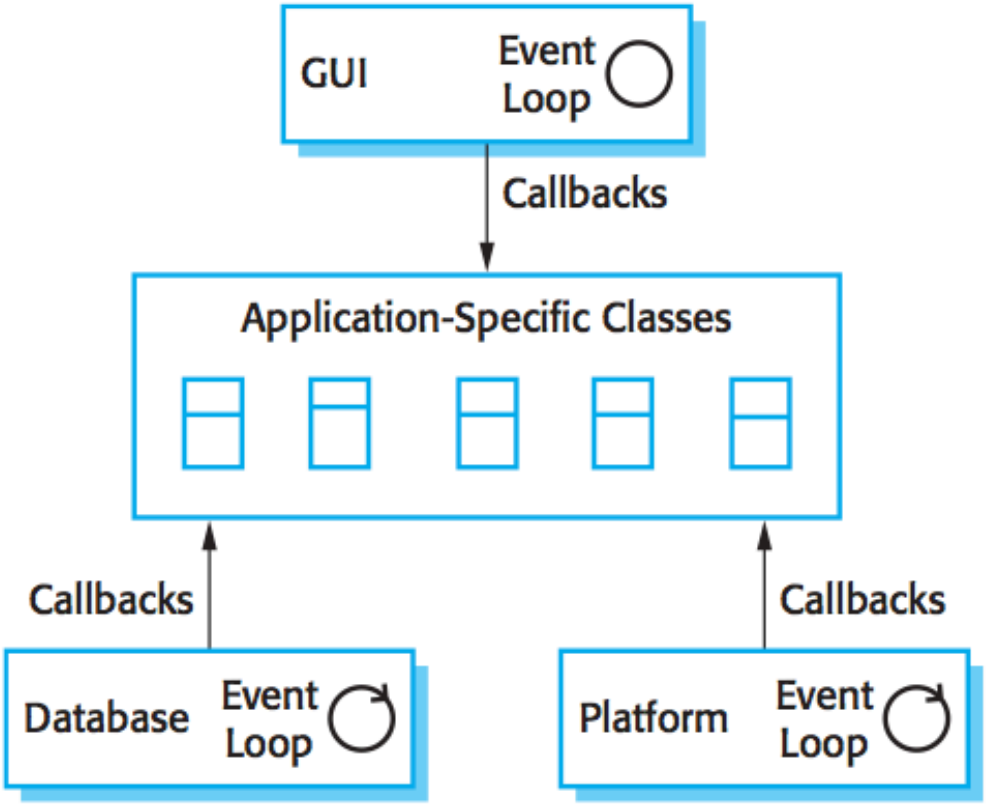
when an event occurs in a framework class (like a mouse click in the GUI, or a Database update), you'll want something to happen in your application-specific class...

application framework



in general this is called a “hook” - you register your “hook” method, and when the event occurs the framework object will invoke your hook

