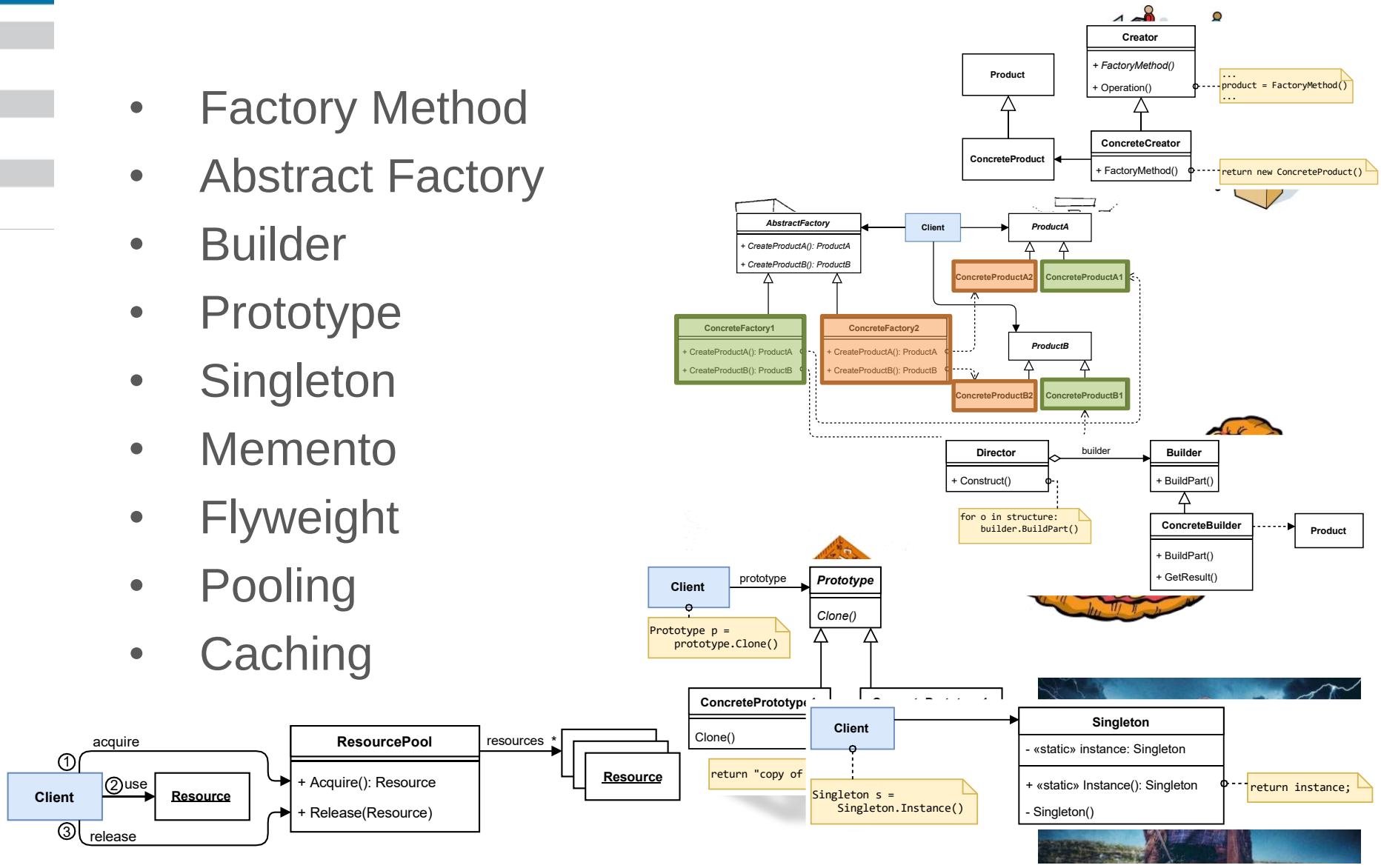
Design Patterns 448.058 (VO)

Michael Krisper
Georg Macher

06.11.2019

Revision from last time…

- Factory Method
- Abstract Factory
- Builder
- Prototype
- Singleton
- Memento
- Flyweight
- Pooling
- Caching



Use-Case: VECTO

http://reducingco2together.eu/

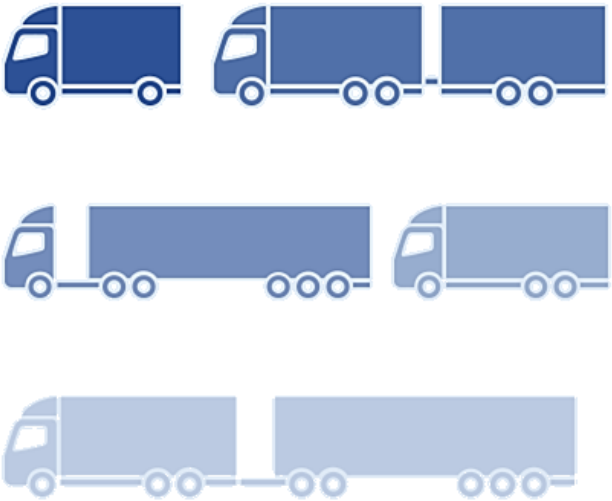
https://ec.europa.eu/clima/policies/transport/vehicles/vecto_de

Figure from ACEA, 2016, reducingco2together.eu,
https://www.reducingco2together.eu/assets/trucks/trucks-two-inforgraphic.png

VECTO

Vehicle Energy Consumption Calculation Tool

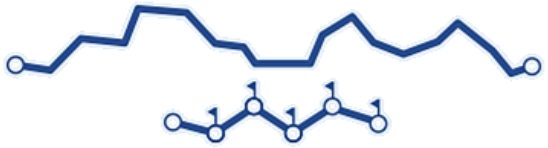
- The shape of the vehicles, which depends on their daily 'mission'.
- The same tractor or engine may end up pulling very different trailers and combinations, affecting the CO2 emissions of the complete vehicle.



- The usage pattern of the vehicles and their cargo, in other words, 'the work they do'.

Is the payload heavy or light, large or small?

Is the road flat or hilly?



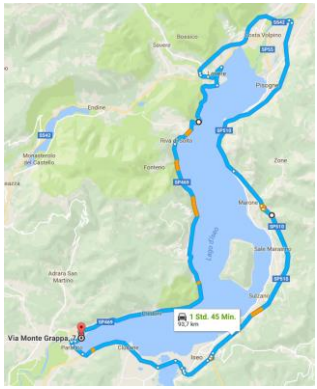
Will the vehicle travel over a long distance in one go, or is the journey short with many starts and stops?

