Tampereen yliopisto
Tampere University

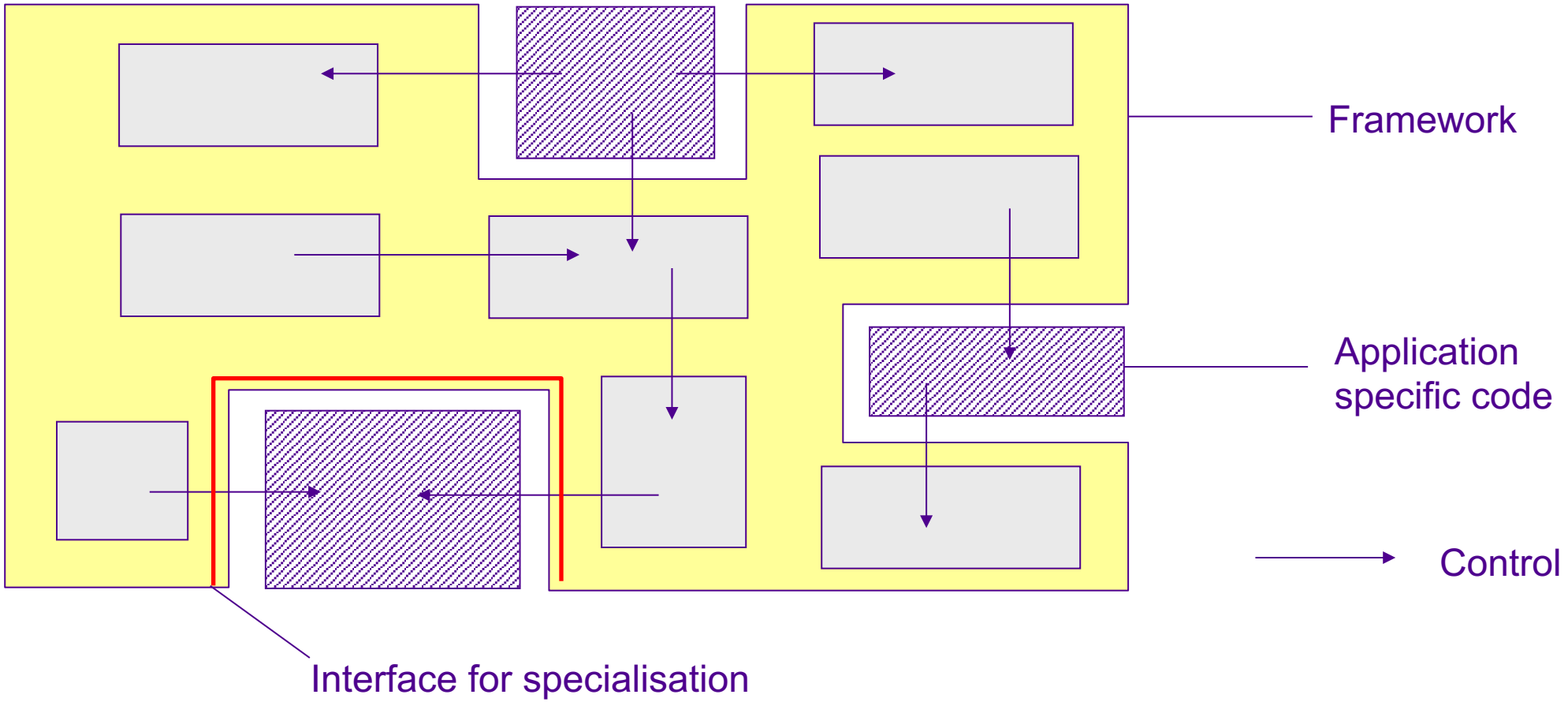
Large Scale Software Design Frameworks & Product Lines

Hannu-Matti Järvinen, David Hästbacka
Spring 2024

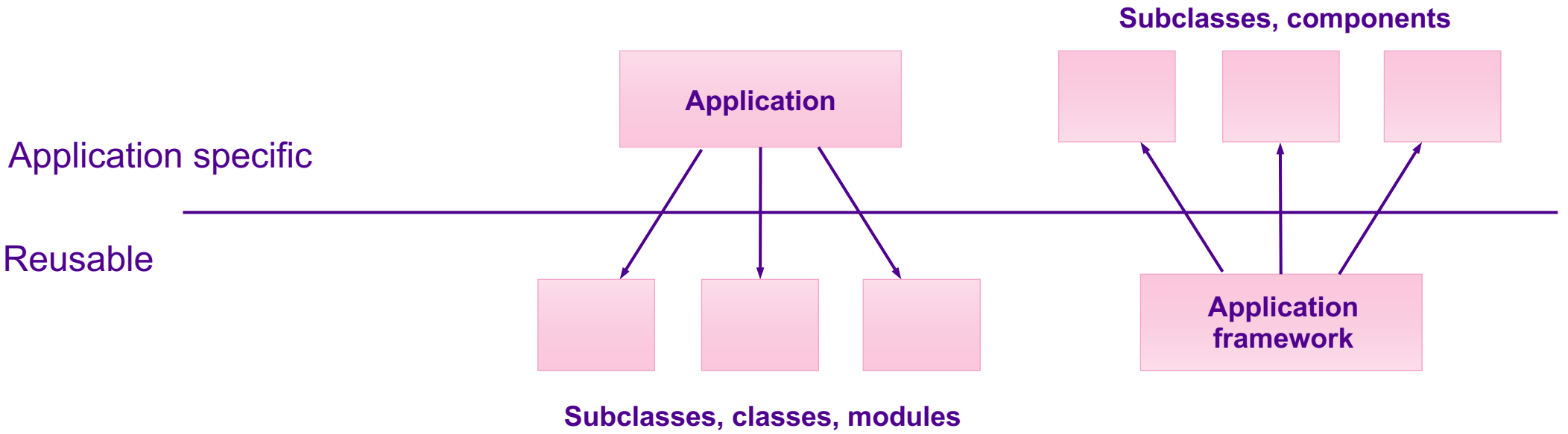
What is a (software) framework?

- Traditionally: software framework is object-oriented paradigm's way to implement product frame
  - Framework is formed of a collection of classes that implement the common architecture and functionality of a product family
  - A framework is specialised to a product
- Frameworks offer program's (or its part's) structure and implementation
- Generalised frameworks offer (a part of) body for the application

The framework is specialised to a working product



Framework vs. traditional software library: Hollywood principle



- Hollywood principle:
- Don’t call us, we call you

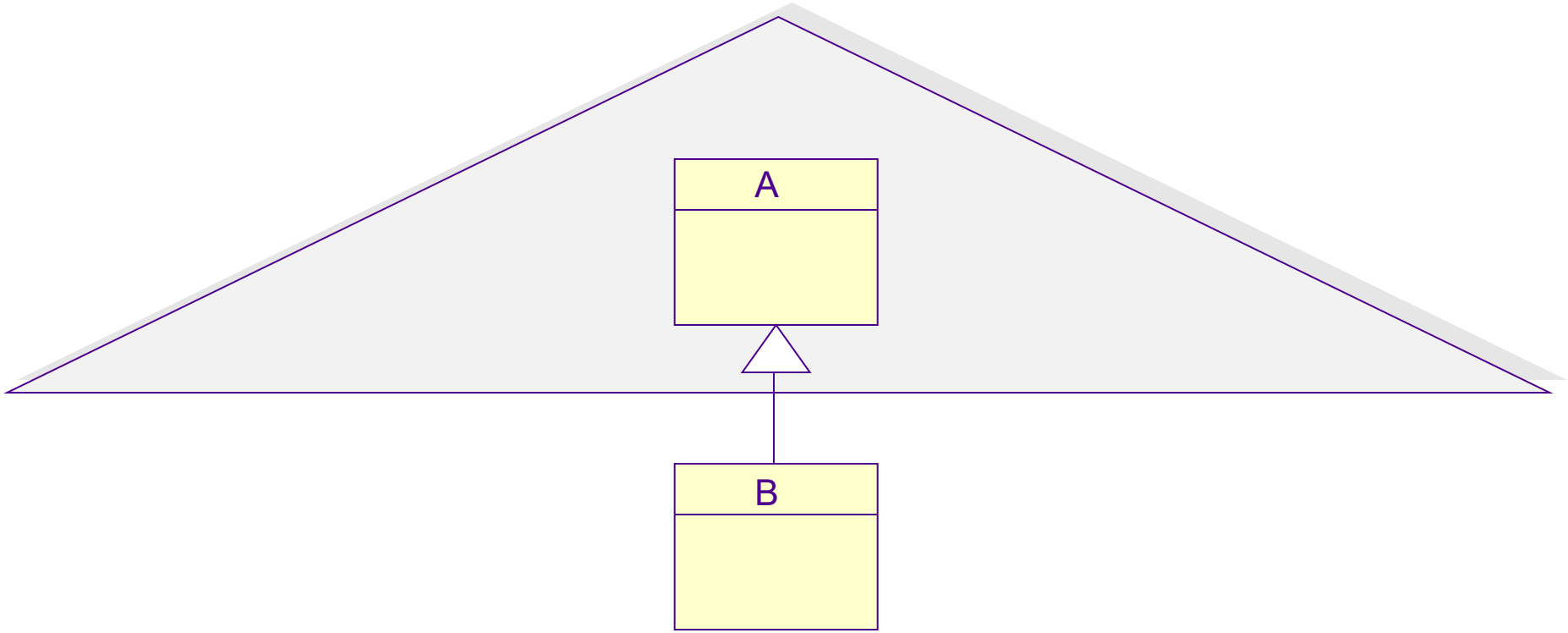
Specialisation technologies in frameworks

- Implementation of interface (~callback)
- Inheritance (~callback)
- Creation, initialisation and configuration of objects and components
- Instantiations of generic classes (templates)
- Reflexivity (e.g. class editor of the framework that can handle also attributes specialised by the application)

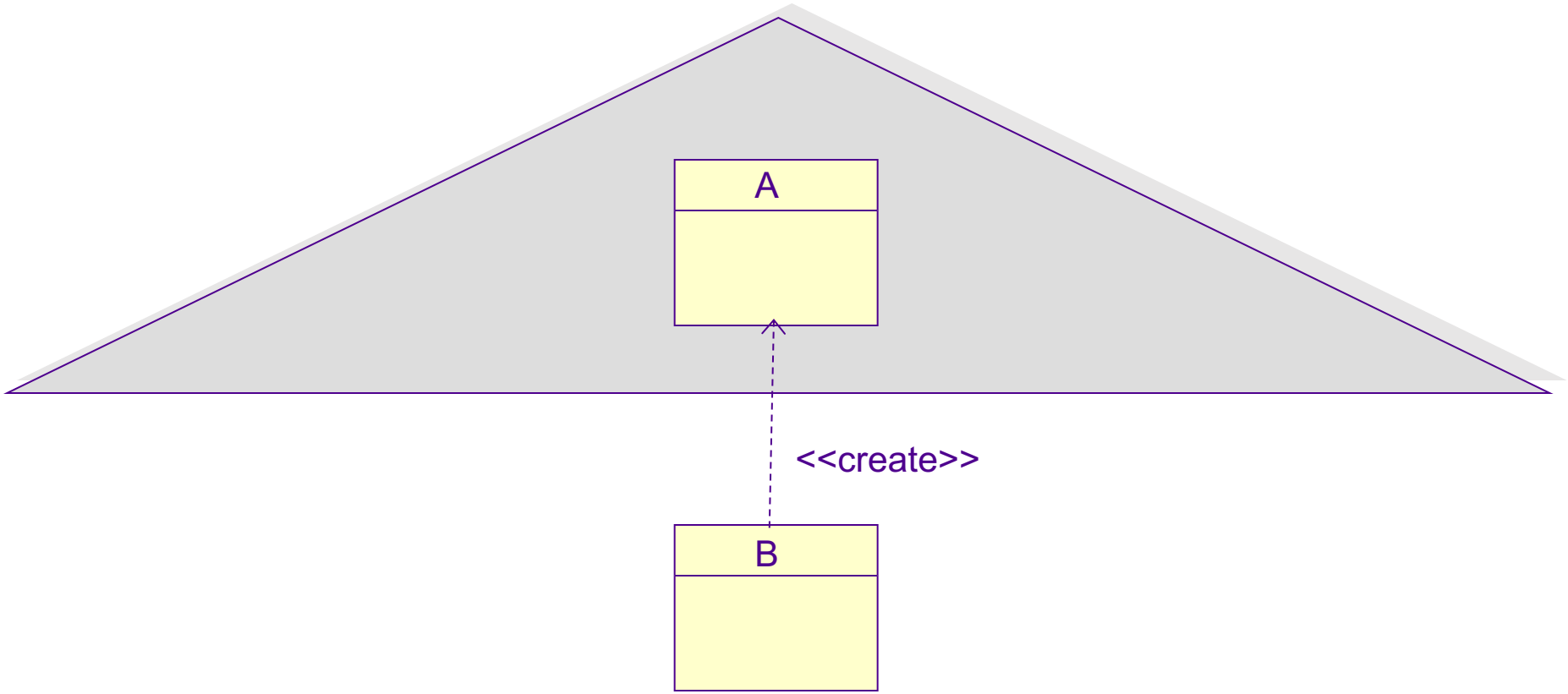
Frame types

- The result of specialisation
  - Application framework: the result is an application
  - Framelet (component framework): result is a component
  - Hierarchical framework: the result is a new framework
- The mechanism of specialisation
  - White box framework: specialisation by inheritance and overloading methods
  - Black box framework: specialisation by instantiation (+parameters) and initialisation configuration
  - Plugin framework: specialisation by implementation of interfaces.

