Architectural Styles & Patterns

Nauman bin Ali
Mikael Svahnberg
{nal, msv}@bth.se

Pattern

"Patterns document existing, well-proven design experience."

Pattern

- A certain context, a problem and a solution

- Examples: Abstract Factory, Singleton, Decorator, Proxy, Strategy, State

Architectural pattern

- A design pattern for software architecture.
- Document existing, well-proven design experience
- Imposes a set of rules on a software architecture.
- Typical architectural styles are:
  - Pipes and filters
  - Layers
  - Blackboard
  - Model-View-Controller
  - Centralized vs. Distributed
  - Monolithic vs. Microkernel
  - And many many more…

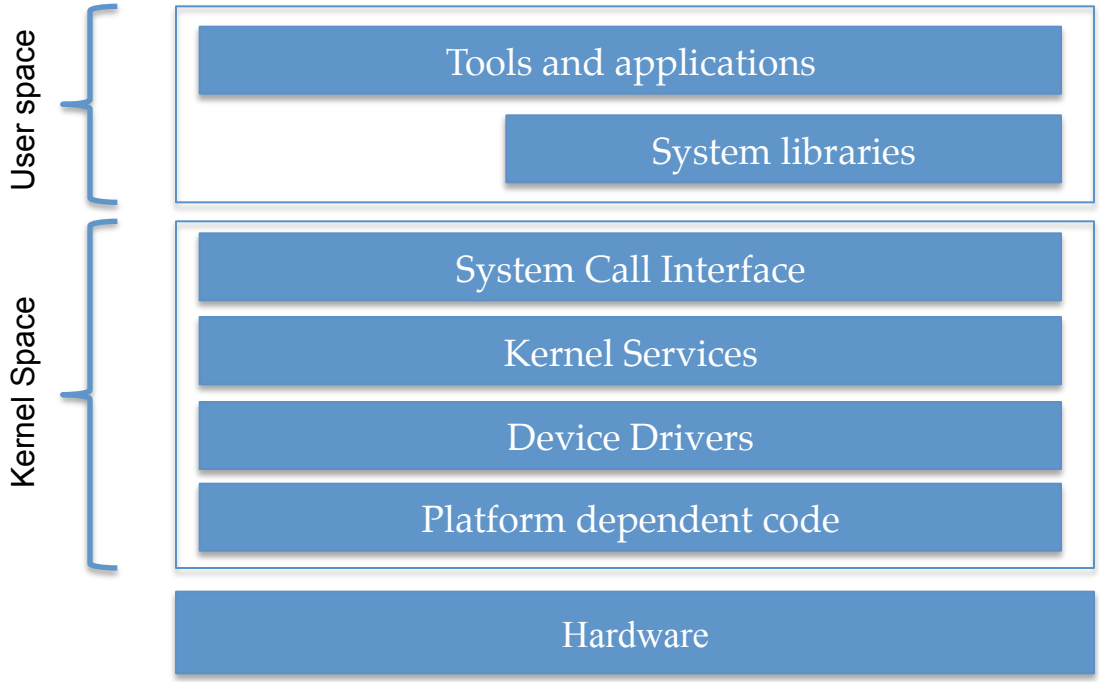
Benefits

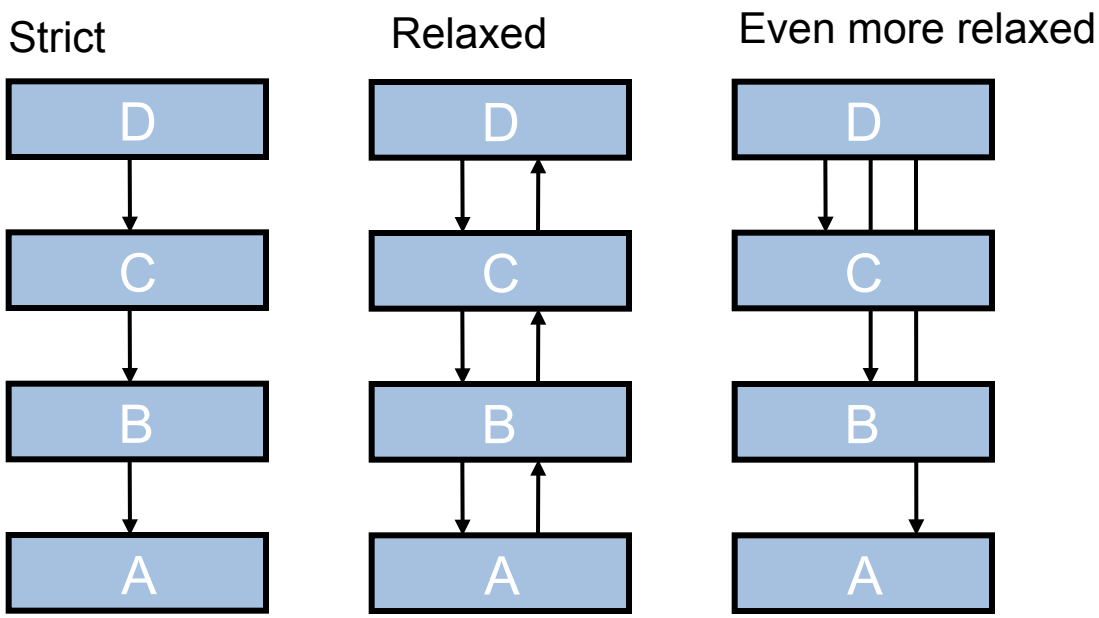
- Help you manage software complexity
- Serve as mental building blocks
- Identify and specify abstractions
- Provide a common vocabulary and understanding for design principles
- Are a means of documenting software architectures
- Support the construction of software with defined properties
- Help you build complex and heterogeneous software architecture

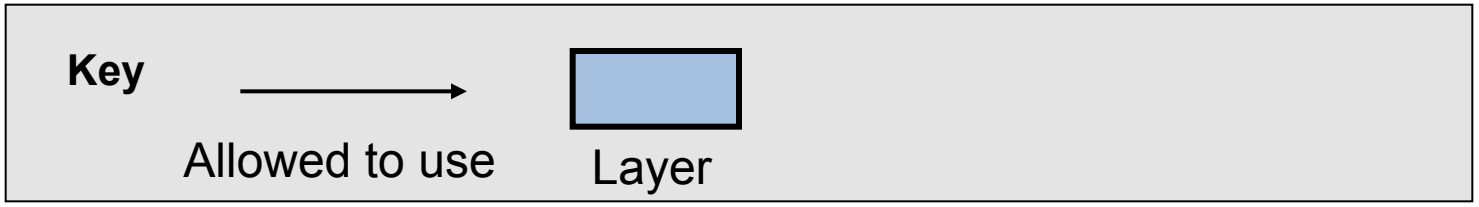
Layered Style

Problem:

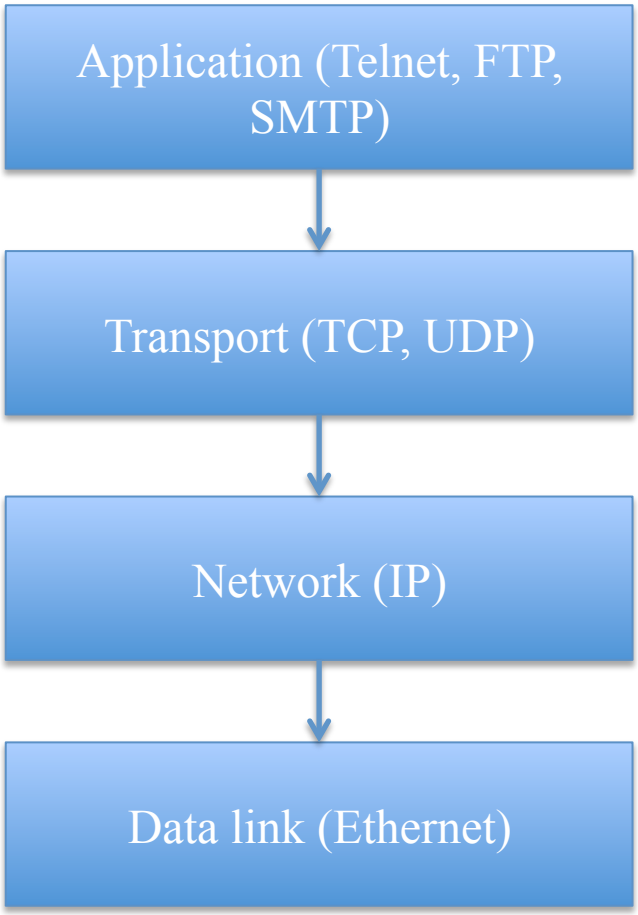
- Application consists of several subtasks that operate on different abstraction levels.
- Example:
  - fopen()
  - Sys trap