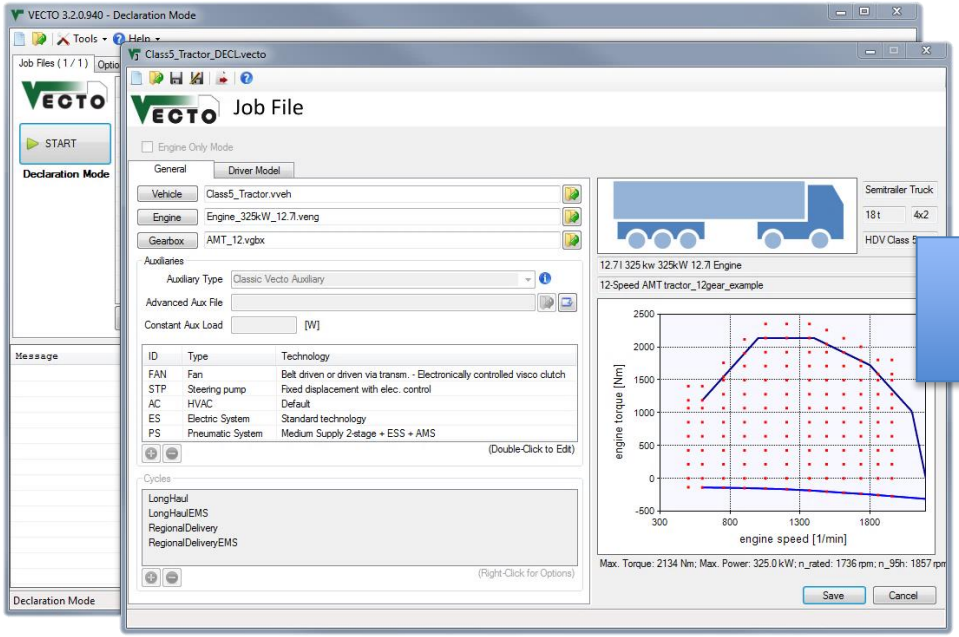
All these variables result in different CO2 emissions.

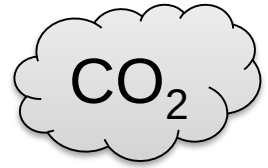
Figure from ACEA, 2016, reducingco2together.eu, https://www.reducingco2together.eu/assets/trucks/trucks-two-inforgraphic.png

VECTO

Vehicle Energy Consumption Calculation Tool

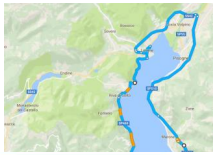


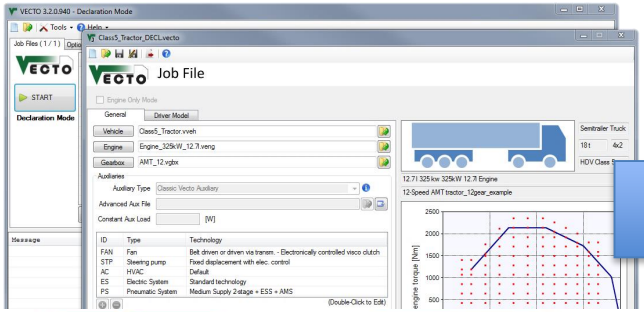




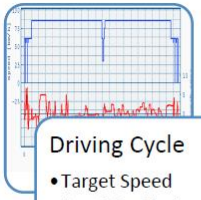
VECTO

Vehicle Energy Consumption Calculation Tool

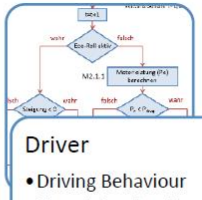




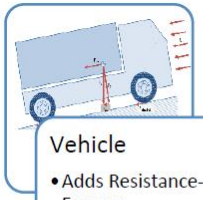
CO_2



Driving Cycle
- Target Speed
- Road Gradient



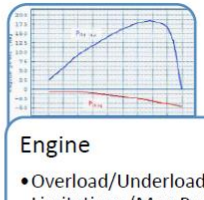
Driver
- Driving Behaviour
- Target Acceleration
- Searches for Optimal Operating Point



Vehicle
- Adds Resistance-Forces: Air, Inertia, Slope, Wheel

Brakes
- Machnically burn up Power if required

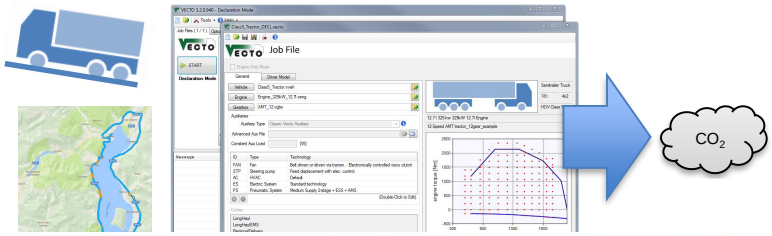
Gearbox
- Shift Strategy
- Transmission Losses

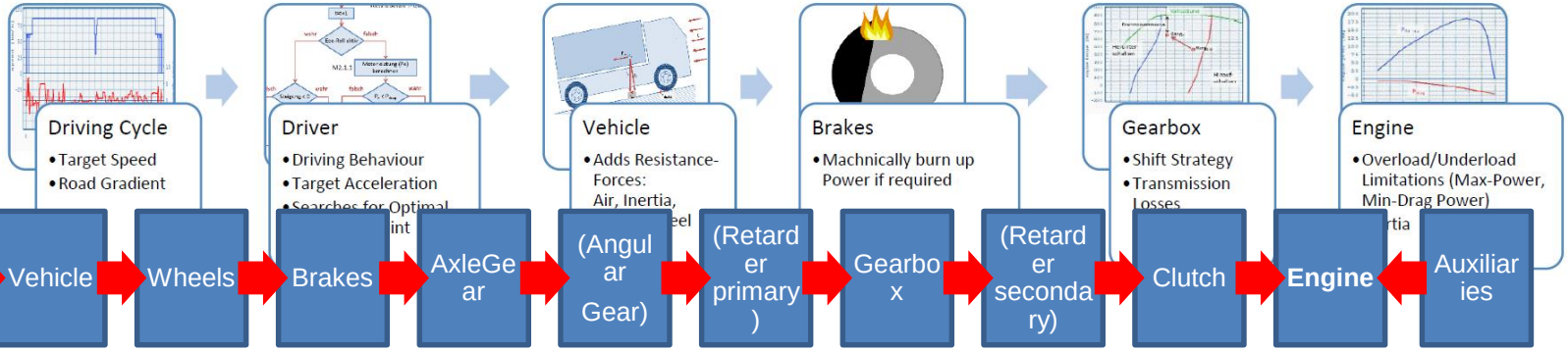


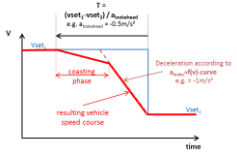
Engine
- Overload/Underload Limitations (Max-Power, Min-Drag Power)
- Inertia

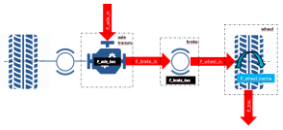
VECTO

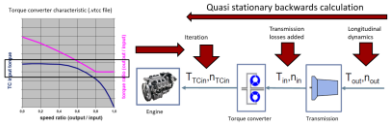
Vehicle Energy Consumption Calculation Tool