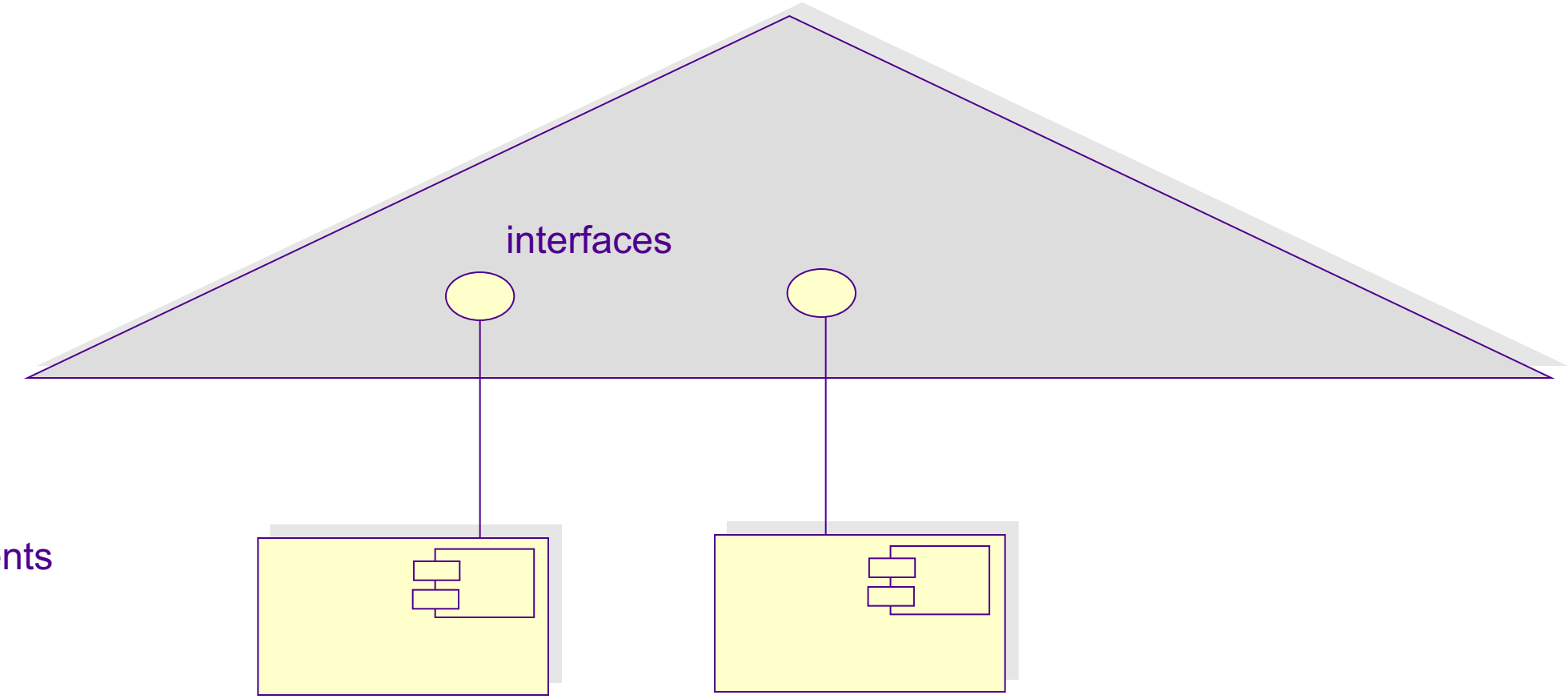
White-box frameworks (muunneltava kehys)



Black-box frameworks (koottava kehys)



Plug-in frameworks



Partitioning of frameworks

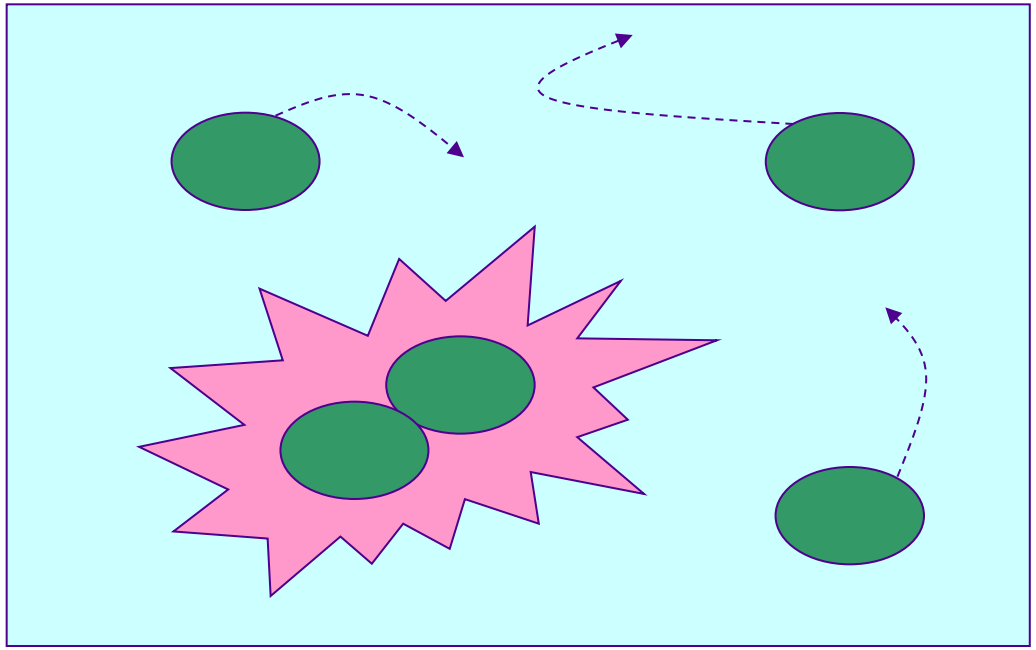
- Conceptual model based approach
- Component-based approach
- Layering: hierarchical frameworks

Concept model based approach

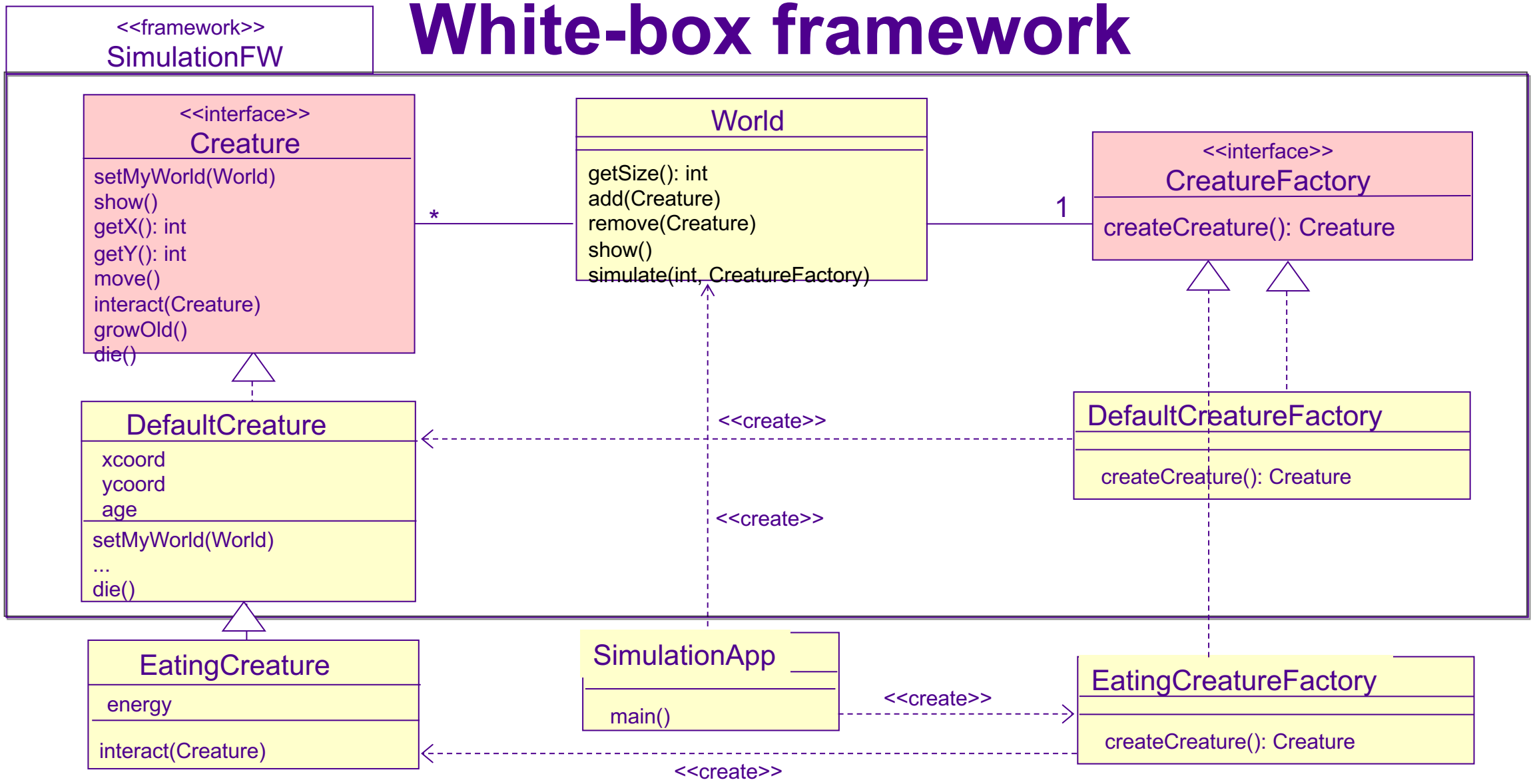
- ”OO architectural style”

1. Make conceptual model of the application area
2. Find and add generalisations (base classes) to the conceptual model
3. Convert conceptual model to a class model, add default implementations, interfaces
4. Identify variation point from the class model
5. Design the implementation of a variation point (e.g. Applying design patterns)