Strict
Relaxed
Even more relaxed
D
C
B
A
D
C
B
A
D
C
B
A


Key
Allowed to use
Layer

TCP / IP protocol stack



Layered cont.

Not so good performance

- **Much overhead between layers.**

Good maintainability

- **Low coupling between layers.**
- **If the functionality is isolated to one layer.**

Low reliability

- **If a layer crashes, more might follow.**

Good security

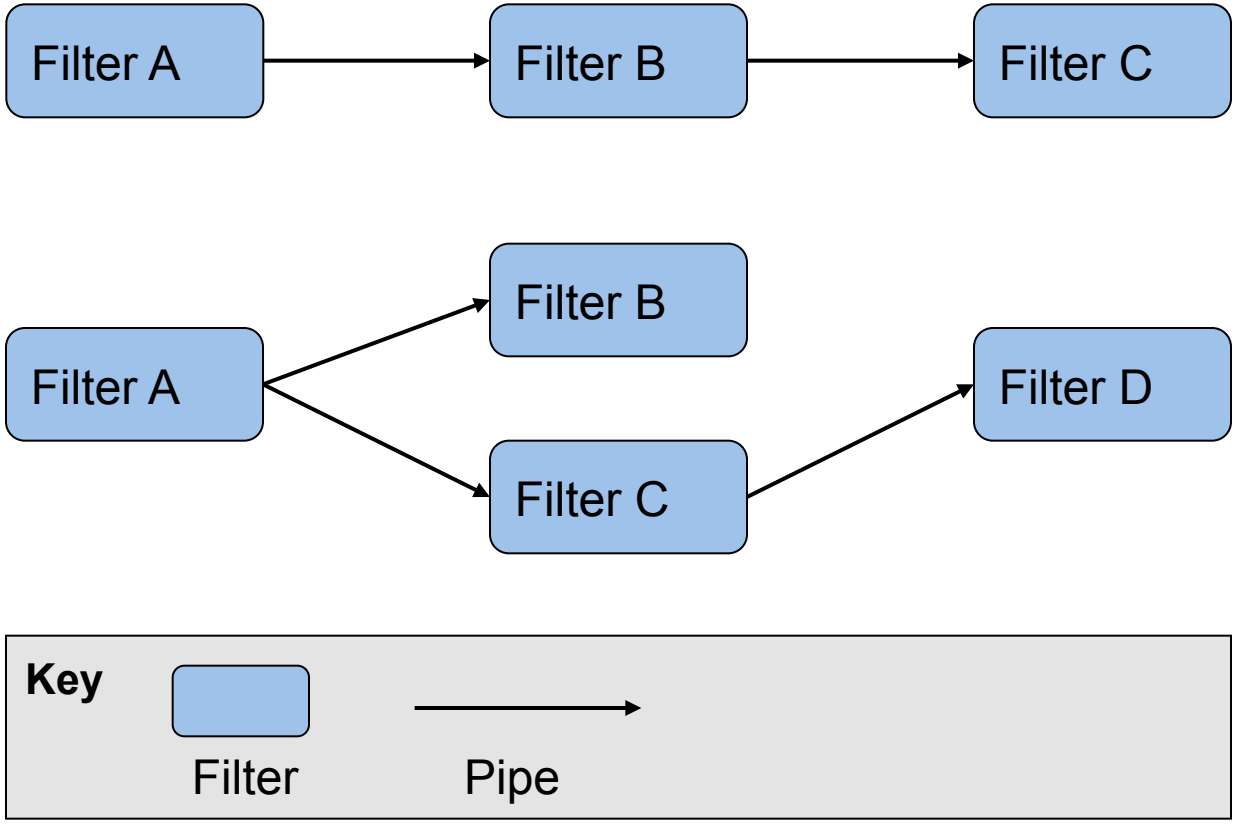
- **Easy to add a security layer.**

Pipes and Filters

Problem:

- **Need to process a stream of data**
- **Several processing steps**

Pipes and Filters



Pipes and Filters cont.