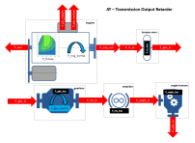


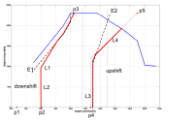


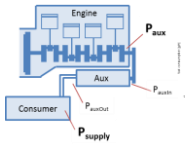


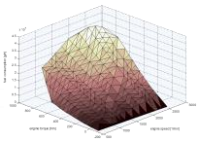






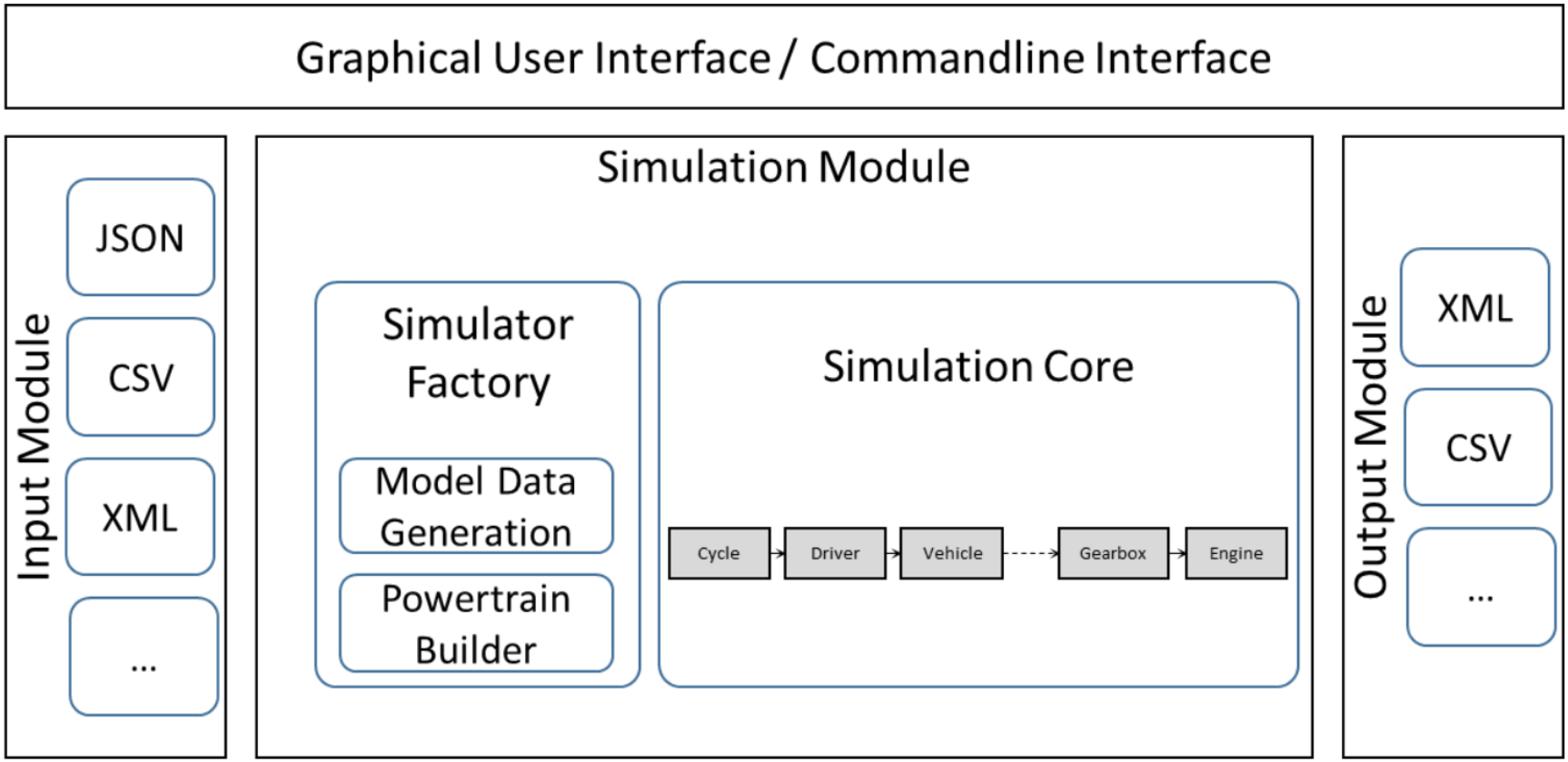






VECTO

Vehicle Energy Consumption Calculation Tool

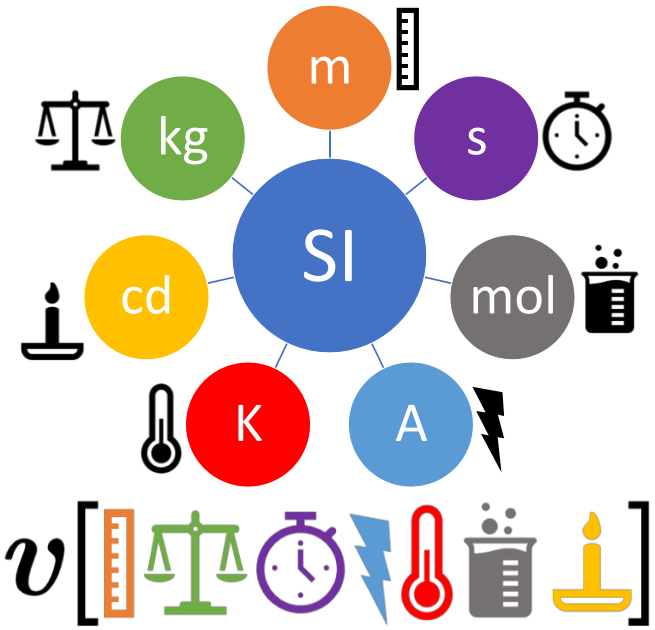


- Development Time: ~7 Years (+ previous works)
- Developed in C# and VB.net
- Lines of Code: ~50 000
- Classes: 900
- Methods: 10 000 (w/o getter/setter: ~4 500)
- ∑Cyclomatic Complexity: ~18 000
- Avg. Complexity / Method: ~2-3
- LOC Test Coverage: ~85%

VECTO Live Demonstration…

“VECTO presents itself”, Daimler AG, 10.10.2018: https://www.youtube.com/watch?v=sf30XrjEJ2M