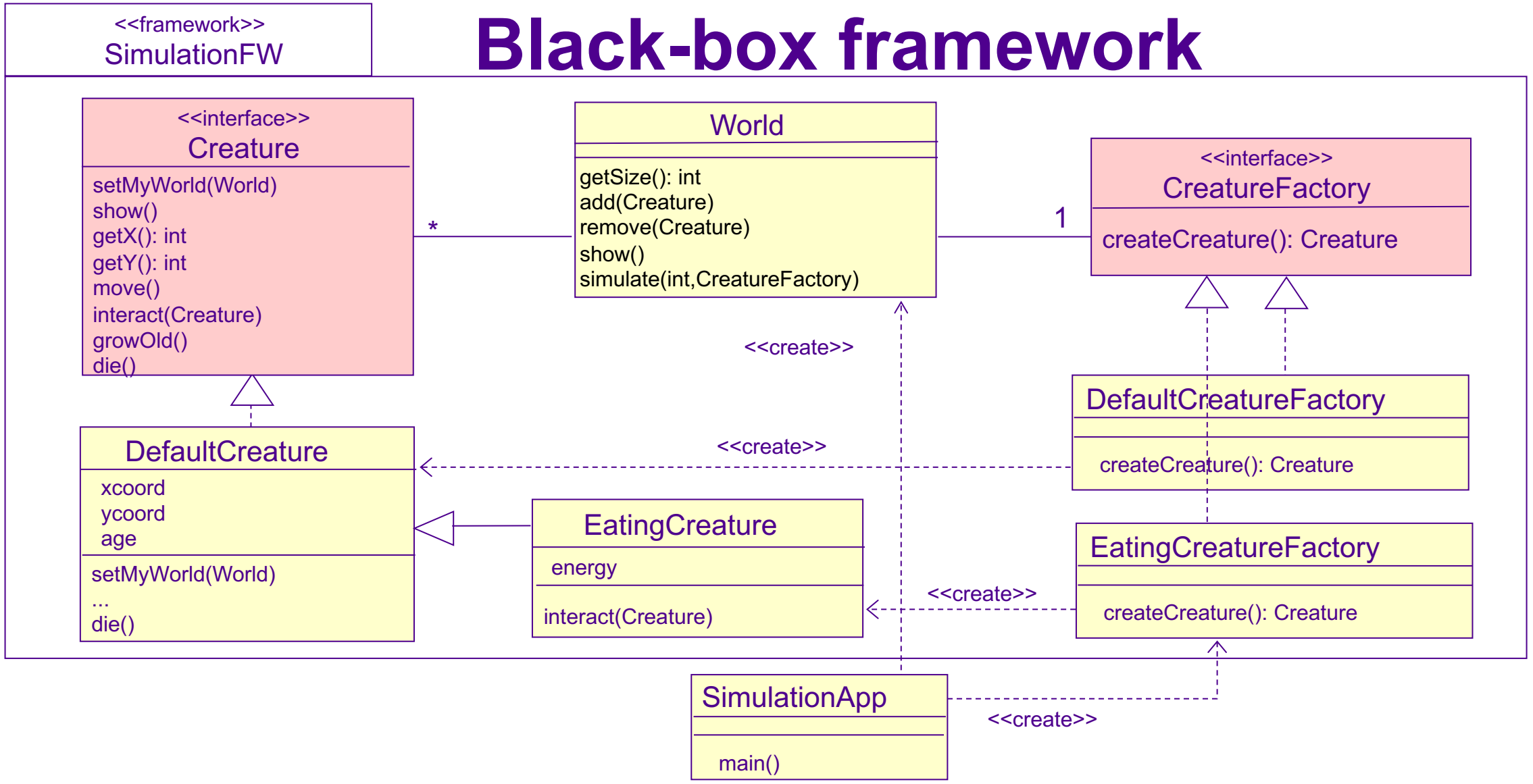
Example: simulation framework



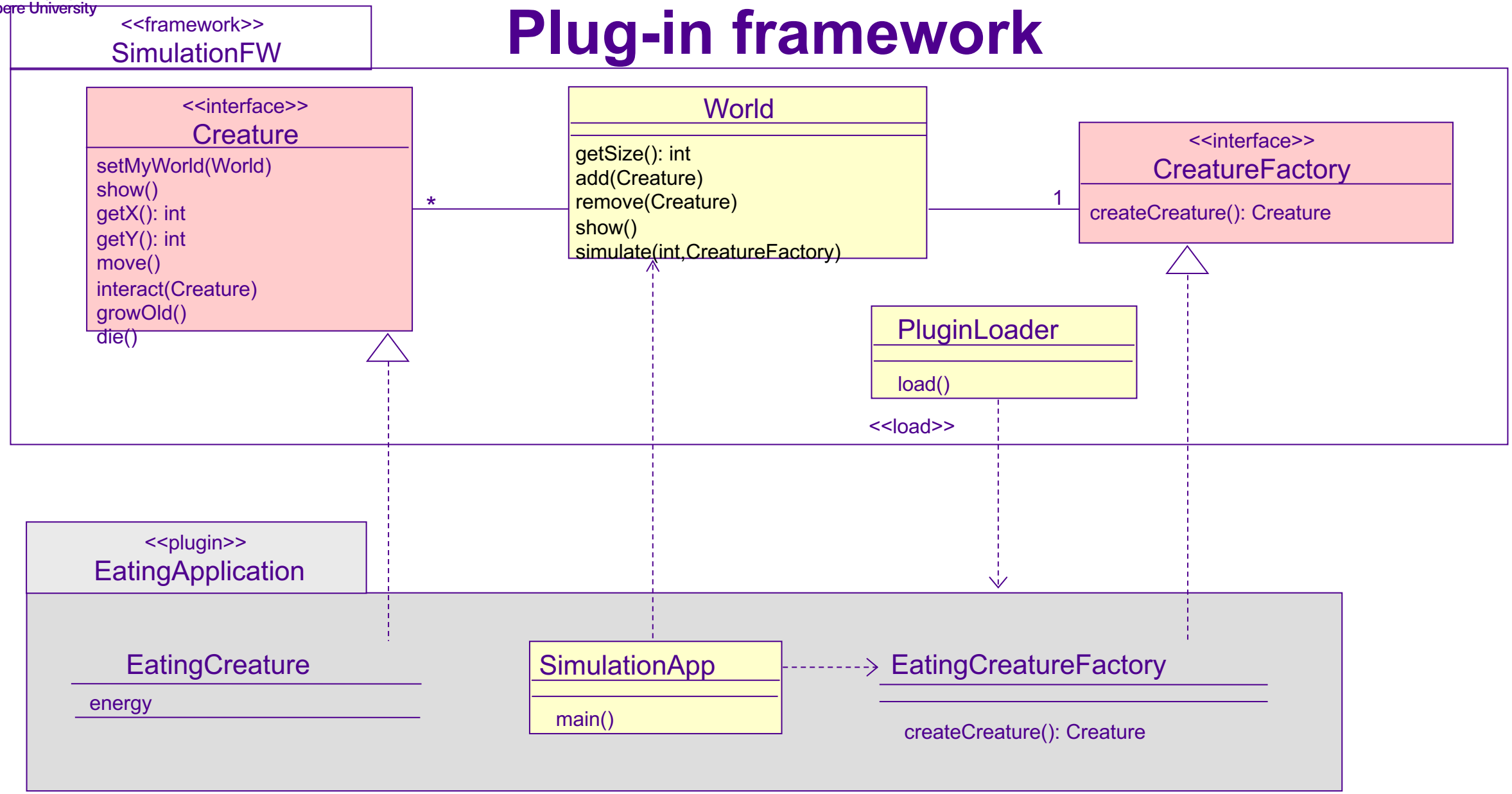
White-box framework



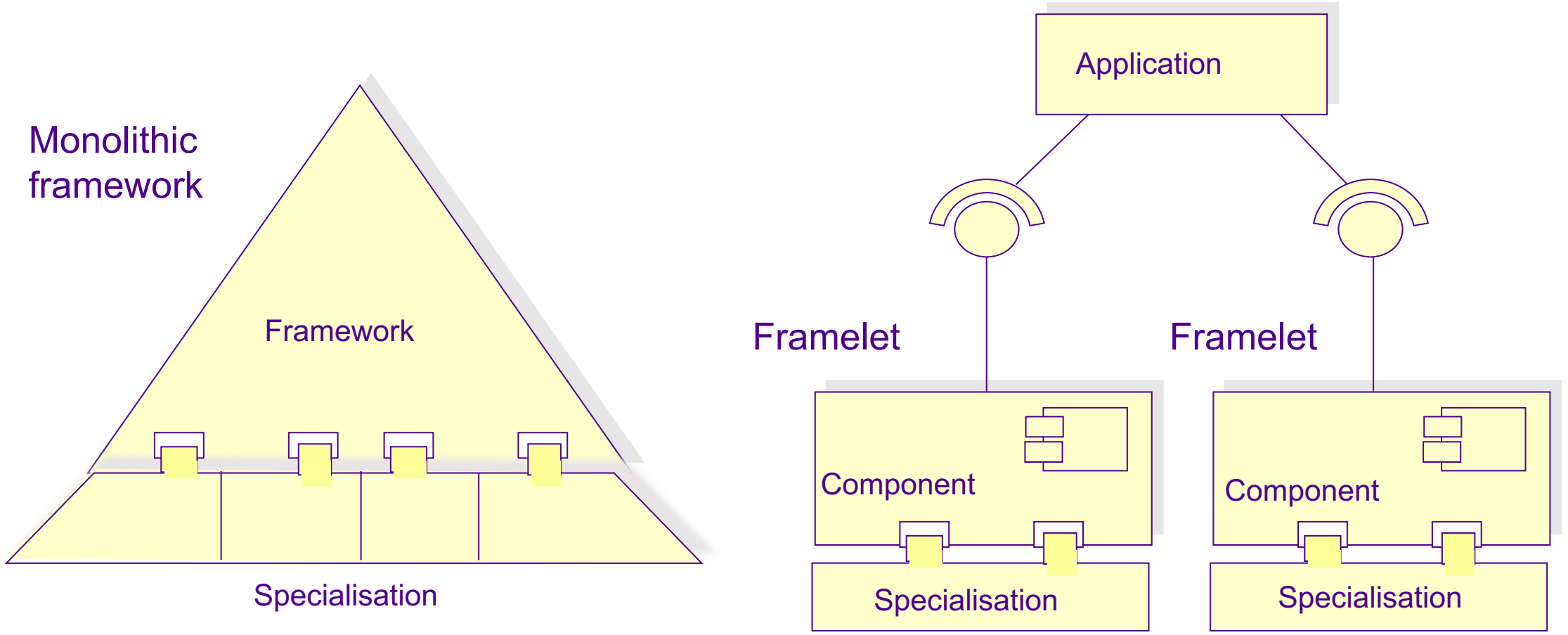
Black-box framework



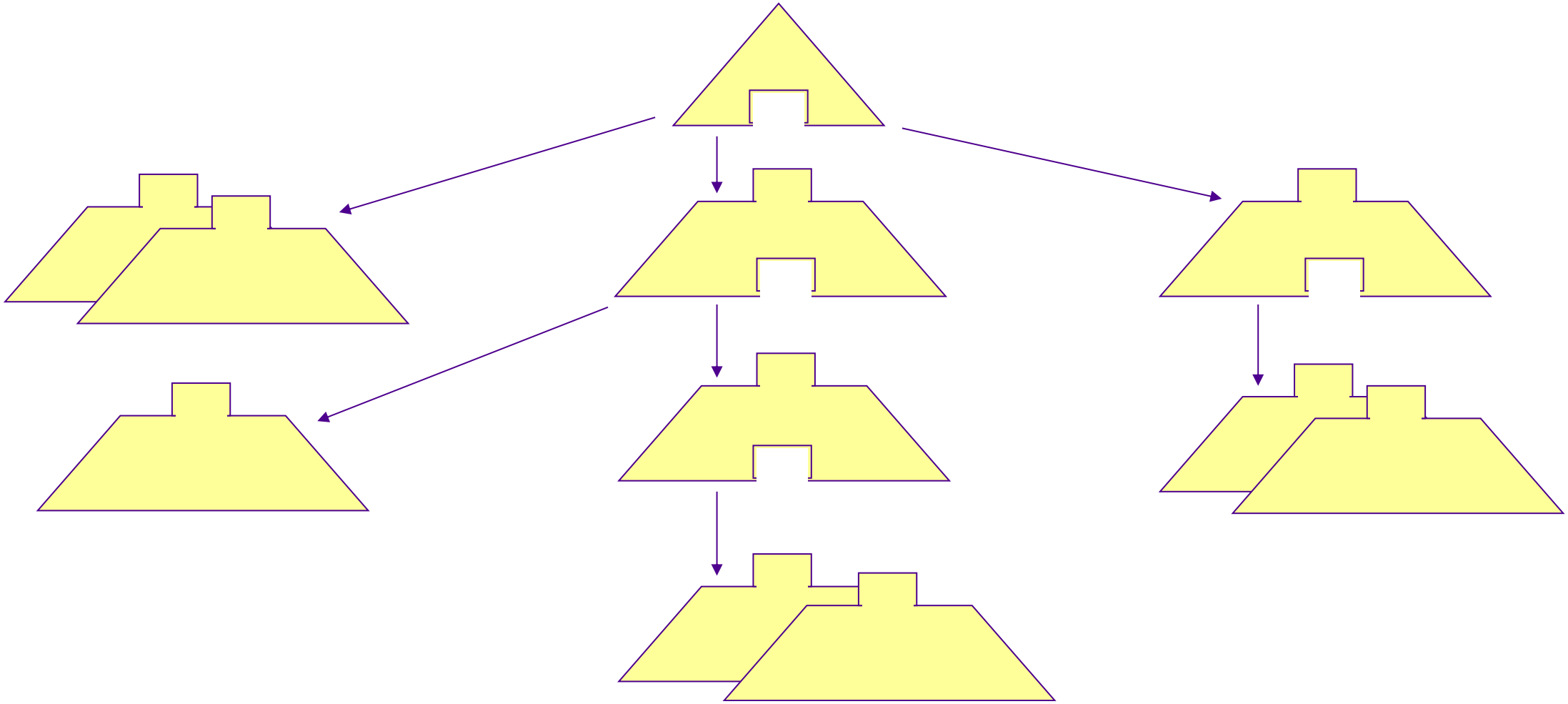
Plug-in framework



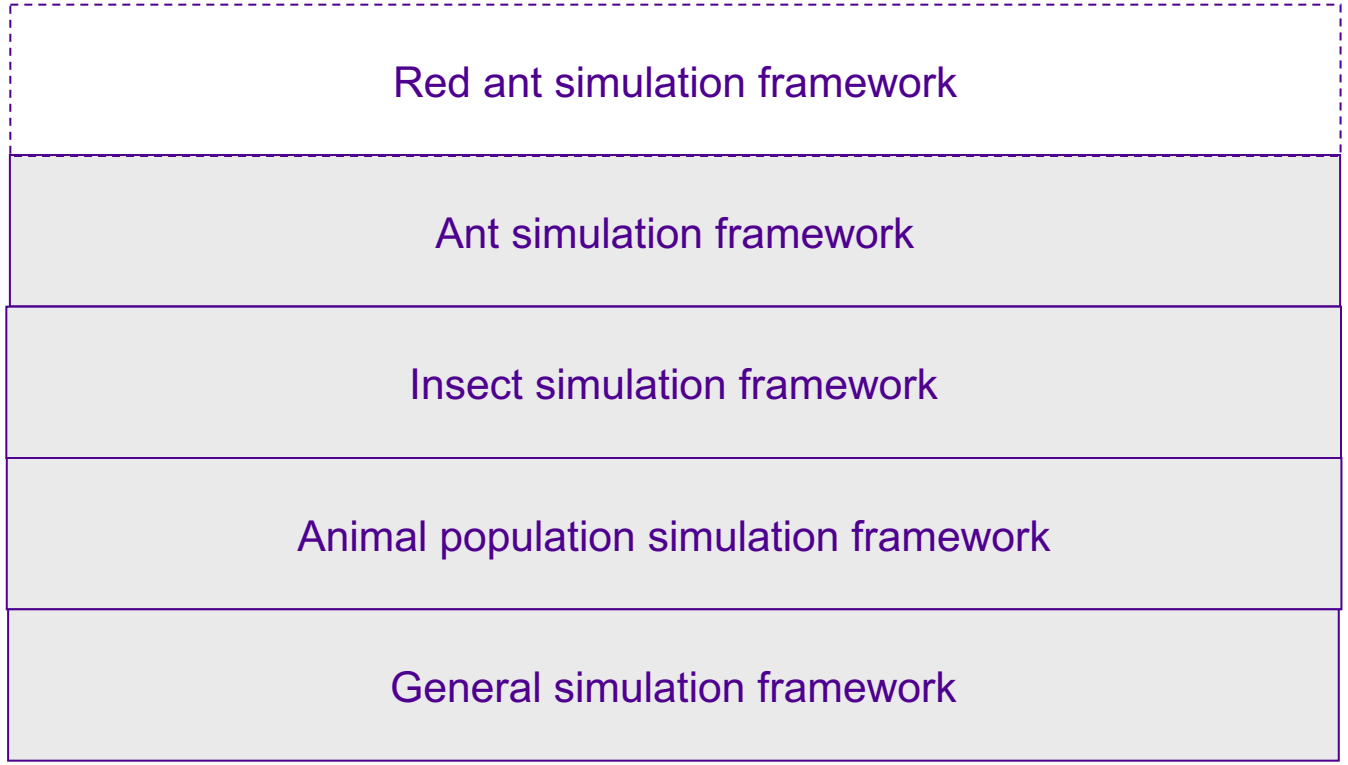
Component-based approach: monolithic frameworks vs. framelets