application framework

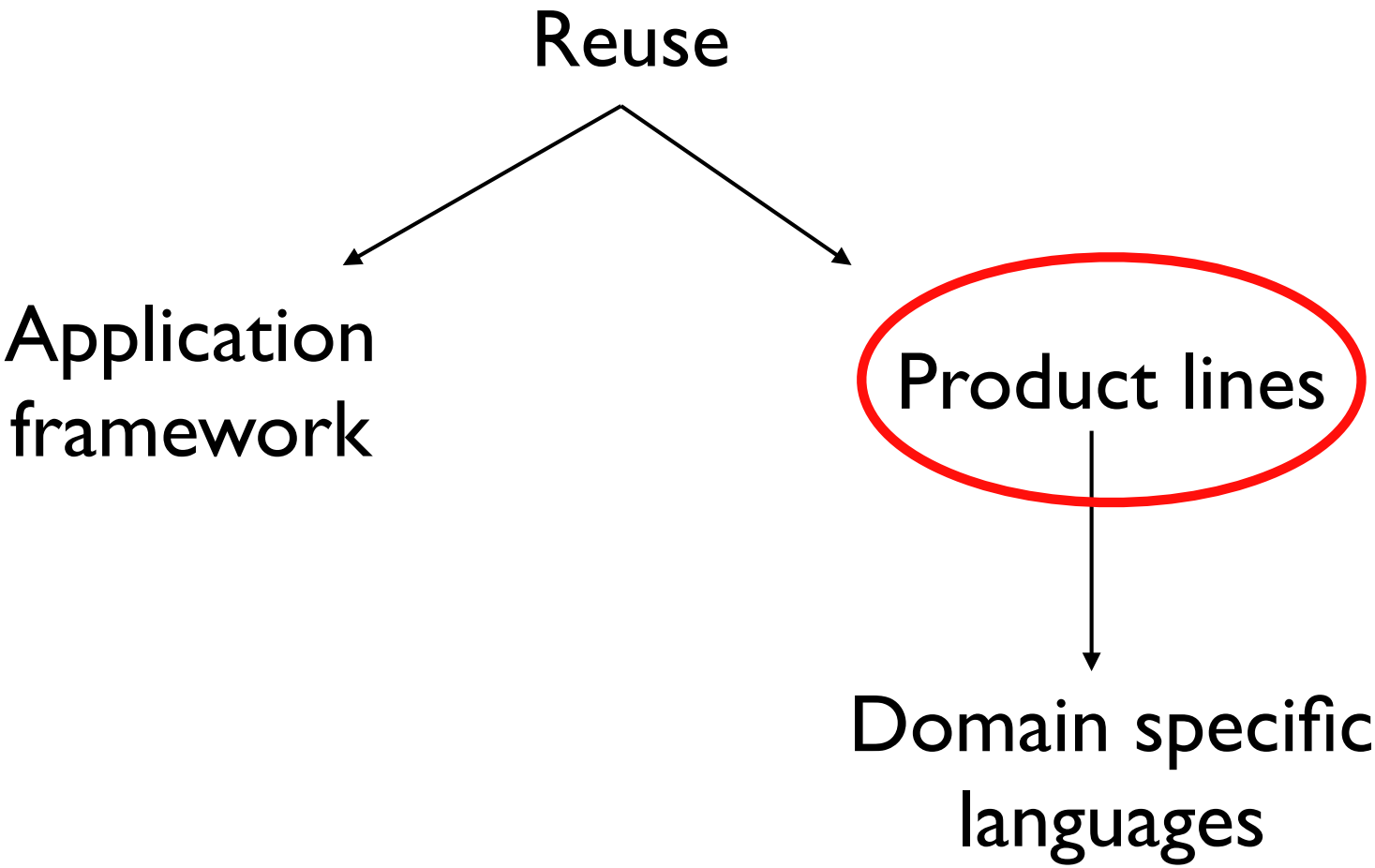


(note. hook and callback terminology can vary)

application framework

"an integrated set of software artefacts (such as classes, objects and components) that collaborate to provide a reusable architecture for a family of related applications."

e.g. Microsoft .NET

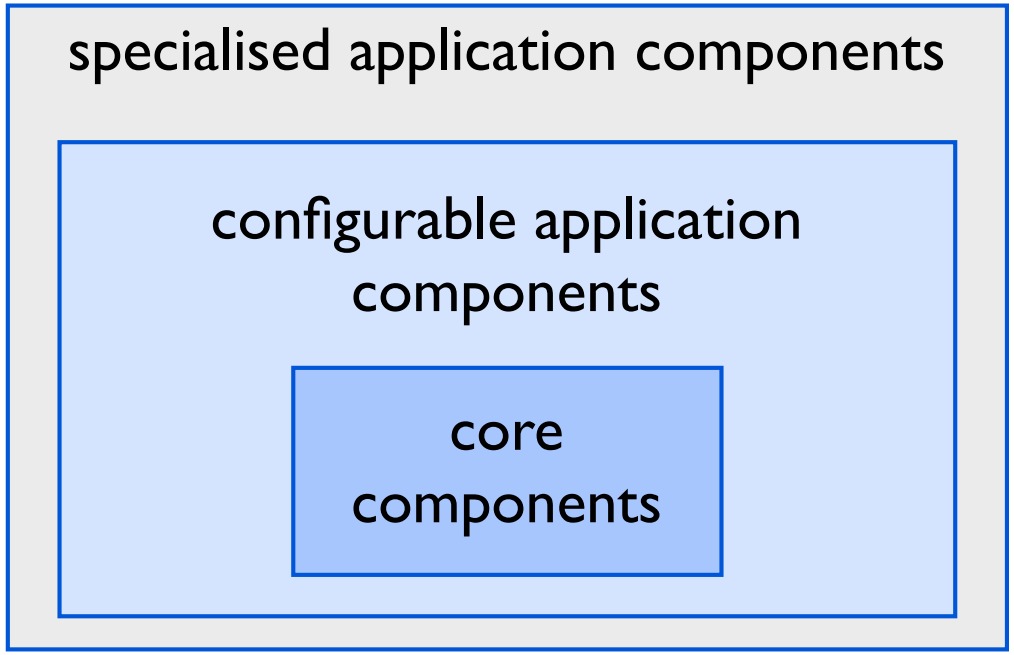

Reuse
Application framework
Product lines
Domain specific languages

software product line

- when company needs to support many similar (but not identical) systems
- e.g. printer manufacturer
  - every printer has its own control software
  - control software very similar
  - so make core product - adapt for each printer