SI Units



$$\text{SI Unit} := v\ [Q]$$

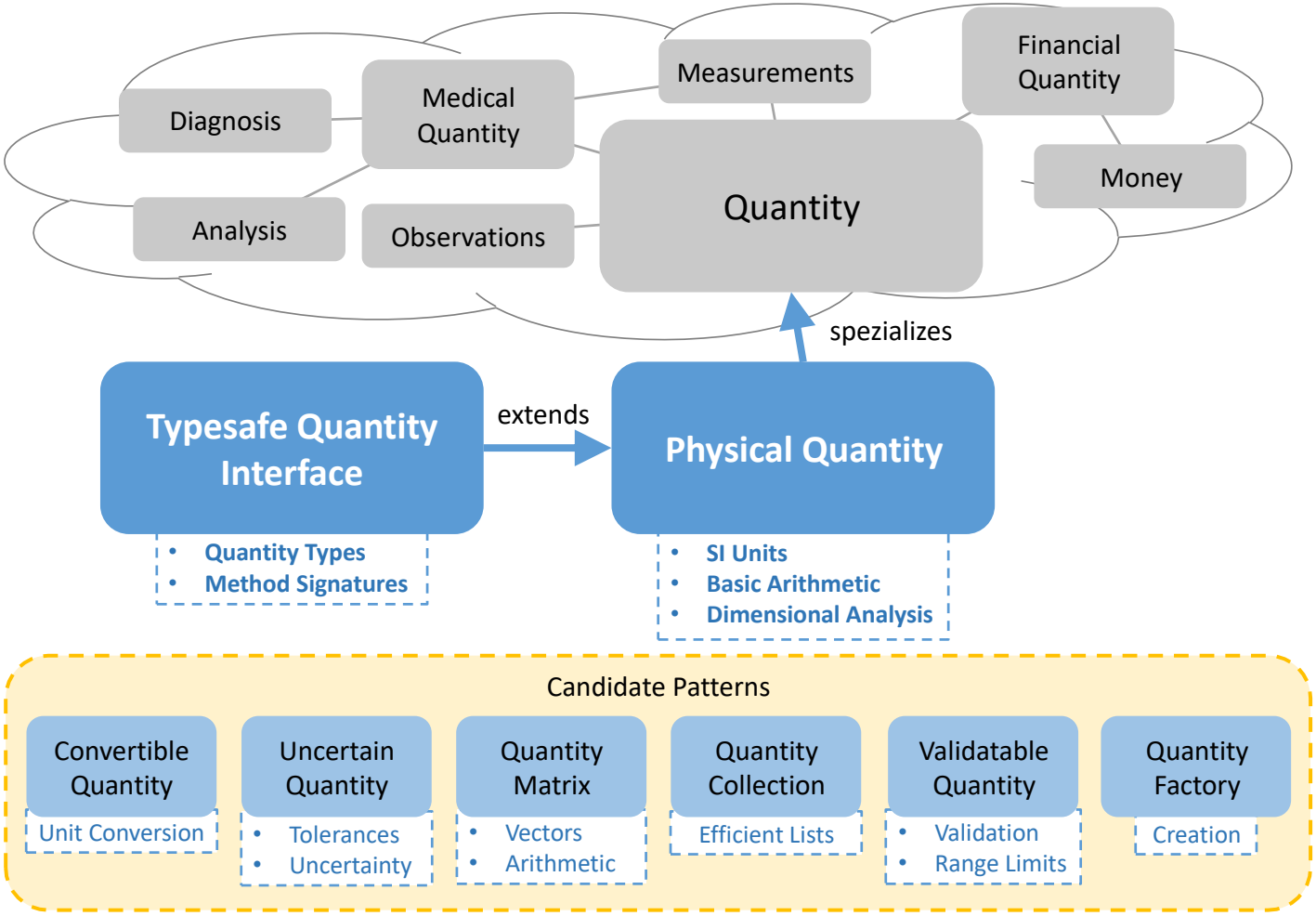
$$v := \text{a scalar value } (v) \text{ or a vector} (\vec{v})$$

$$[Q] := [\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta] \mathrel{\hat{=}} \left[[m]^{\alpha} \times [kg]^{\beta} \times [s]^{\gamma} \times [A]^{\delta} \times [K]^{\epsilon} \times [mol]^{\zeta} \times [cd]^{\eta}\right]$$

$\mathbf{v}\ [\mathbf{Q}]$	$= \mathbf{v}$	$[$	$[\mathbf{m}],$	$[\mathbf{kg}],$	$[\mathbf{s}],$	$[\mathbf{A}],$	$[\mathbf{K}],$	$[\mathbf{mol}],$	$[\mathbf{cd}]$	$]$
$5\ kg$	$= 5$	$[$	$0,$	$1,$	$0,$	$0,$	$0,$	$0,$	0	$]$
$2.8\ m/s^2$	$= 2.8$	$[$	$1,$	$0,$	$-2,$	$0,$	$0,$	$0,$	0	$]$
$7\ Nm = 7\ kgm^2/s^2$	$= 7$	$[$	$2,$	$1,$	$-2,$	$0,$	$0,$	$0,$	0	$]$

Source: [Krisper Michael, Johannes Iber, Tobias Rauter, and Christian Kreiner. 2017. “Physical Quantity: Towards a Pattern Language for Quantities and Units in Physical Calculations.” In *Proceedings of the 22nd European Conference on Pattern Languages of Programs - EuroPLoP ’17*, 1–20. Irsee, Germany: ACM Press. https://doi.org/10.1145/3147704.3147715.]

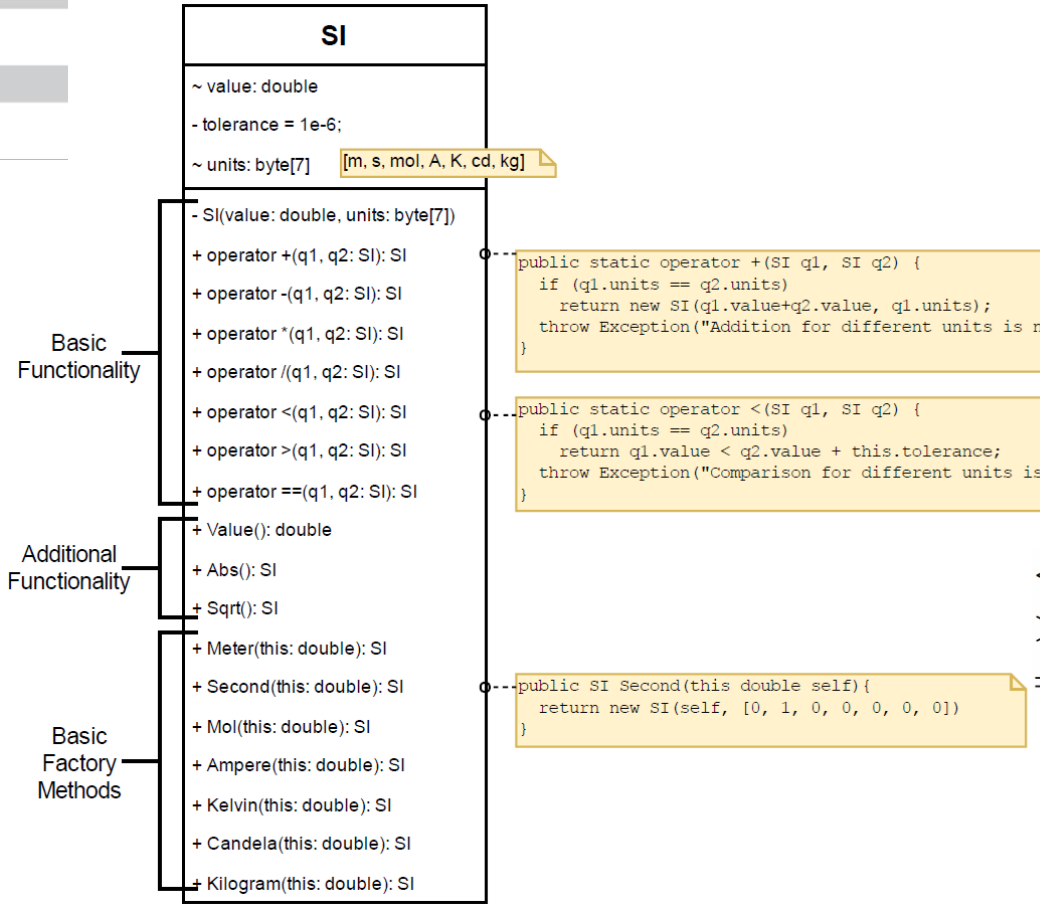
Physical Quantities Pattern Language



Source: [Krisper Michael, Johannes Iber, Tobias Rauter, and Christian Kreiner. 2017. “Physical Quantity: Towards a Pattern Language for Quantities and Units in Physical Calculations.” In *Proceedings of the 22nd European Conference on Pattern Languages of Programs - EuroPLoP ’17*, 1–20. Irsee, Germany: ACM Press. https://doi.org/10.1145/3147704.3147715.]