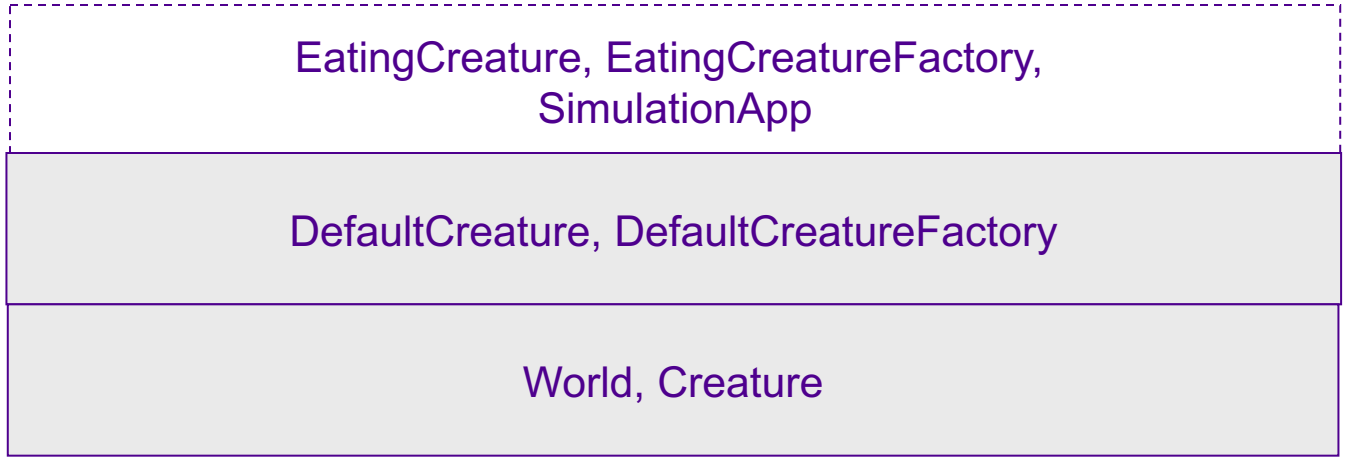
Hierarchical frame



Hierarchical frame: general simulation framework



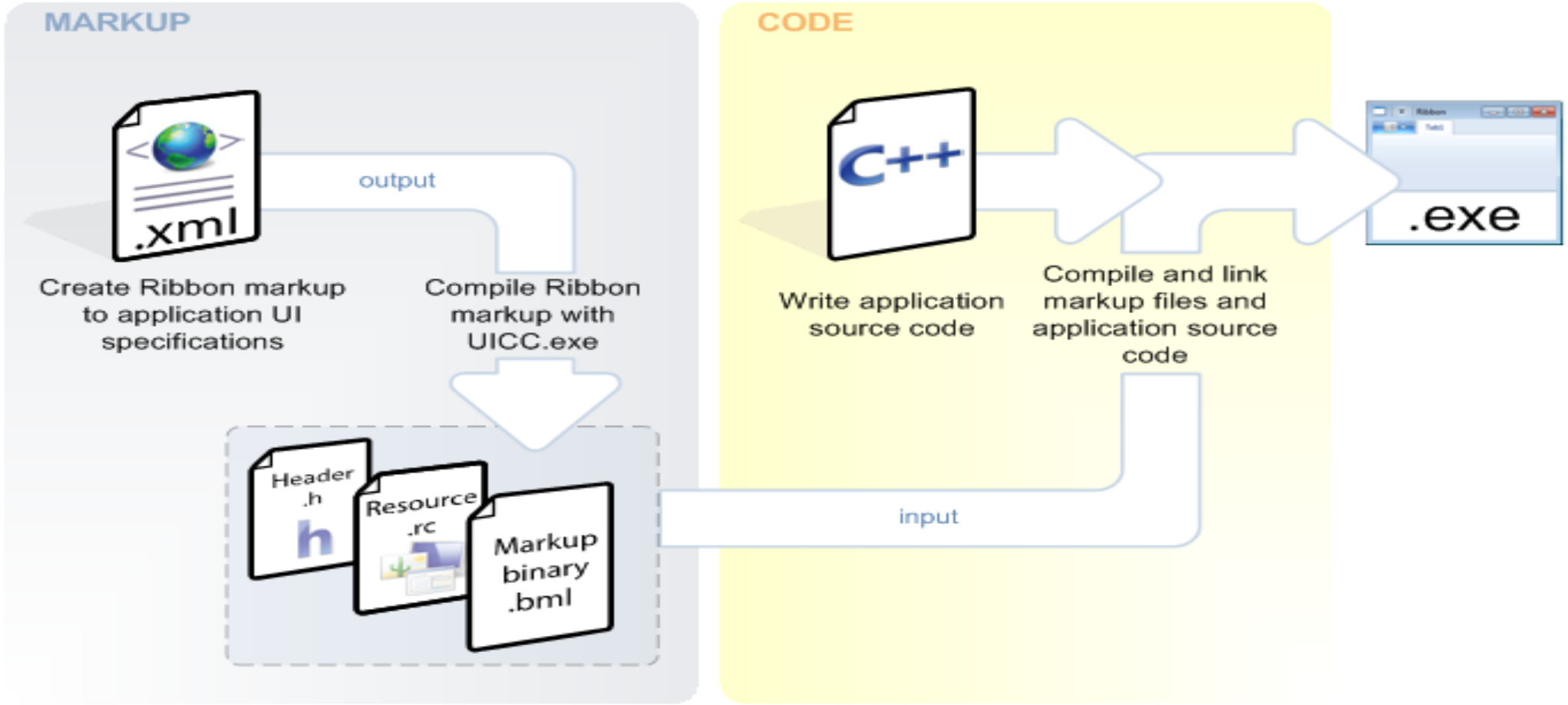
Hierarchical frame: example framework



Windows Ribbons: Example of a framework and its documentation

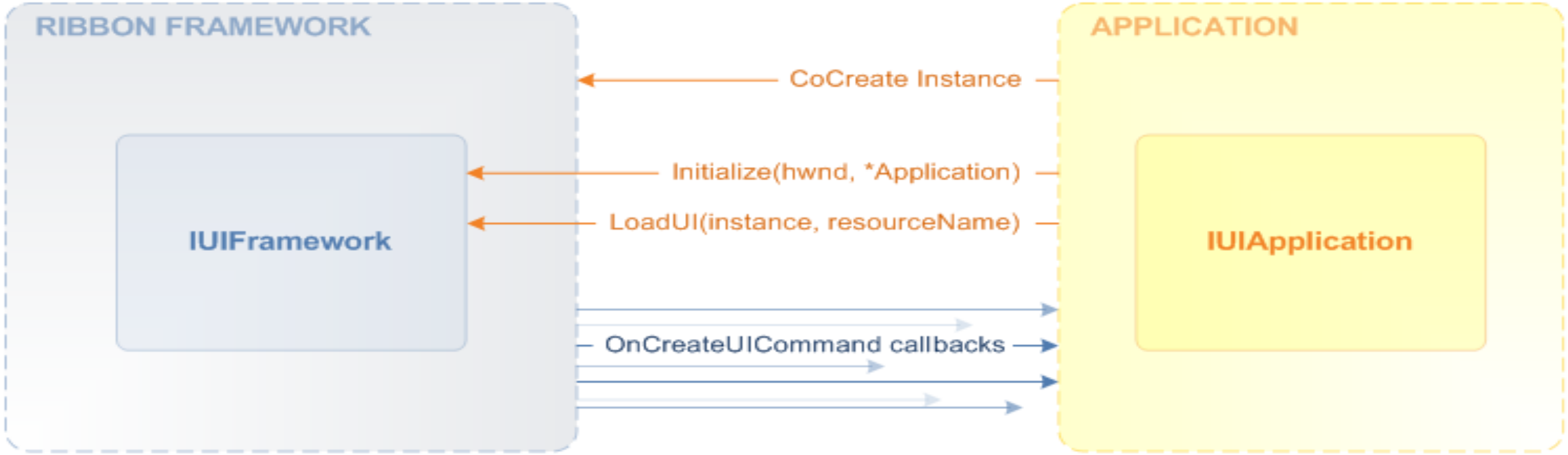
- Windows and Office 2007 –style user interfaces
- How documented:
  - What is it, is it worth of using: https://msdn.microsoft.com/en-us/library/windows/desktop/dn742393%28v=vs.85%29.aspx
  - Application instructions: https://msdn.microsoft.com/en-us/library/windows/desktop/dd316924%28v=vs.85%29.aspx
  - Example: http://www.codeproject.com/Articles/119319/Windows-Ribbon-Framework-in-Win-C-Application

Framework example, Windows Ribbons



Cont.

Command and Control structure



https://msdn.microsoft.com/en-us/library/windows/desktop/dd742866%28v=vs.85%29.aspx

Examples of frameworks

- Example code: Knockout.js
  - Knockout and tutorial 2:
  - http://jsfiddle.net/nfzycs4k/
- Javascript frameworks: less specialisation, more creating own instances and relying on the services provided by the framework.
  - http://en.wikipedia.org/wiki/Comparison_of_JavaScript_frameworks
- Web application frameworks, Struts, Django, Ruby on Rails, Vaadin…
- Games, game engines, physics, etc.
- GUI frameworks: Qt, …
- Eclipse…

Javascript frameworks

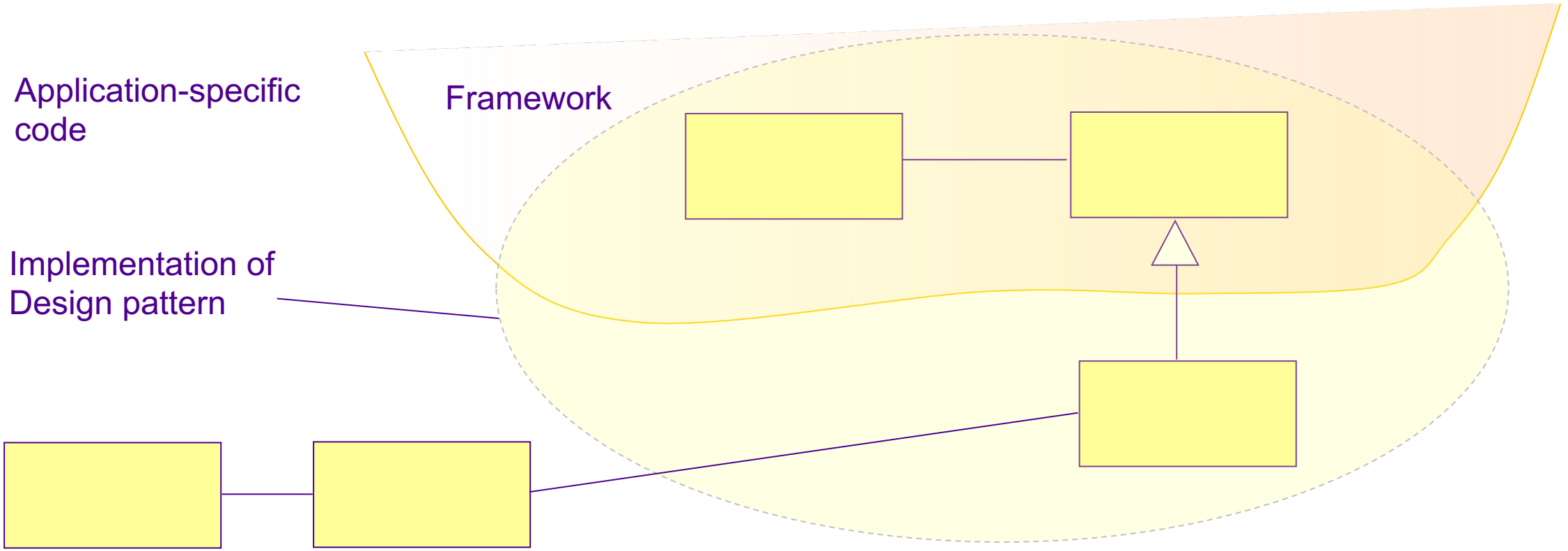
Javascript frameworks:

http://www.allenpike.com/2015/javascript-framework-fatigue/
http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html
http://www.developereconomics.com/feature-comparison-of-4-popular-js-mv-frameworks/
http://www.developereconomics.com/comparison-4-popular-javascript-mv-frameworks-part-2/

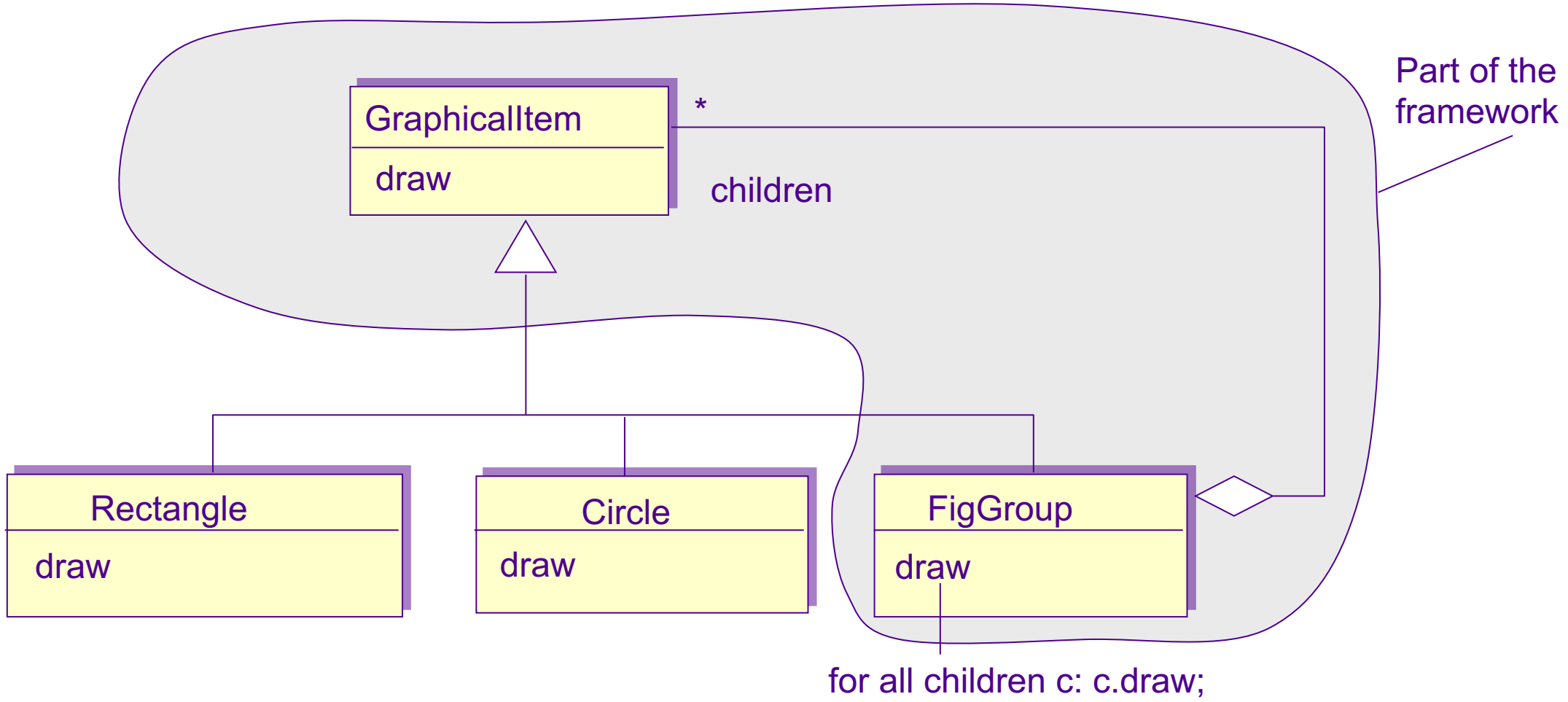
Frameworks and patterns

- Flexibility of the system can be increased by design patterns both in regular applications (maintainability, portability) and frameworks (reusability).
- Well defined object-oriented application can often be understood as a specialisation of an (implicit) framework.