Good performance

- **As it allows for concurrency**
- **If work units are kept at an efficient size…**

Good maintainability

- **…but only if changes are local to one filter.**
- **Filters can be added / replaced.**
- **Pipes and filters can be reconnected at runtime.**

Low reliability.

- **If one filter fails, all filters fail.**

Good security.

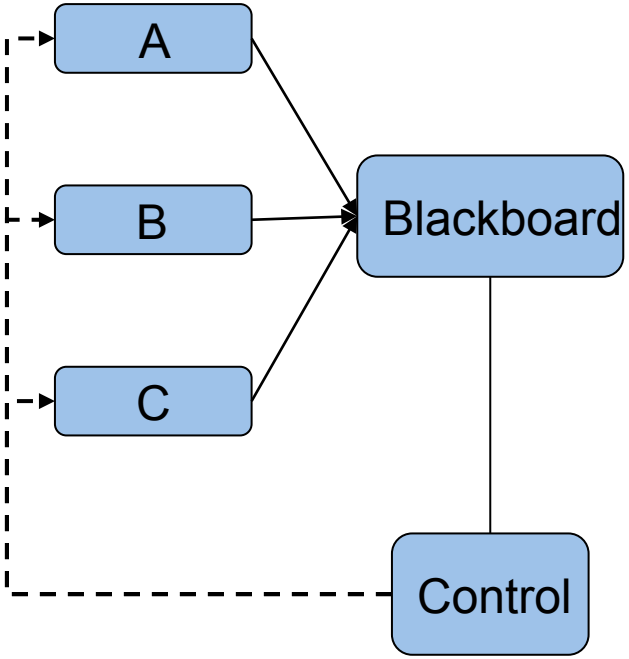
- **Security filters are easy to add.**

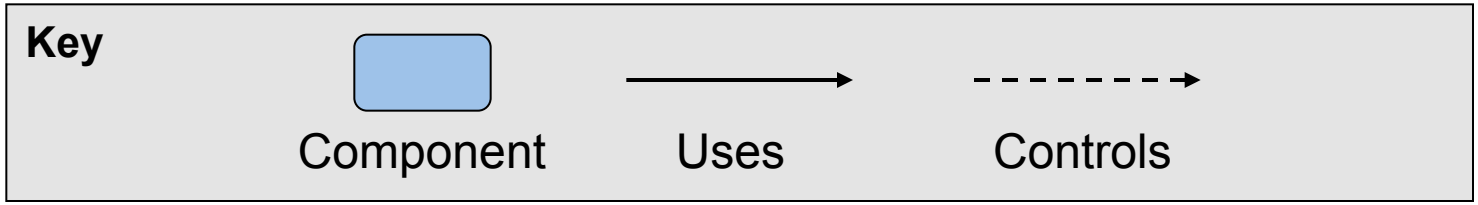
Blackboard

Problem:

- **Several subtasks**
- **No determined order in which to execute the subtask**
- **Is depending on the currently available data**

Blackboard





Blackboard cont.

Varying performance

- Depends heavily on implementation.
- Controller lowers performance even more.

Good maintainability

- Easy to add or remove components.
- A controller makes it a bit more difficult.

Low reliability

- Non-deterministic execution. Difficult to debug.