Physical Quantity

Ensure correctness of physical units for calculations.



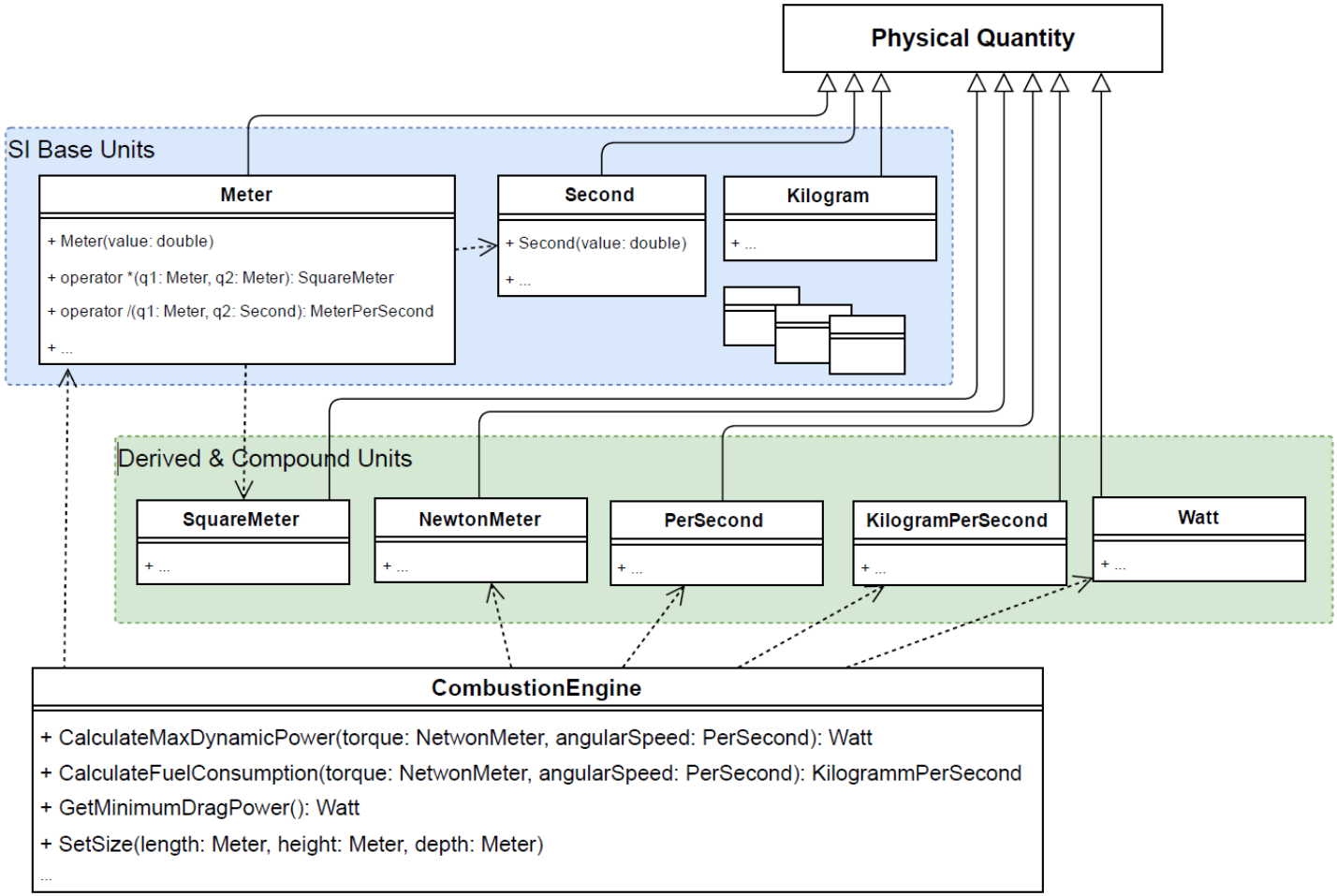
$$+ : v_1\ [Q] + v_2\ [Q] := v_1 + v_2\ [Q]$$
$$- : v_1\ [Q] - v_2\ [Q] := v_1 - v_2\ [Q]$$
$$* : v_1\ [Q_1] * v_2\ [Q_2] := v_1 * v_2\ [Q_1 + Q_2]$$
$$/ : \frac{v_1\ [Q_1]}{v_2\ [Q_2]} := \frac{v_1}{v_2}\ [Q_1 - Q_2]$$

$$< : v_1\ [Q] < v_2\ [Q] \Leftrightarrow v_1 < v_2 + \varepsilon$$
$$> : v_1\ [Q] > v_2\ [Q] \Leftrightarrow v_1 + \varepsilon > v_2$$
$$= : v_1\ [Q] = v_2\ [Q] \Leftrightarrow v_1 - \varepsilon < v_2 < v_1 + \varepsilon$$

Source: [Krisper Michael, Johannes Iber, Tobias Rauter, and Christian Kreiner. 2017. “Physical Quantity: Towards a Pattern Language for Quantities and Units in Physical Calculations.” In *Proceedings of the 22nd European Conference on Pattern Languages of Programs - EuroPLoP ’17*, 1–20. Irsee, Germany: ACM Press. https://doi.org/10.1145/3147704.3147715.]

Typesafe Quantity Interfaces

Make physical units explicit and verifiable at compile time.



Source: [Krisper Michael, Johannes Iber, Tobias Rauter, and Christian Kreiner. 2017. “Physical Quantity: Towards a Pattern Language for Quantities and Units in Physical Calculations.” In *Proceedings of the 22nd European Conference on Pattern Languages of Programs - EuroPLoP ’17*, 1–20. Irsee, Germany: ACM Press. https://doi.org/10.1145/3147704.3147715.]