Frameworks and design patterns as specialisation interface



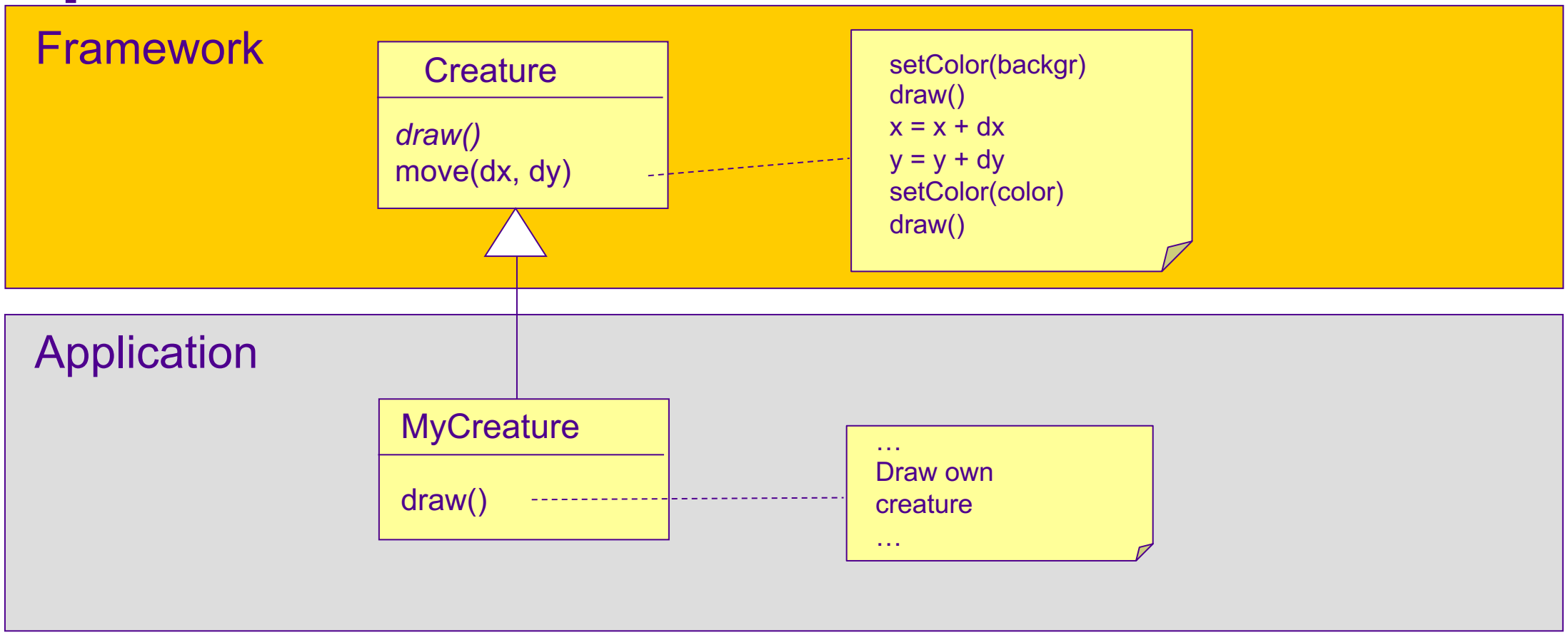
Design patterns as specialisation interface of a framework



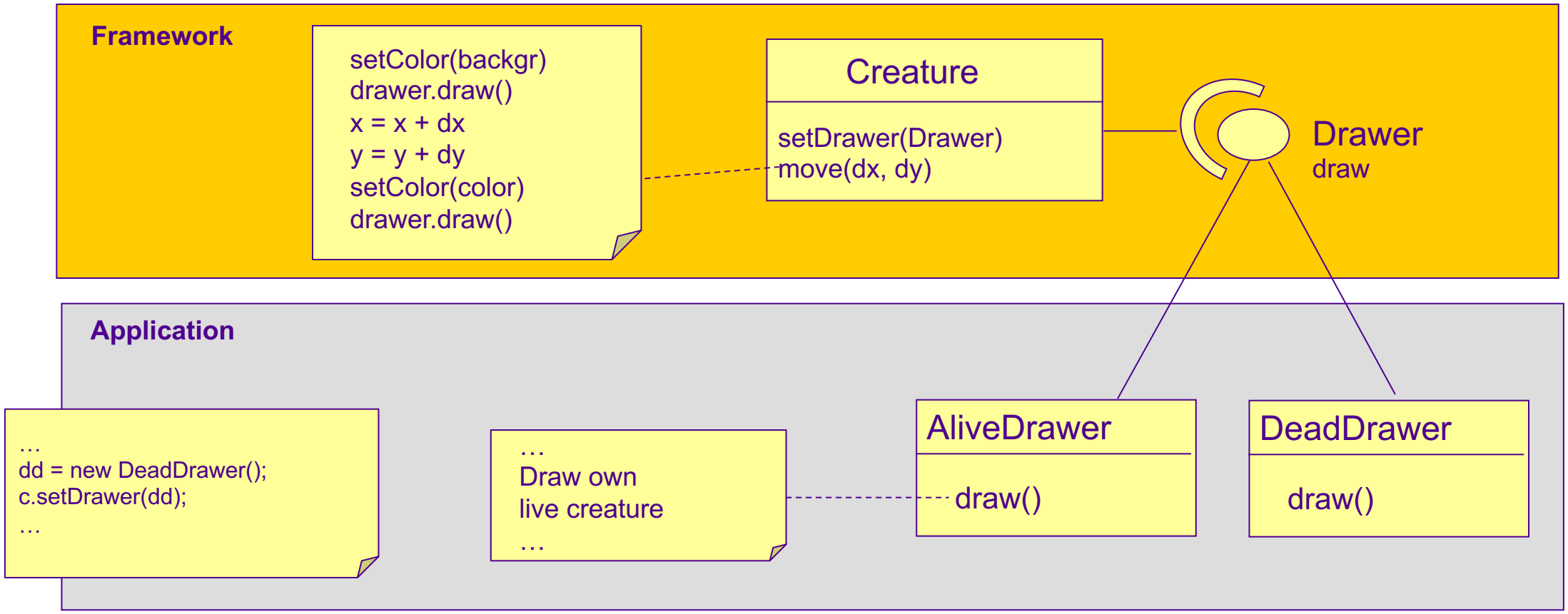
Typical GoF design patterns used in frameworks

- Template Method (operaatiorunko):
  - Problem: Static application-specific variation of a method
- Strategy
  - Problem: Dynamic application-specific variation of method (during the lifetime of the master object)
- Decorator (kuorruttaja)
  - Problem: How to give a possibility to include dynamically application-specific functionality to a given component of a framework
- Abstract Factory, Factory Method (tehtaat)
  - Problem: How to create consistently objects of a given application-specific class collection in a framework?
- Observer (tarkkailija)
  - Problem: How to give a possibility to include application-specific functionality dynamically into a component of the framework

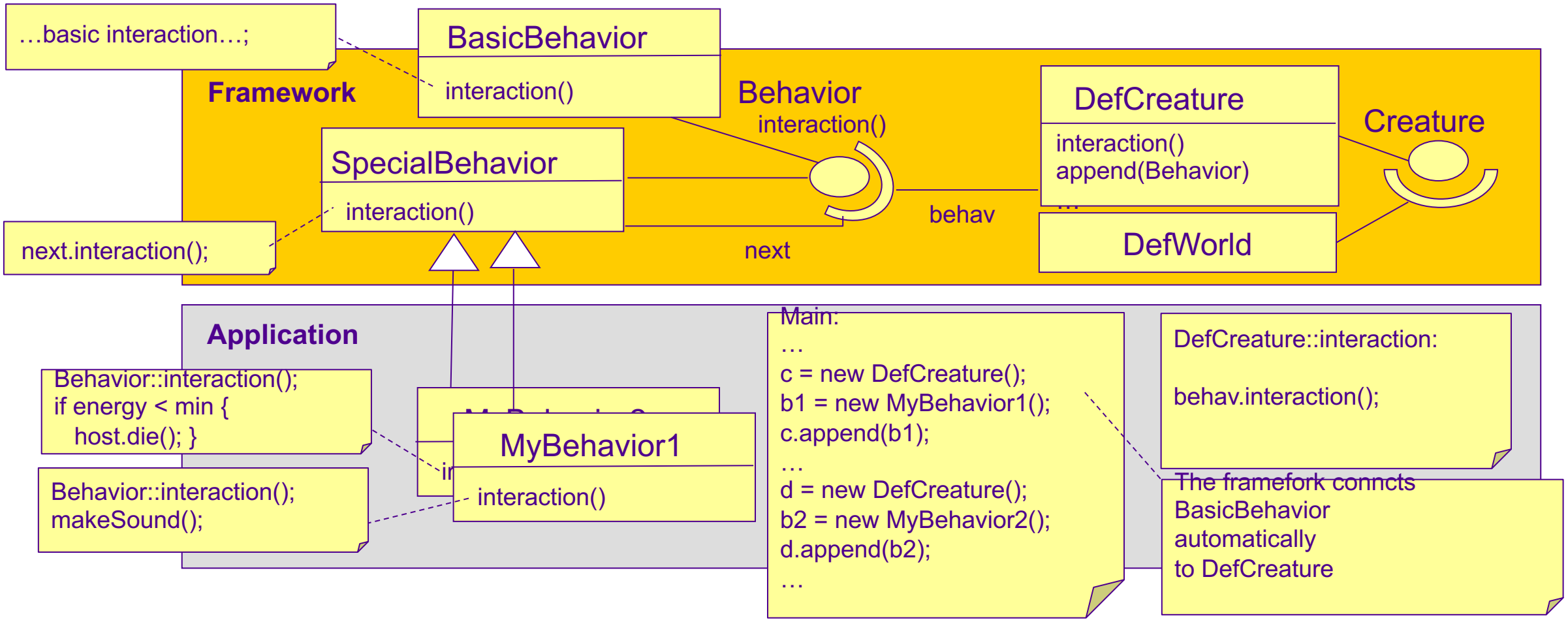
Template Method



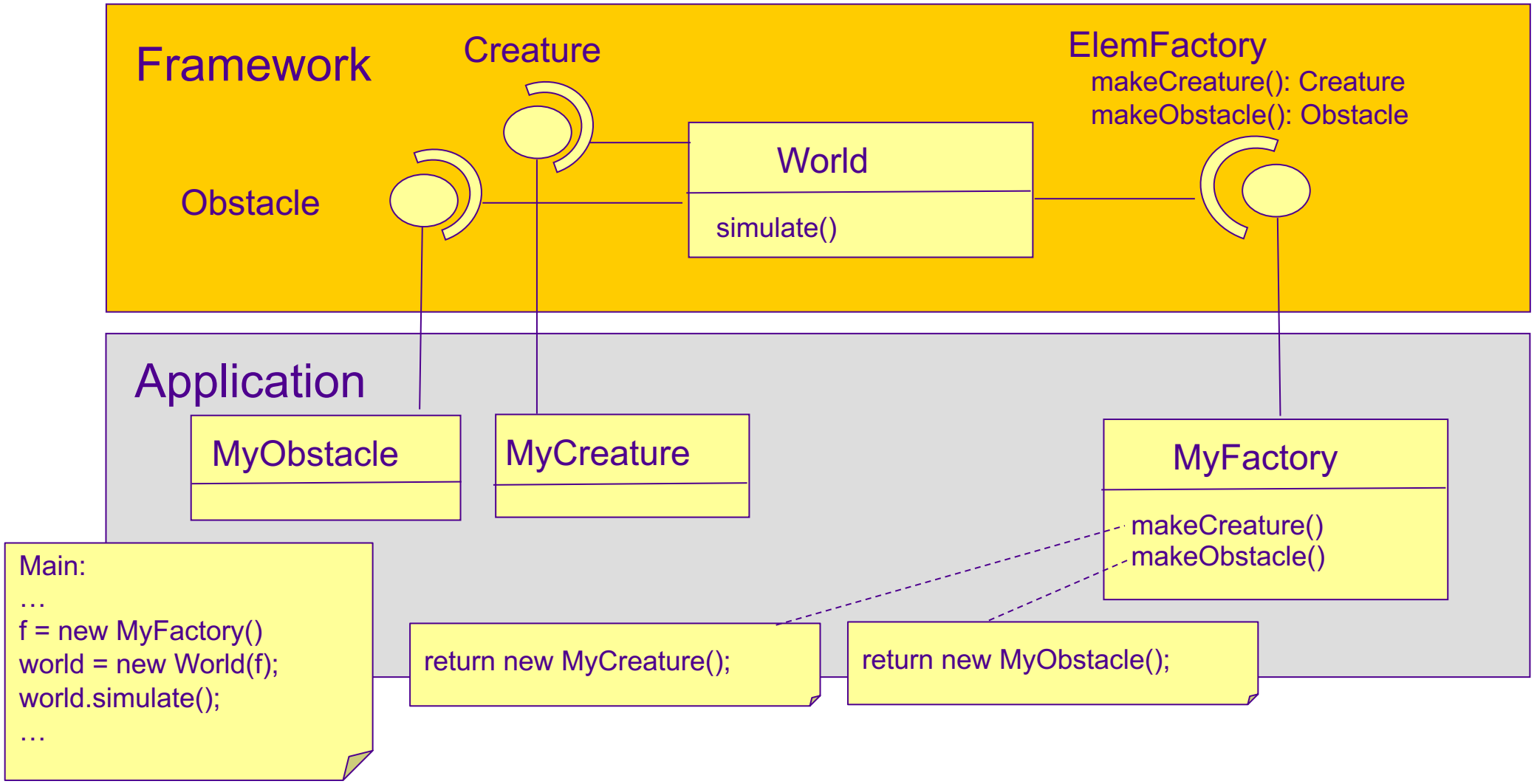
Strategy


Framework
setColor(backgr)
drawer.draw()
x = x + dx
y = y + dy
setColor(color)
drawer.draw()
Creature
setDrawer(Drawer)
move(dx, dy)
Drawer
draw
Application
…
dd = new DeadDrawer();
c.setDrawer(dd);
…
…
Draw own
live creature
…
AliveDrawer
draw()
DeadDrawer
draw()