software product line

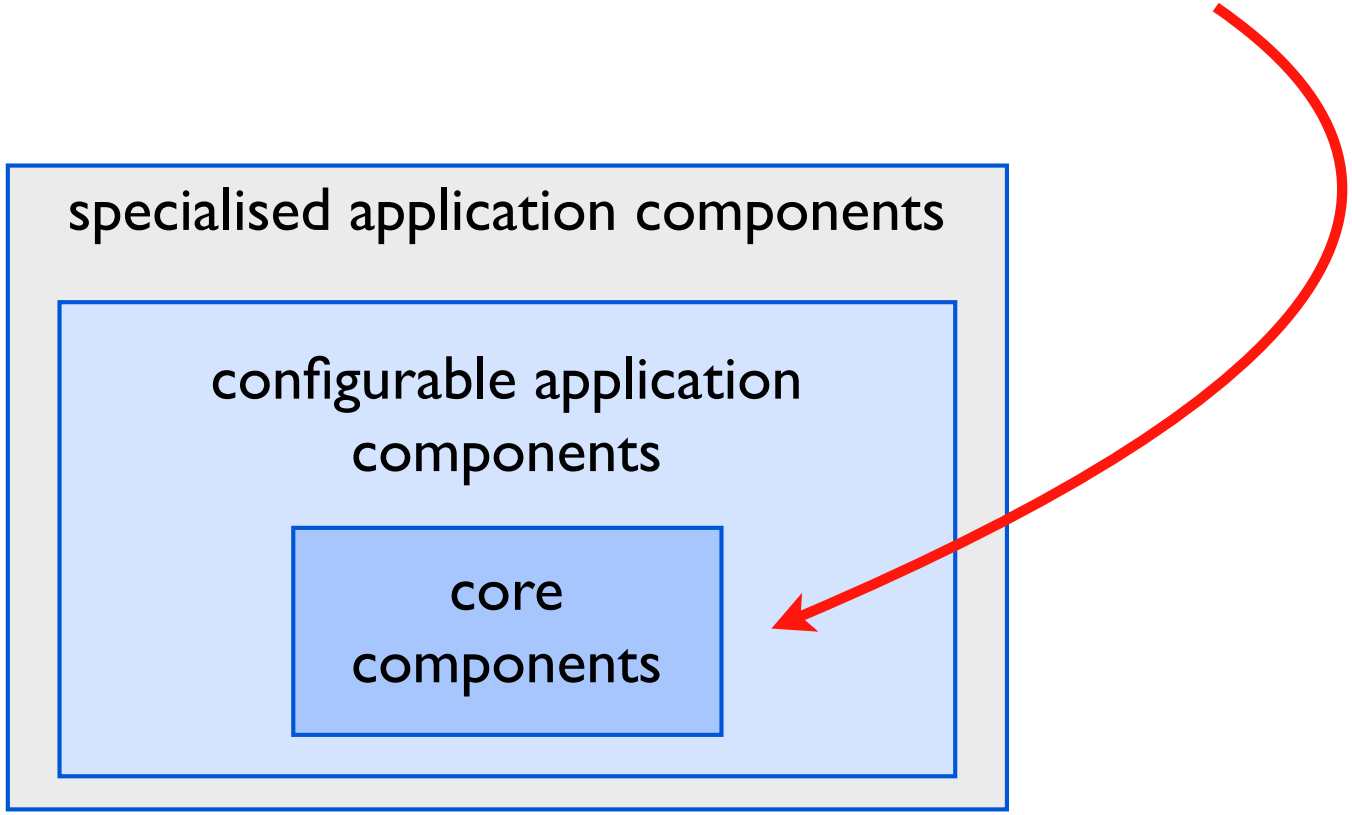
- common architecture + shared components + some specialisation



software product line

Software "evolves"

What does this mean for core components?