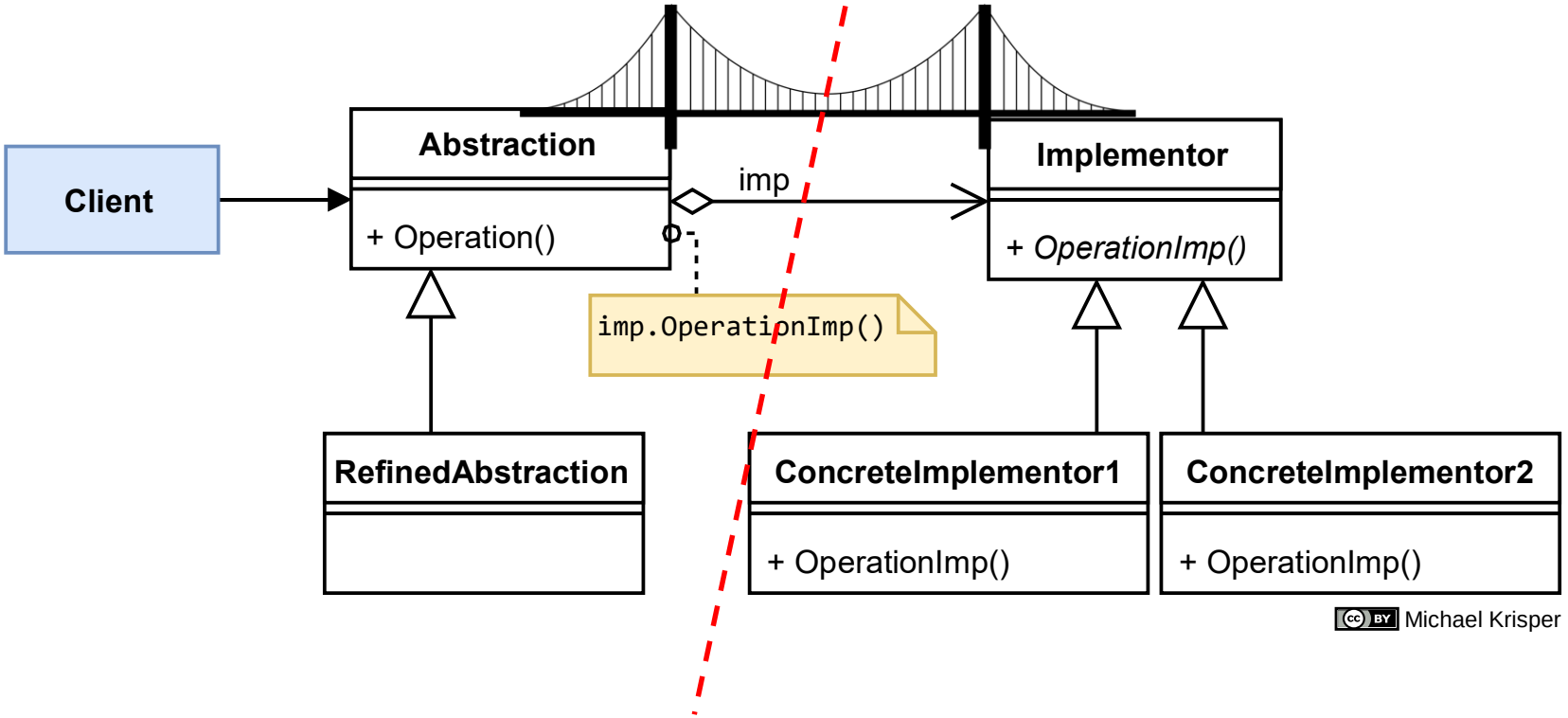
Learning Goals for Today

Communication Patterns:

- **Bridge**
- **Broker**
- Mediator
- Blackboard
- Microkernel
- Client-Dispatcher-Server / Lookup
- Messages
- Message Endpoint
- Message Translator
- Message Router
- Request Handler
- Requestor

Bridge

Decouple abstractions from implementations



Bridge

Context: Application with Abstraction and Implementation Hierarchies

Problem: How to decouple the development of abstractions from its implementations

Forces:

- Avoid permanent binding between abstraction and implementation
- Both sides should be extensible by subclassing
- Changes should be contained to one side
- You want to hide the implementation side completely
- Implementations should be compatible to multiple abstractions

Solution:

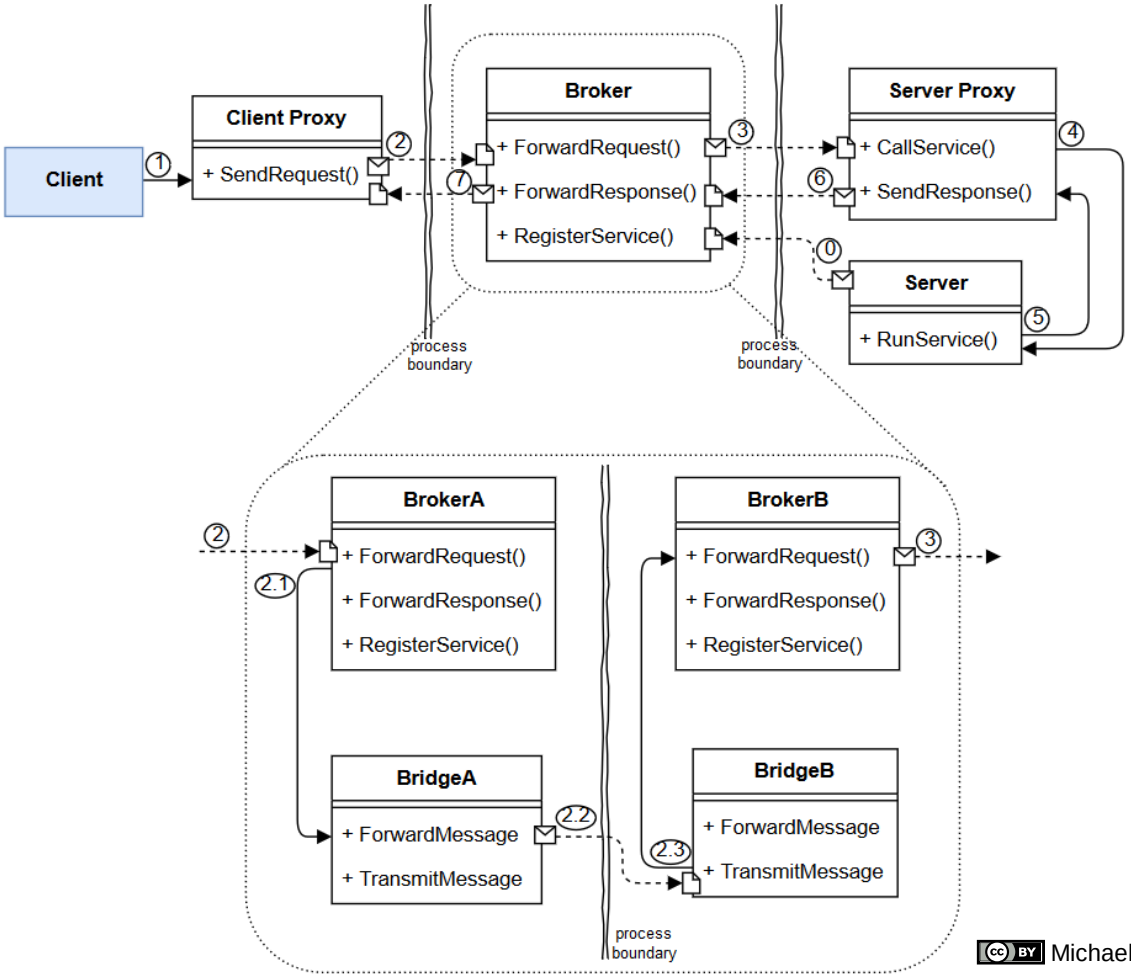
- Create two interfaces:
  1. Implementor-Interface (internal primitives)
  2. Abstraction-Interface (client-requirements)
- Implement those Interfaces with individual classes.
- Only use the implementation-interface in the abstraction