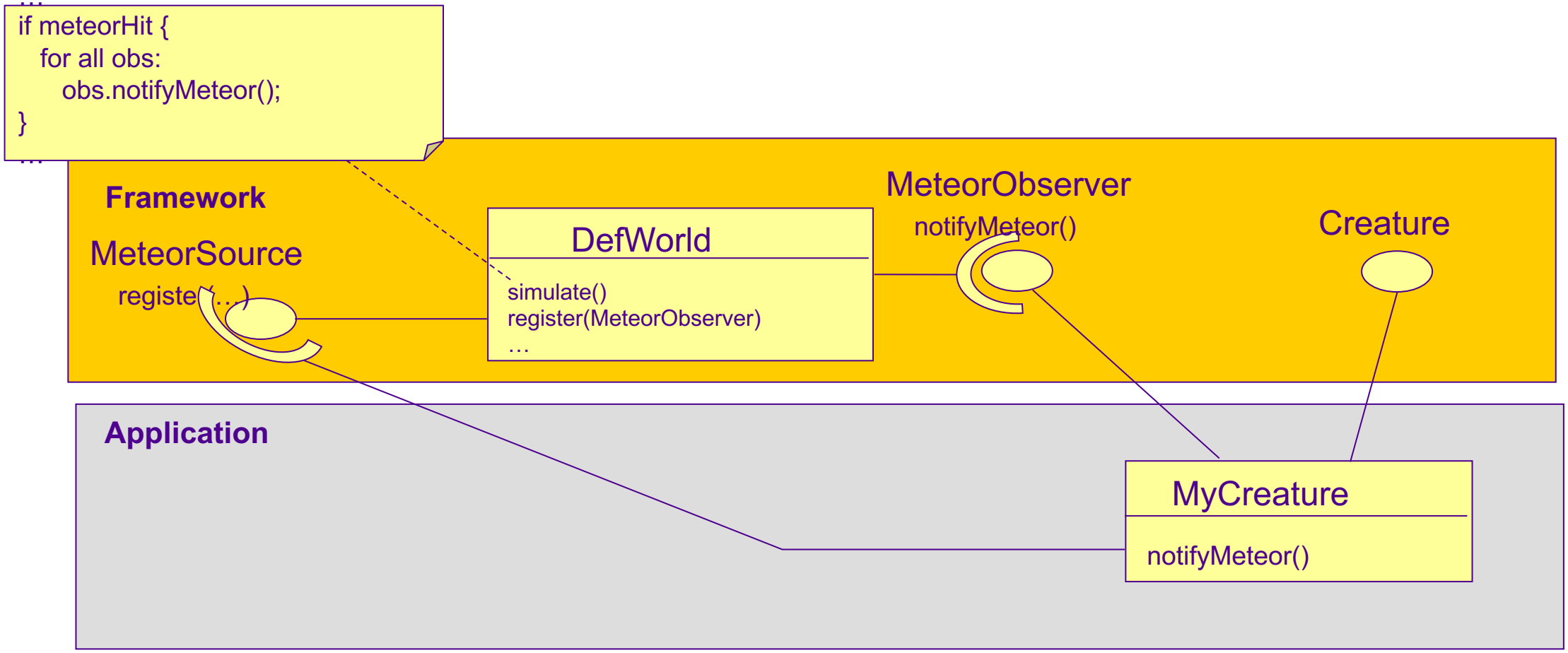
Decorator


…basic interaction…;
BasicBehavior
interaction()
Framework
Behavior
interaction()
DefCreature
interaction()
append(Behavior)
Creature
SpecialBehavior
interaction()
next.interaction();
next
behav
DefWorld
Application
Behavior::interaction();
if energy < min {
host.die(); }
Behavior::interaction();
makeSound();
MyBehavior1
interaction()
Main:
…
c = new DefCreature();
b1 = new MyBehavior1();
c.append(b1);
…
d = new DefCreature();
b2 = new MyBehavior2();
d.append(b2);
…
DefCreature::interaction:
behav.interaction();
The framefork conncts
BasicBehavior
automatically
to DefCreature

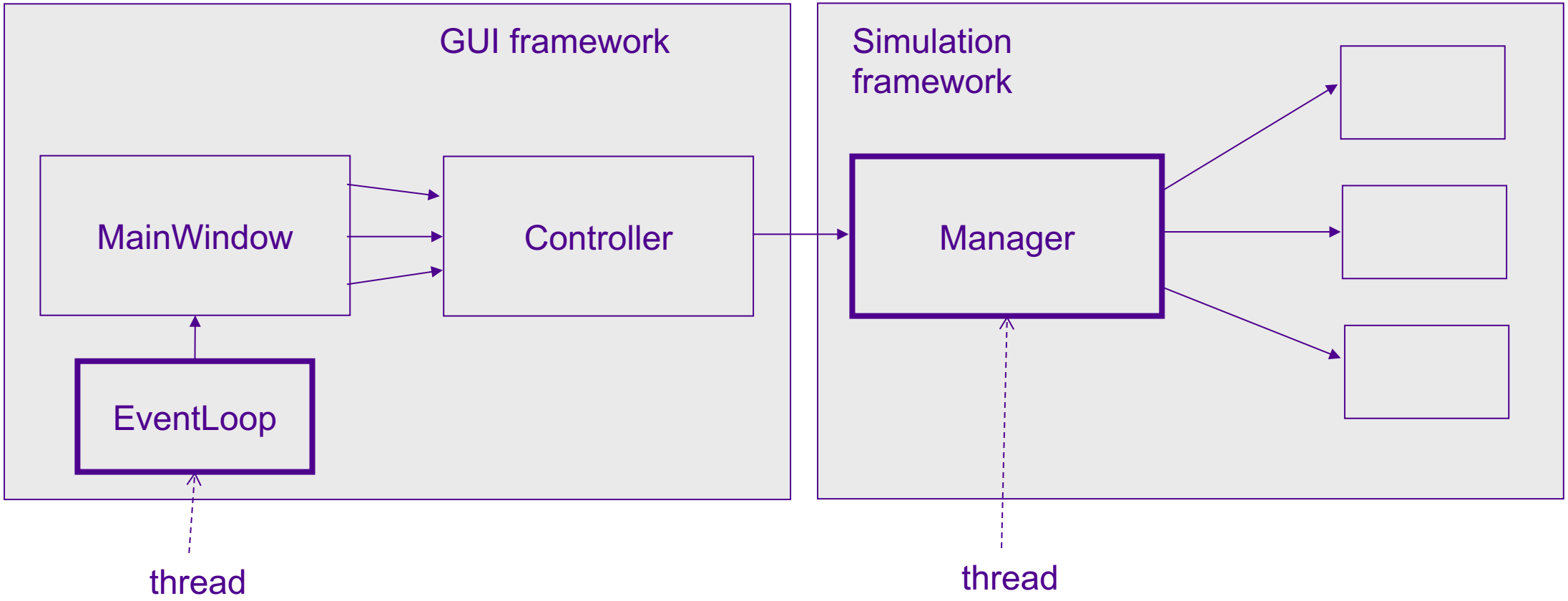
Abstract Factory



Observer


...
if meteorHit {
for all obs:
obs.notifyMeteor();
}
...
Framework
MeteorSource
registe(…)
DefWorld
simulate()
register(MeteorObserver)
…
MeteorObserver
notifyMeteor()
Creature
Application
MyCreature
notifyMeteor()

Combining two frameworks with a thread



Developing frameworks 1

- Readability. The code of the framework is readable, use of framework is producing readable code.
  - Reading code vs. Writing code
- Ease of use, simplicity
  - Do not force the users to repeat the same thing (first UI description, then the same in code, etc.)
  - Think code completion, too (no: getElementName, getElementType, getElemen...)
  - Searching for error in runtime if something does not work correctly vs. compile time error (if statically typed) → favor compile-time checks.

Developing frameworks 2

- Hidden meanings and leaking abstractions
  - Document hidden meanings
  - Try to make abstraction so that they do not leak in part of the situations = consistency
- Scalability
  - Simplicity, dependency injection pattern, ease of starting

Framework properties

- Inversion of control: the framework is responsible on general execution of the program (not the programmer or the code using the framework)
- Scalability, customisation: creating new parts, specialising basic services of the framework, using parameters...
- The framework has a default functionality: the framework is doing something, its not just a collection of empty interfaces.
- Frameworks own code is not modified (like in the case of libraries).

Pros of frameworks

- Benefits of frameworks as implementation technology of product platforms:
  - A lot of experience (e.g. GUI frameworks)
  - Applies common, well-known OO technology
  - Supports open variation points
  - Supports layered or hierarchical product platforms well
  - “Hard experts make the framework”

Cons of frameworks

- Technically demanding way to make the software, the process is often very iterative.
- The frameworks become easily large and complex software that is difficult to manage.
- Usage of time, costs, if only a single application is made.
- Testing of these applications can be difficult without framework's code.
- Making an application on top of the frame: learning, flexibility ?, dependencies.

Conclusions 1

- Traditional framework is the way of OO to implement product platform.
- Framework architectures are used widely in companies, experiences are mostly positive.
- Making a framework is much more demanding then writing a single application.
- Avoid making white-box frameworks.
- Framework may be slim when compared to the application itself: functional frame and contents.

Conclusions 2

- Basic use of design patterns and frameworks should be successful without studying design patterns
  - Avoid unnecessary details, make usage as simple as possible
  - Describe things on users' point of view, not how they are implemented inside the framework
- Javascript frameworks and similar
  - Offer help to implement the application
  - E.g., Ready MVC platform; no need to implement it from scratch.
  - The user's code takes care of actual application part, framework takes care of general matters.

Product-line architectures

Product-line approach in general

- A same kind of product with different properties for different target groups
- Cars:
  - Different levels of equipments
  - Different motors
  - Sedan, hatchpack, cabriolet,…
- Display drivers:
  - Performance, power dissipation, noise
  - Same production line, cut versions (testing premium properties, if failing, sell the product as lower model; functionalities cancelled by bios or mechanical solutions)

Reuse

- **Opportunistic**: Code that by coincidence fits for the new application is used
- **Designed**: organisation uses resources to develop generally reusable software that provides abstractions and variation points suitable for the industry.
- **Opportunistic way does not work well in practise**
  - Reuse hard, even designed way does not guarantee success
- **Bottom-up**: Potentially reusable components are added to the commonly used library from which ready-made components are searched for a new application.
- **Top-down**: Reusable software is tailored to a wider areas (e.g. Interfaces, architectures, frames).
- **The bottom-up approach leads to low-level reuse.**

Definitions

- **Product family**: a set of software products having a same kind of structure and functionality
- **Product line**: All the artefacts, tools and processes that support development and maintenance of product family members.
- **Product line architecture**: When the products share the product line, they share also its architecture

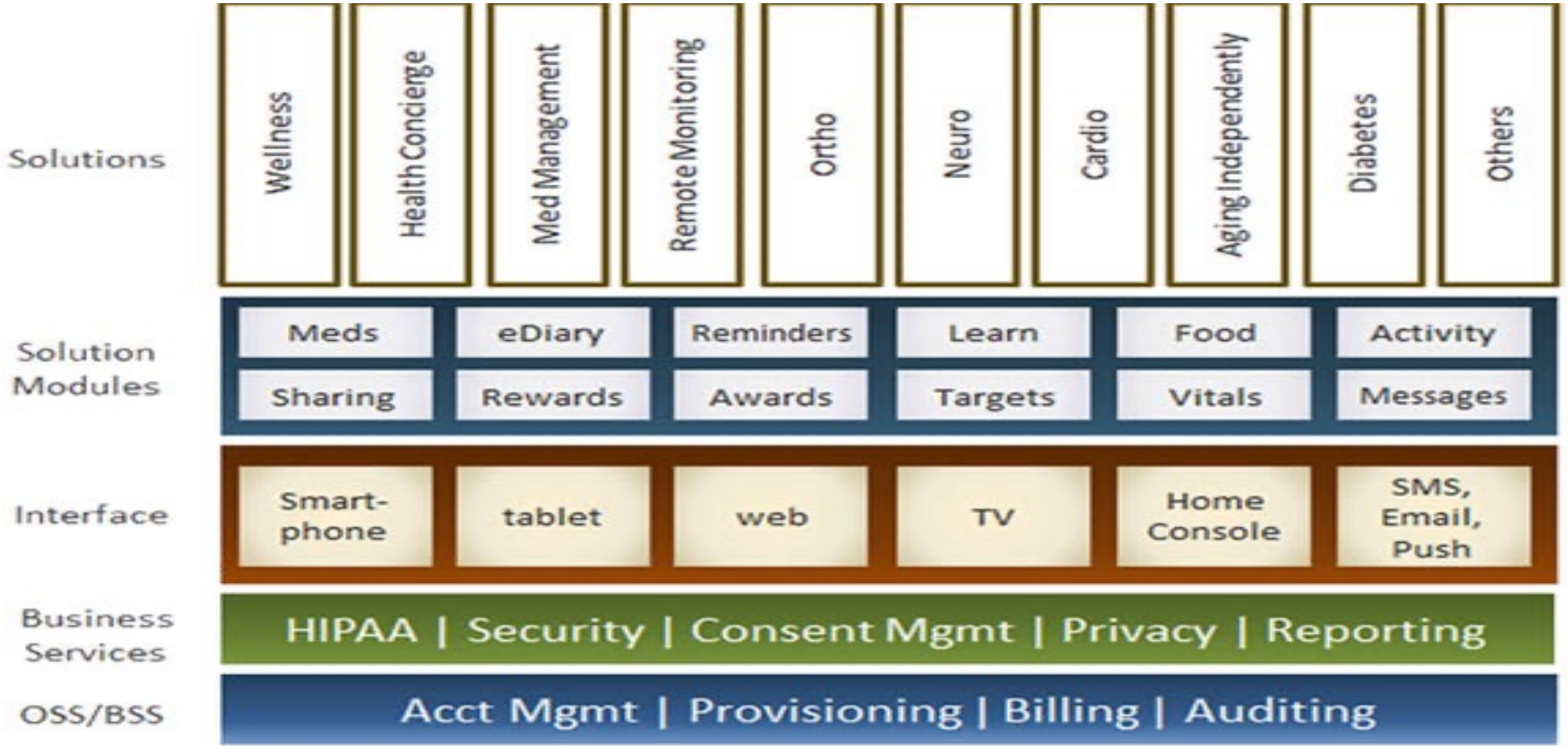
Product line = reuse of software that are based on a common architecture and platform