Low security

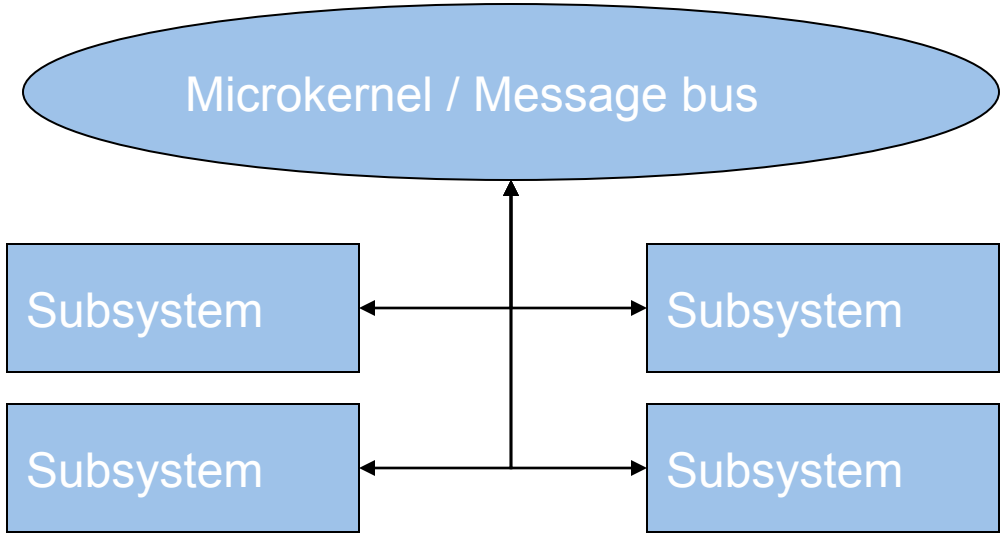
- All components can access all data.

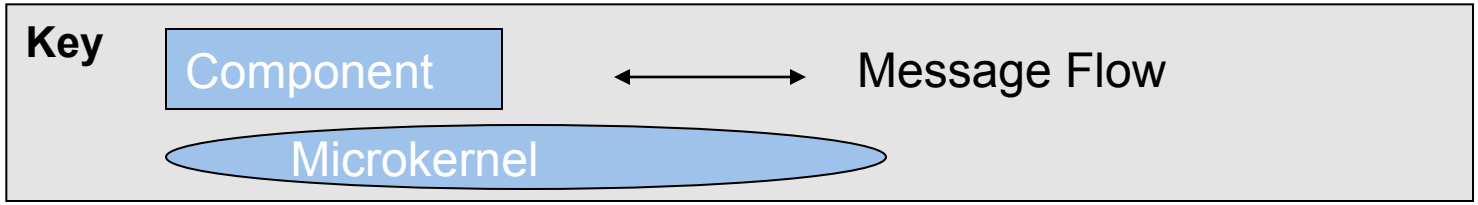
Microkernel

Problem:

- Several subtasks (possibly collaborating)
- May need to add or remove subtasks
- Need “pluggable” architecture

Microkernel





Microkernel cont.

Good portability

- May separate hardware-dependent functionality to separate subsystems
- May often just need to port the microkernel – not the other subsystems

Flexible and Extensible

Good Maintainability

- Separate policy from mechanism

Reliable

- Not depending on that all subsystems are up-and-running.

Good Transparency

- Easy to distribute some subsystems

Not so good on performance (compared to monolithic system)