Consequences:

\+ Decoupling of abstraction and implementation
\+ Improved extensibility: Both sides can grow independently
\+ Hiding implementation details from client
\+ Implementation can be configured at runtime
\+ Elimination of compile-time dependencies
\+ Encourages layering
~ Who defines the composition? (Who builds the bridge?)
\- Higher complexity (more classes, more interfaces)

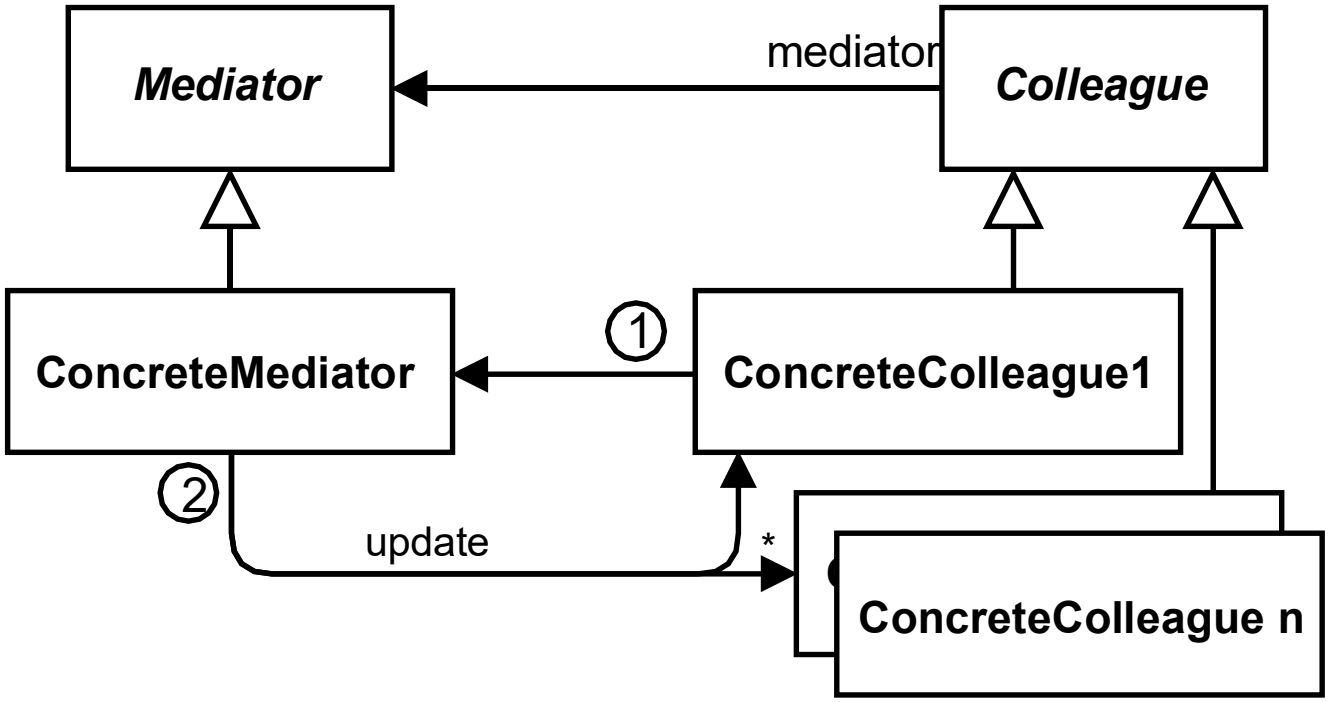
Broker

Manage dynamic communication between clients and servers in distributed systems.