- Product-line architecture
  - http://www.sei.cmu.edu/productlines/
  - Software Product Line Engineering: Foundations, Principles and Techniques: Klaus Pohl, Günter Böckle, Frank J. van der Linden (2005)

Examples of product lines

- Cellular phones
- Insurance systems
- Banking systems
- Computer games
  - Angry Birds, Angry Birds Rio, Angry Birds Magic, Angry Birds Space, Angry Birds Star Wars, Angry Birds Star Wars II, Angry Birds Seasons, Angry Birds Stella….
- Machine control systems

Product family example (health services)



Different versions and service levels

- Freehand / free version / test version
- Basic version
- Premium model

Some ways to implement

- Registrations:
  - The simplest model: a code to activate full version is given when purchasing.
  - A little bit advanced: registration by the applications, application is connected to the device it resides.
  - Check is done during start-up (license file or network check).
- Often all versions in the same package, but a run-time check if license is available.
- Also by libraries: extended library loaded when a better version is registered.

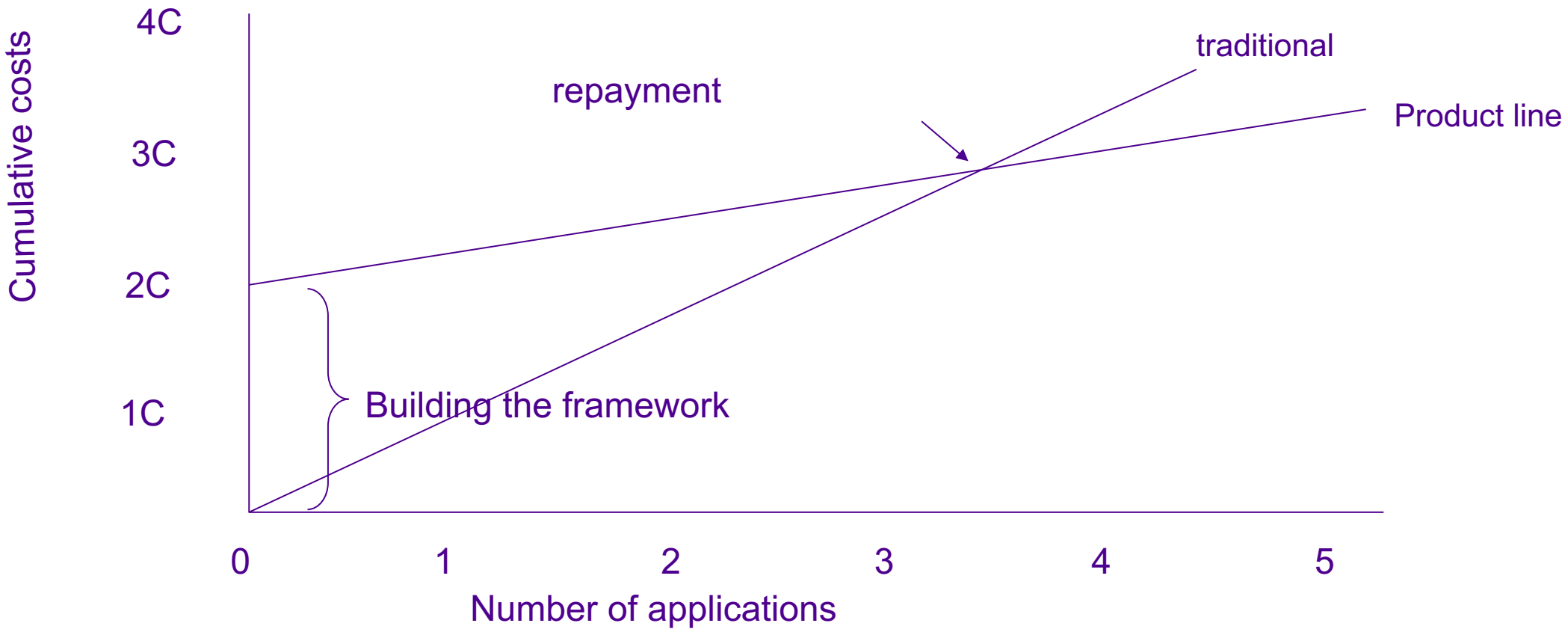
Software development based on product lines

- **Key objectives**: significant re-use, shorter development time, better quality with less resources, a consistent and streamlined development process, consistent products
- **Prerequisites**: A product family with sufficient common features and a well-understood variation is desirable: requirements must define scope, common requirements and variation points
- Product-line type situations arise sometimes without a clear product family concept:
  - Ignorant requirements often result in variation points
  - Open source is often interpreted as a product line
  - Products are wanted to be "customisable"

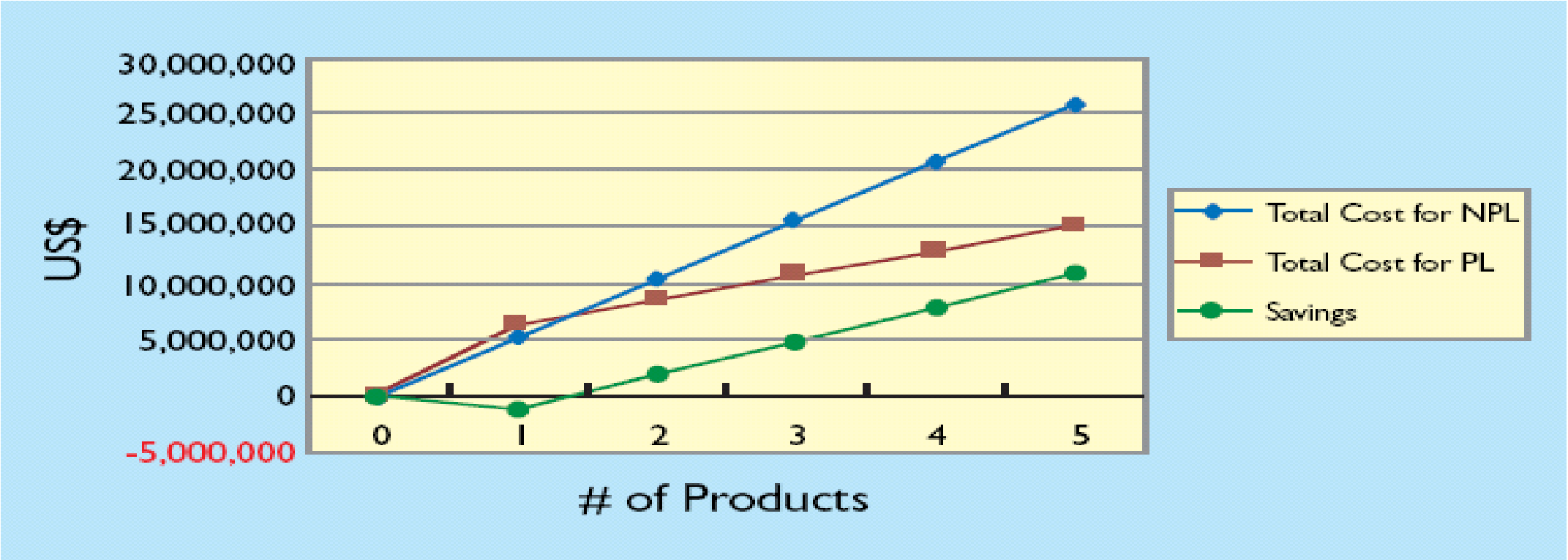
Viewpoints of product-line-based software development

- The business point of view
  - When is the product line approach economically feasible? What kind of business models?
- The organisational point of view
  - How can an organisation adopt and support product-line approach and development?
- Process point of view
  - What kind of development process is suitable of product lines?
- Technical point of view
  - What kind of architecture models and technologies are used for product lines?

Business viewpoint 1



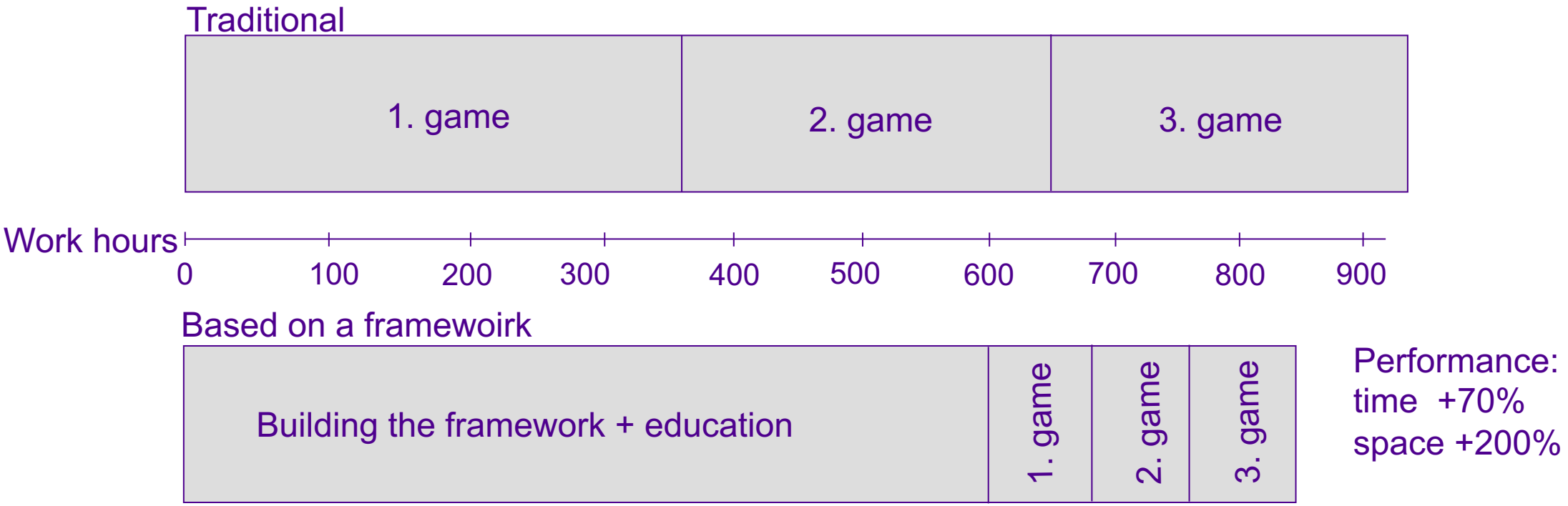
Business viewpoint 2



- Is based on work estimation methods
- In H.P. et al.: A quality-based cost estimation model for the product-line life-cycle. CACM 49 (2006), 85-88

Example (“real”)

- Application area: videogames
  - Easy to make new games with small changes
  - The performance and space requirements of the application got worse



Santelices R.A., Nussbaum M.: A framework for the development of videogames. Software Practice & Experience 31 (2001), 1091-1107.

Product lines and costs

- Making product line without a product is challenging
  - Abstracting or generalising wrong concepts
  - Easy to stuck
  - Without product hard to estimate essential things
- A common way is to make the first product fast
  - Utilising the experiences, making product line easier and faster
  - The product is got quickly to the market
  - Danger: The first product is forced be more general.