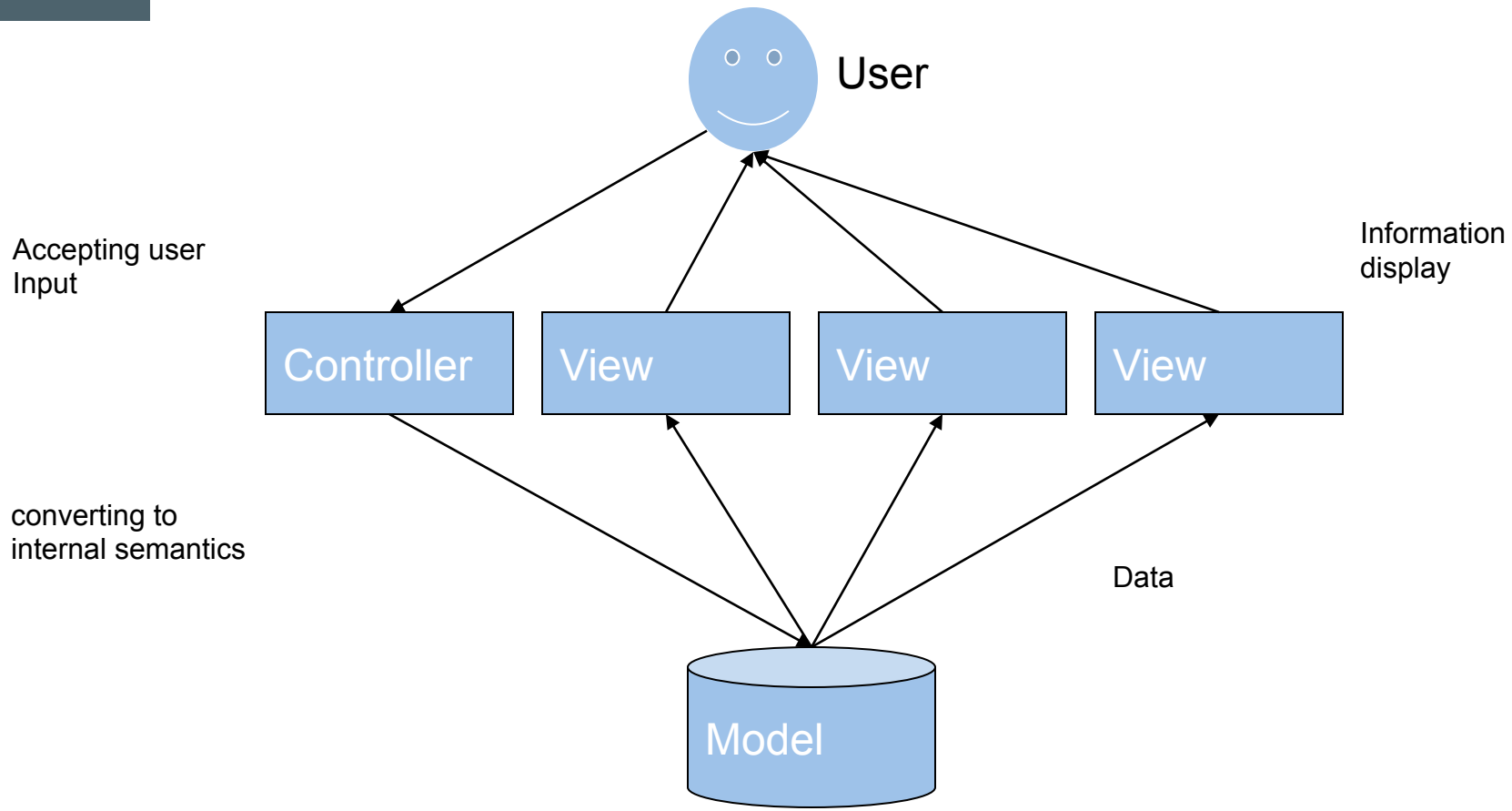
Complex to design and implement

Model-View-Controller

Problem:

- Interactive system
- Multiple views
- How to present data
- How to manipulate data