Application Frameworks vs. Software Product Lines

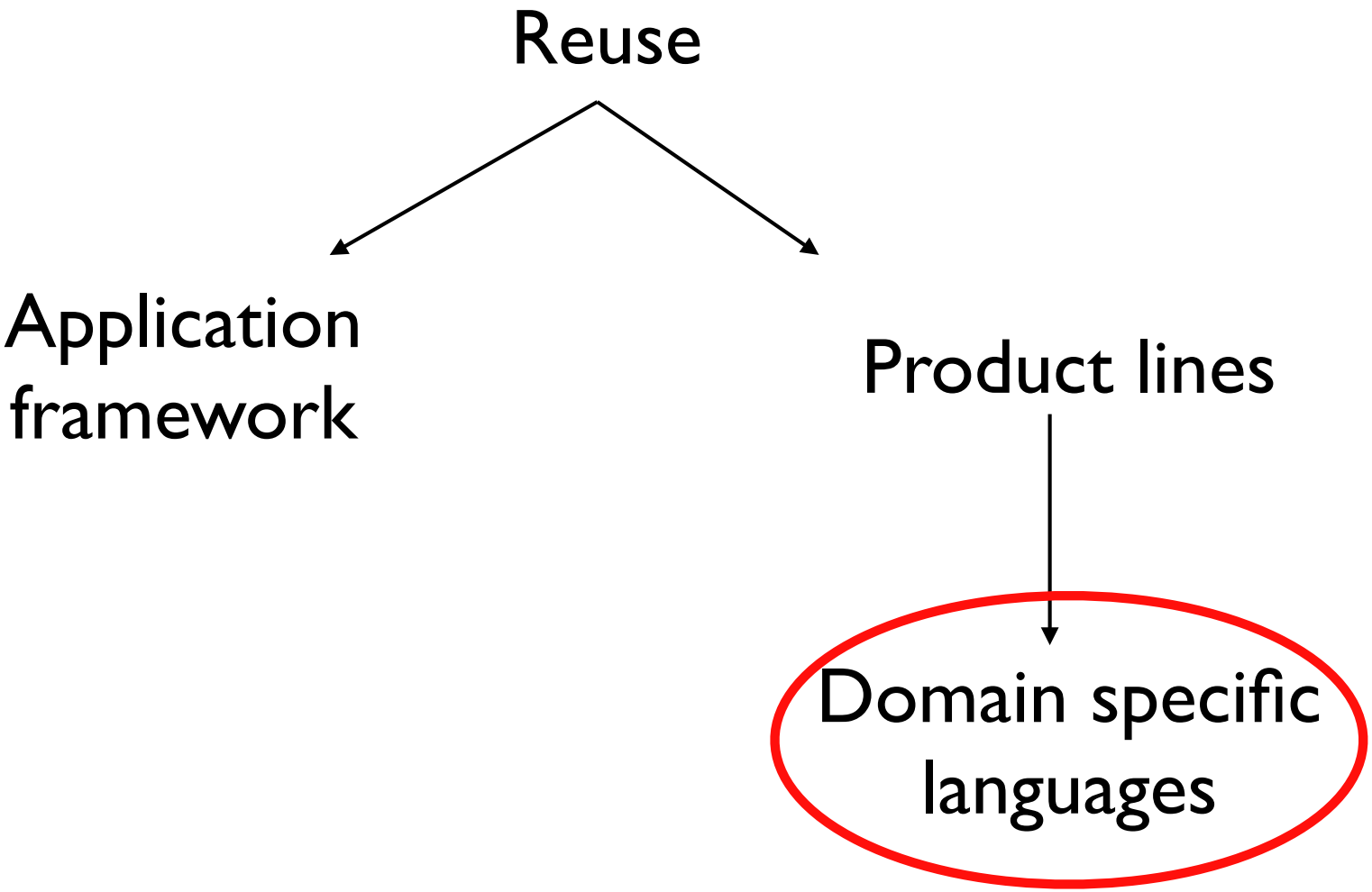
Application Frameworks	Software Product Lines
rely on OO: inheritance, polymorphism, . . .	use any suitable technique
provide technical support	provide domain-specific support
software-oriented	often hardware-oriented, e.g., family of printer drivers
shared by different organisations	developed and maintained by one organisation

It may be a good idea to base software product lines on application frameworks.

How to Configure or Adapt a Software Product Line?

How to Configure or Adapt a Software Product Line?