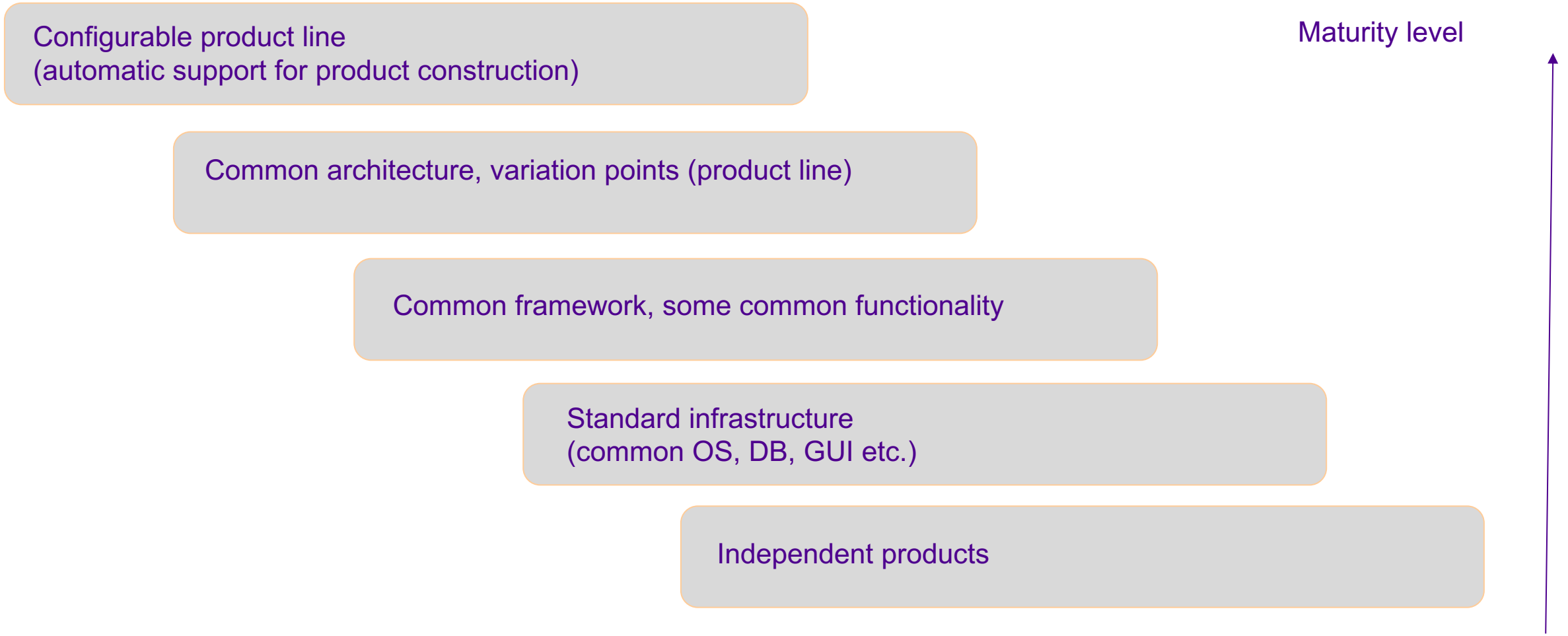
Pros of product lines

- Extremely reused code and know-how
- Special expertise of implementers decreased
- Accelerated product cycle
- Productivity growth in the long run
- Product standardisation
- Standardisation of development process and tools
- Quality improvement
- Support fast prototypes

Cons of product lines

- Staff turnover: motivation, expertise
- Stiffens development
- Conflict frameworks vs. products (coverage, schedule, resources, etc.)
- Conflicts between desired properties of products
- The first product takes a long time
- How to test a product line?
- Product-line focus may disappear
- Quarterly economics

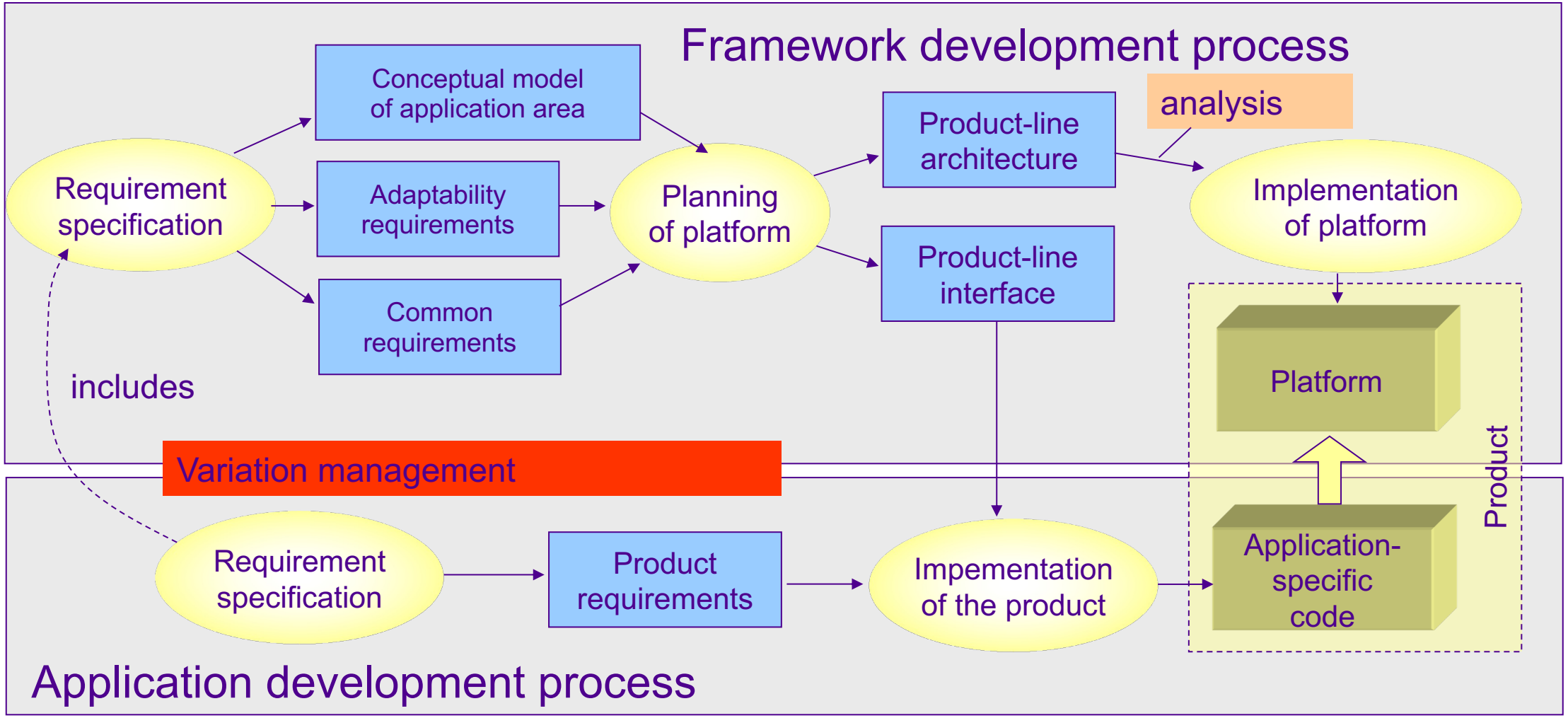
Maturity levels of reuse



Different types of product-line software development

- Has the company previously had software in the target area?
- Will the product line be made progressively or at a time?
  - Convert existing components to more general ones.
  - Replace existing components with a product platform.
  - Develop a new platform gradually for a growing product family (no existing software).
  - Develop a new platform at a time for the entire product family planned (no existing software).

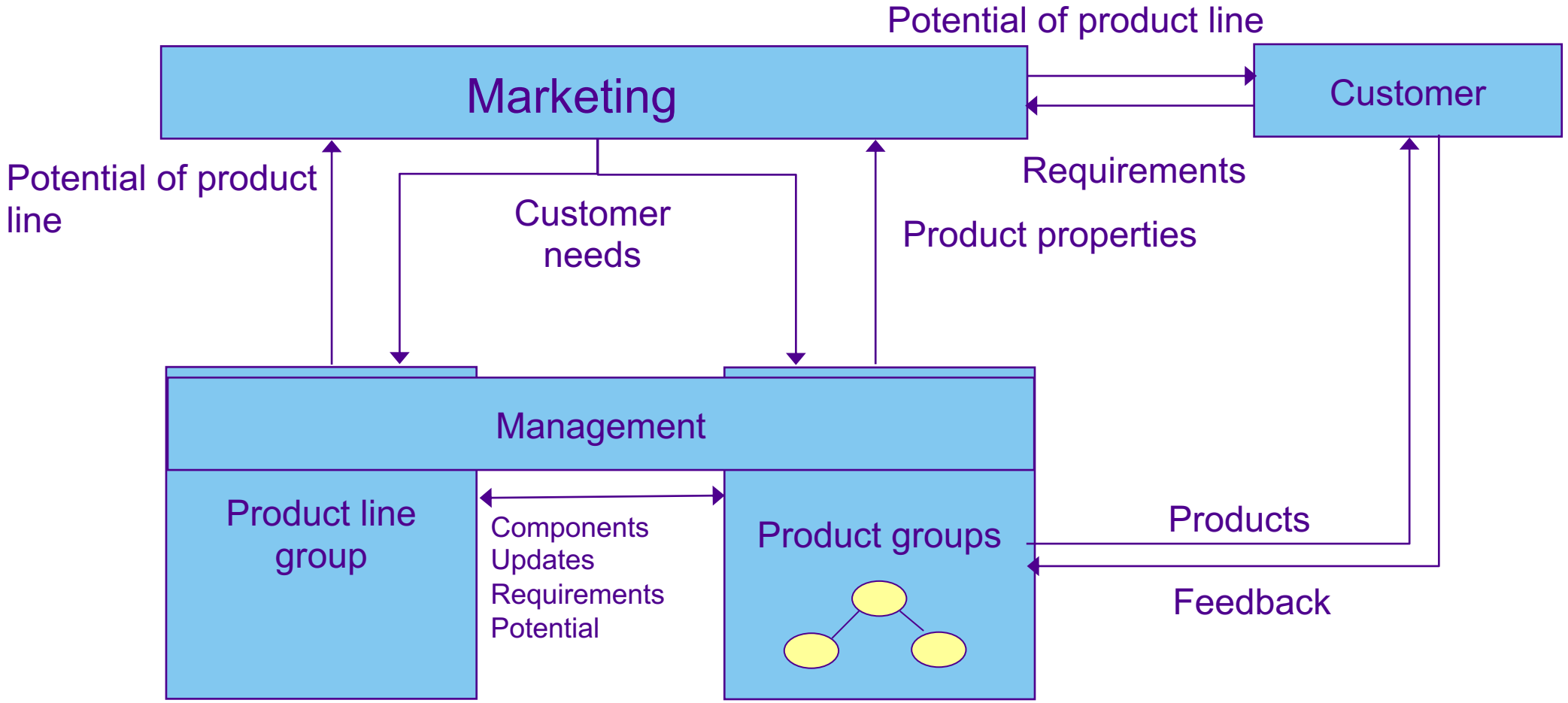
Product line process



Requirements and design decisions

- Ultimately, the contract defines the requirements, all subsequent decisions are planning
- In the case of a product line, there is usually no contract, but the product line is made for internal use
- (Especially) in the case of a product line, there is no clear distinction between requirements and design: the requirements are planned, too

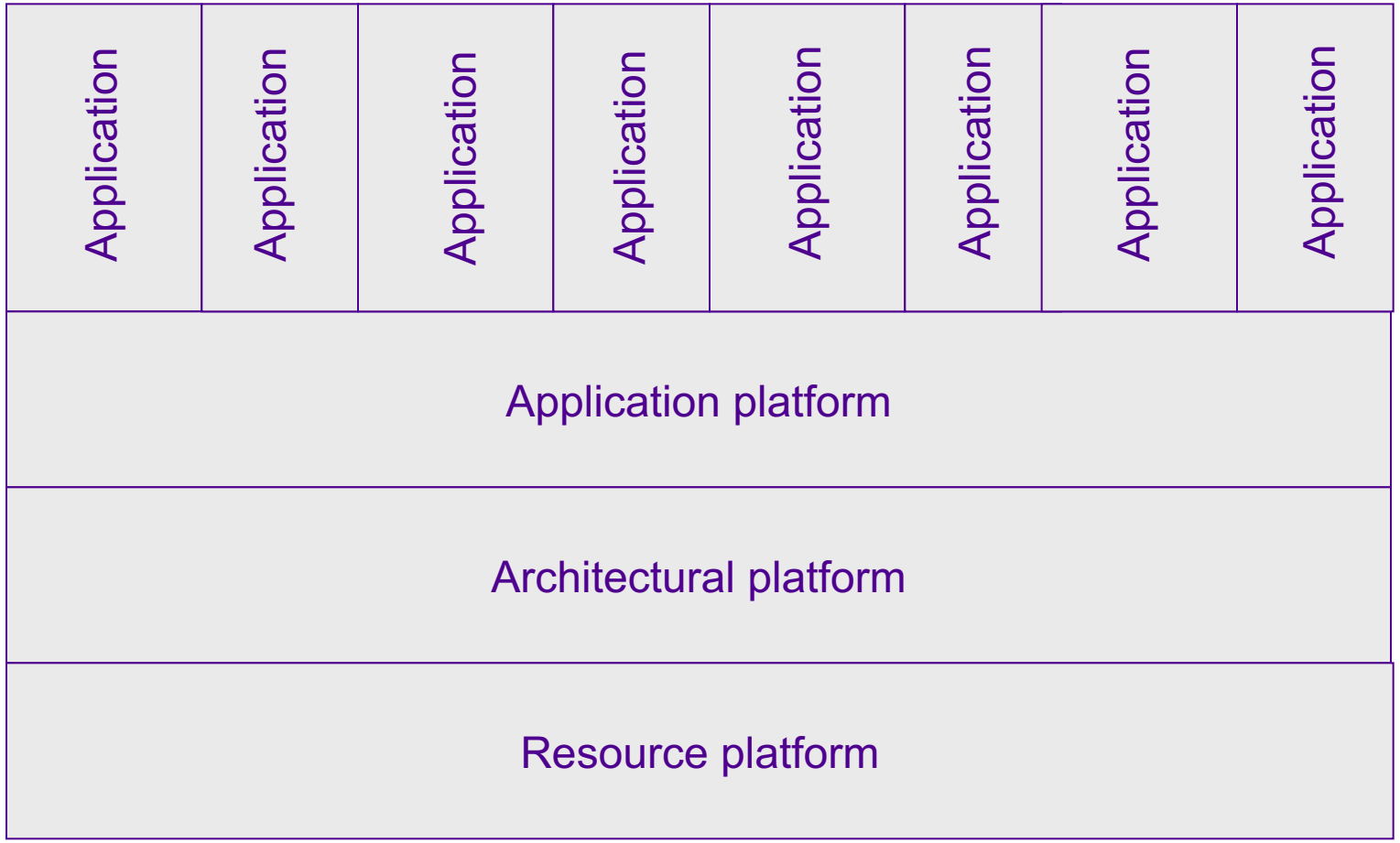
Product-line organisation



Technologies supporting product lines

- Component technologies
- Object technologies, frameworks
- Model-based methods
  - Domain specific languages (DSL), application area oriented models
- Parameterisation, parts to be interpreted
- Environments for textual and visual languages

Tier style for product-line architectures



Designing a product line based on tier style

1. Decide on the general support services and plan their abstraction
2. Decide on a basic architectural style and design the infrastructure it needs (e.g. messaging, client-server)
3. Design the common components of the product family and implement the variation points
4. Note: Some layers may be very thin or even missing

Tier architecture helps to maintain the product line

- What parts are affected by database changes?
- How to ensure that the basic product-line architecture is not changed?
- How to ensure that single product issues are not brought into the basic architecture?
- How to ensure that product-specific issues are not messed up with application-specific issues?
- Which parts have the most affect on the quality features? What parts are (likely) to be changed if the quality requirements change?