Model-View-Controller



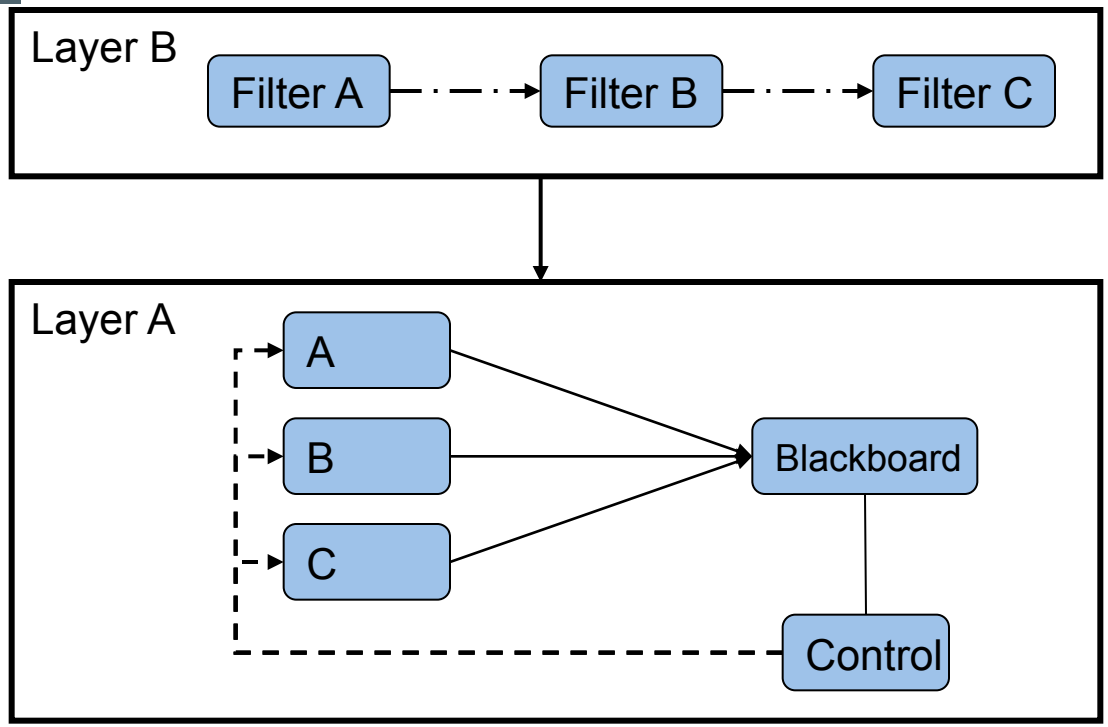
Model-View-Controller cont.

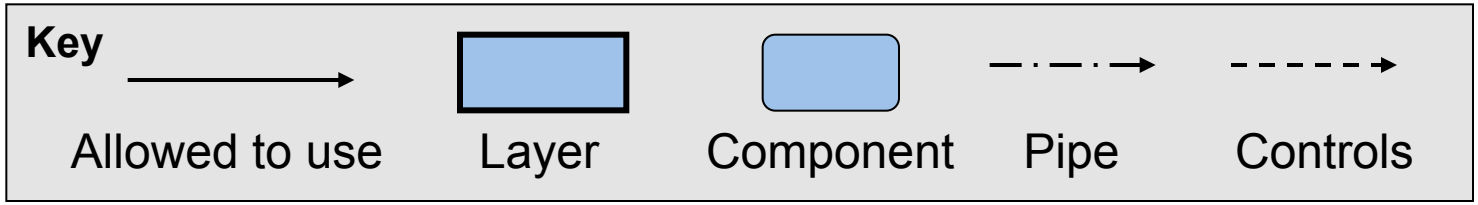
- Flexible
  - Easy to add new views
  - Easy to add new controllers
  - Easy to add new look-and-feel
- Increased complexity
- May have low performance (many updates), unchanged data
- Difficulties in re-using views separate from controllers
- Dependency on model interface
- Platform-dependent code in view and controller
- Document-View is a variant of MVC.

Choosing an architectural style

- Nature of the system e.g. interactive system with slight variants of presentation
- System's quality requirements
- Consider alternate patterns
- Often a single pattern will not be enough for the whole system.

Combining architectural styles





Combining architectural styles

Styles can be combined *if*:

- Conflicts between constraints can be resolved.
- If styles are isolated in separate components.

Usually one dominating style with other styles in large components.

What is an architectural pattern?

Terminology: Buschmann book refers to architecture styles as architecture patterns.

Imposes one rule on a system rather than a set of rules.

For example how the system deals with:

- **Concurrency**
- **Synchronization**
- **Distribution**
- **Etc..**

Concurrency

OS processes

OS threads

Non-Preemptive threads

Application-level scheduler

Synchronization

Semaphores

Events

Distribution

Broker

- **Bosch talks about Broker as a pattern, others talk about it as a style.**

RMI

Further readings

F. Buschmann, C. Jäkel, R. Meunier, H. Rohnert, and M. Stahl. *Pattern-Oriented Software Architecture - A System of Patterns*. John Wiley & Sons, Chichester UK, 1996.

Questions?