- One possibility: use a domain-specific language (DSL)
- encode the domain knowledge in the DSL
- implement functionality of the product in the DSL


Reuse
Application framework
Product lines
Domain specific languages

domain-specific language

- restricted programming language to make “saying things” in your domain easier
- in contrast with general purpose programming language like C++
- examples:
  - SQL
  - regular expressions
  - HTML
  - MATLAB ?
  - R ?

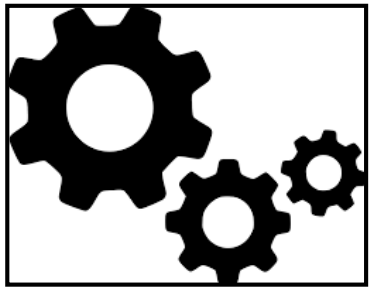
domain-specific language

here's a program in a DSL we just made up for spatial reasoning:

p: Point2D
c: Circle
p inside c
p outside c
has solution?

domain-specific language

p: Point2D
c: Circle
p inside c
p outside c
has solution?



Answer: no

we could write an **interpreter** in some other general-purpose language (e.g. C++) that can parse, analyse, and execute programs in our DSL

domain-specific language

p: Point2D
c: Circle
p inside c
p outside c
has solution?

```
AbstractQsExpression* value_interval_exterior(QsExp
{
    //- (x1 < x2 - w2/2) or (x2 + w2/2 < x1)

    GiNaC::ex left  = *x1 - *x2 + *w2*0.5;
    GiNaC::ex right = *x2 + *w2*0.5 - *x1;

    AbstractQsExpression* e1 = fc.Poly(left,  Polyn
    AbstractQsExpression* e2 = fc.Poly(right, Polyn

    return fc.Or(e1,e2);
}
```

we write a **translator** that parses our DSL programs and generates a corresponding program in a general-purpose programming language (e.g. C++)

domain-specific language

p: Point2D
c: Circle
p inside c
p outside c
has solution?

```
AbstractQsExpression* value_interval_exterior(QsE
{
    //- (x1 < x2 - w2/2) or (x2 + w2/2 < x1)

    GiNaC::ex left  = *x1 - *x2 + *w2*0.5;
    GiNaC::ex right = *x2 + *w2*0.5 - *x1;

    AbstractQsExpression* e1 = fc.Poly(left,  Polyn
    AbstractQsExpression* e2 = fc.Poly(right, Polyn

    return fc.Or(e1,e2);
}
```

we can then compile and run this program to solve our problem written in our DSL

domain-specific language

p: Point2D
c: Circle
p inside c
p outside c
has solution?

```
AbstractQsExpression* value_interval_exterior(QsExp
{
    //- (x1 < x2 - w2/2) or (x2 + w2/2 < x1)

    GiNaC::ex left  = *x1 - *x2 + *w2*0.5;
    GiNaC::ex right = *x2 + *w2*0.5 - *x1;

    AbstractQsExpression* e1 = fc.Poly(left,  Polyn
    AbstractQsExpression* e2 = fc.Poly(right, Polyn

    return fc.Or(e1,e2);
}
```

in this case we didn't use any existing programming language for our DSL, we just made it up

domain-specific language

```
(declare-fun p () Point2D)
(declare-fun c () Circle)
(assert (inside (p c)))
(assert (outside (p c)))
(check-sat)
```

```
AbstractQsExpression* value_interval_exterior(QsExp
{
    //- (x1 < x2 - w2/2) or (x2 + w2/2 < x1)

    GiNaC::ex left  = *x1 - *x2 + *w2*0.5;
    GiNaC::ex right = *x2 + *w2*0.5 - *x1;

    AbstractQsExpression* e1 = fc.Poly(left,  Polyn
    AbstractQsExpression* e2 = fc.Poly(right, Polyn

    return fc.Or(e1,e2);
}
```

we could also use an existing programming language (e.g. this example is in SMT-LIB language)

domain-specific language

- DSL for software product line
- we want to manage range of products by only talking about (i.e. programming) the **differences** (i.e. parts that can be configured)
- idea: express a particular product configuration on higher-level of abstraction (DSL) than product programming language (e.g. C++)
- automatically generate code in product's original programming language (e.g. C++) from DSL

an interesting paper with some nice examples:

Voelter, M., & Visser, E. Product line engineering using domain-specific languages. In: 15th International Software Product Line Conference (SPLC), 2011, pp. 70-79. IEEE.