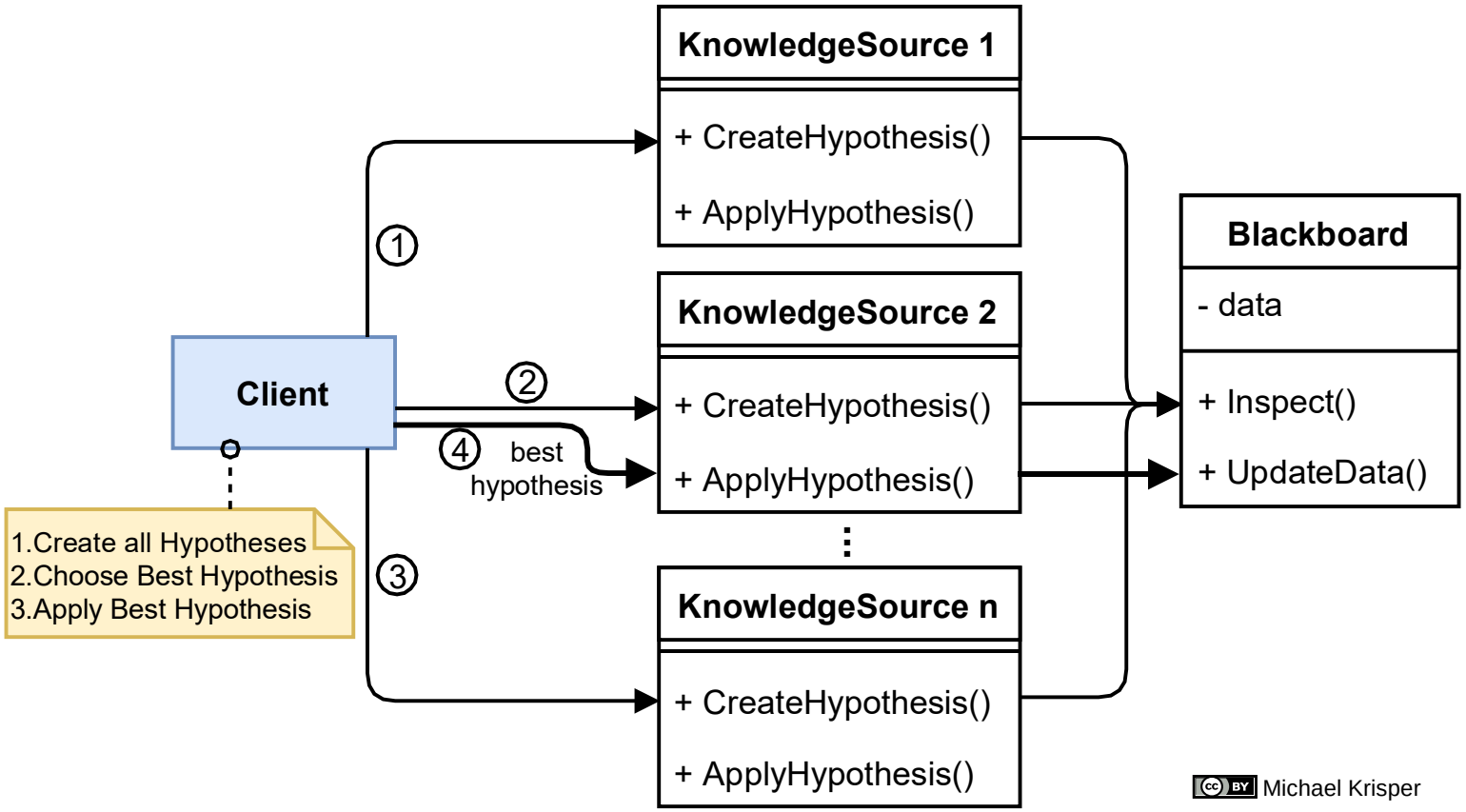
Mediator

Mediate communication between multiple objects



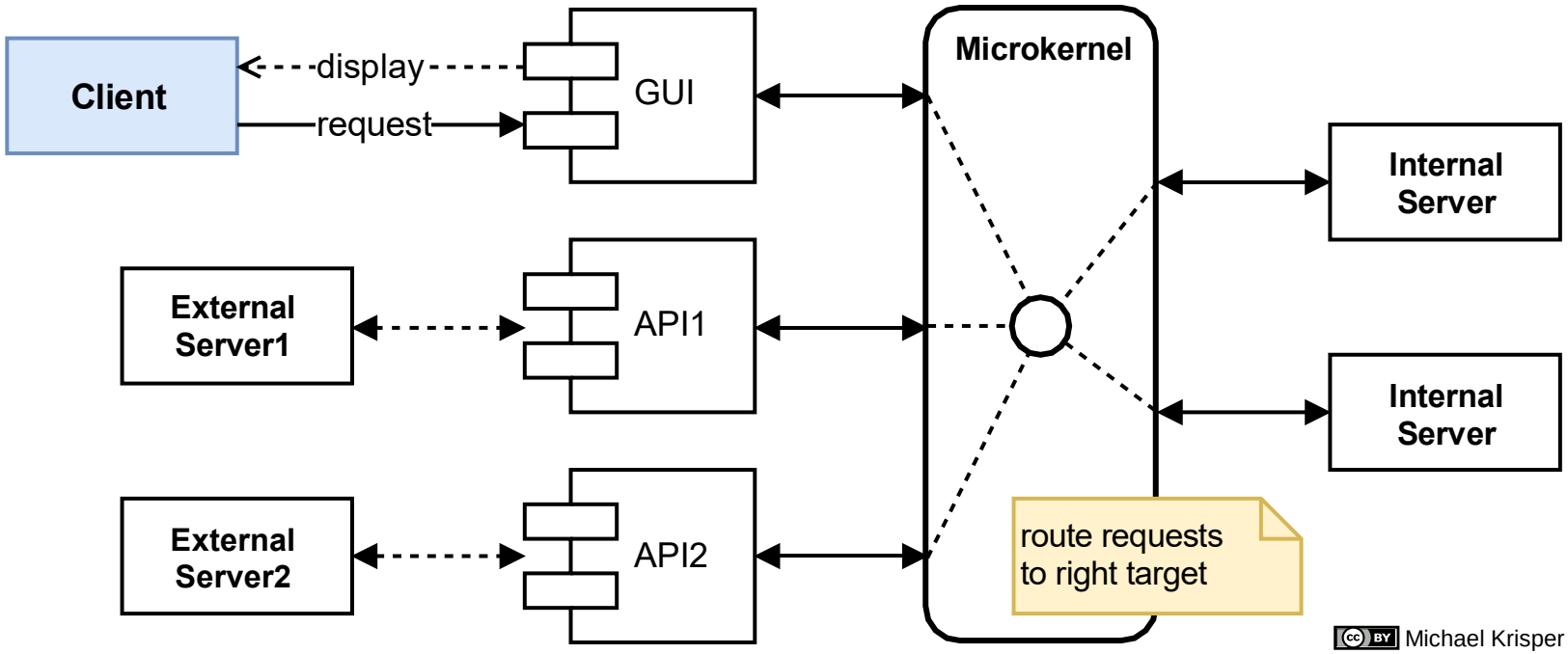
Blackboard

Collaborate on common data to get the best solution.



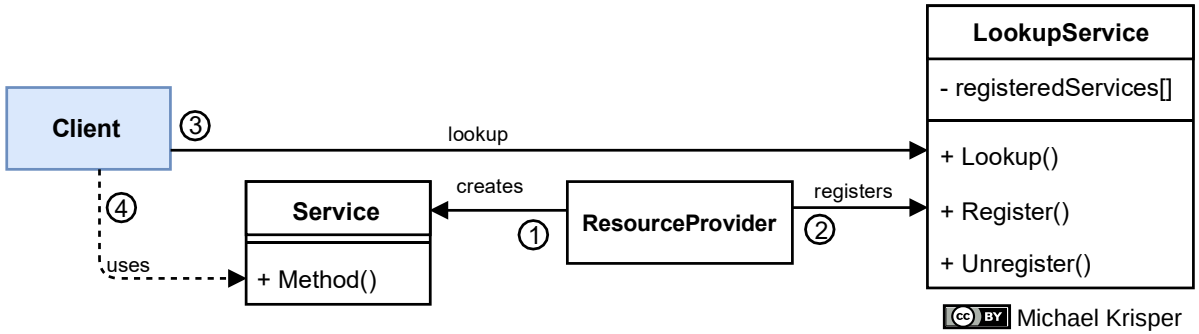
Microkernel

Route requests to the responsible components



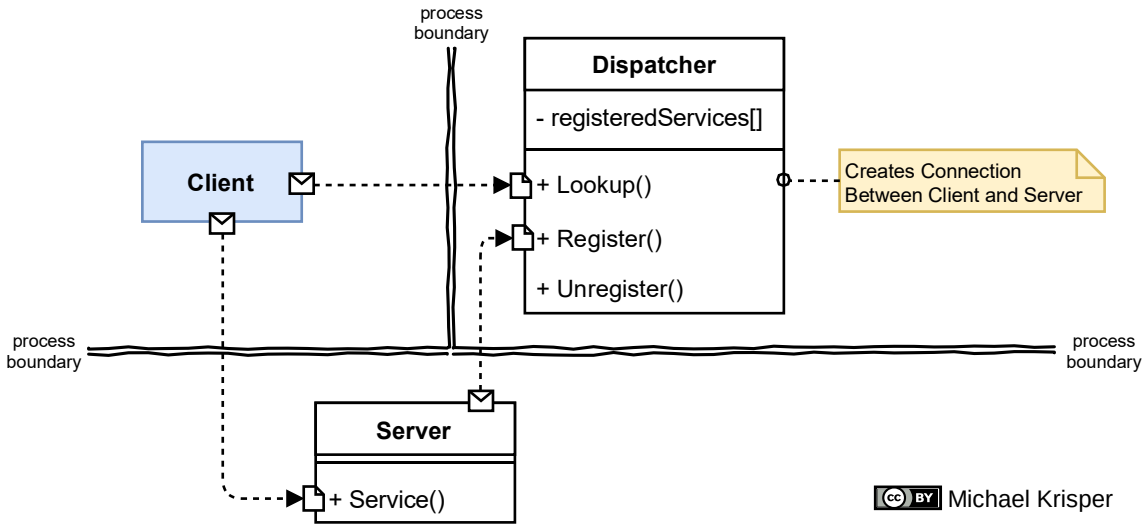
Lookup

Store service-providers and let clients search for it.



Client-Dispatcher-Server

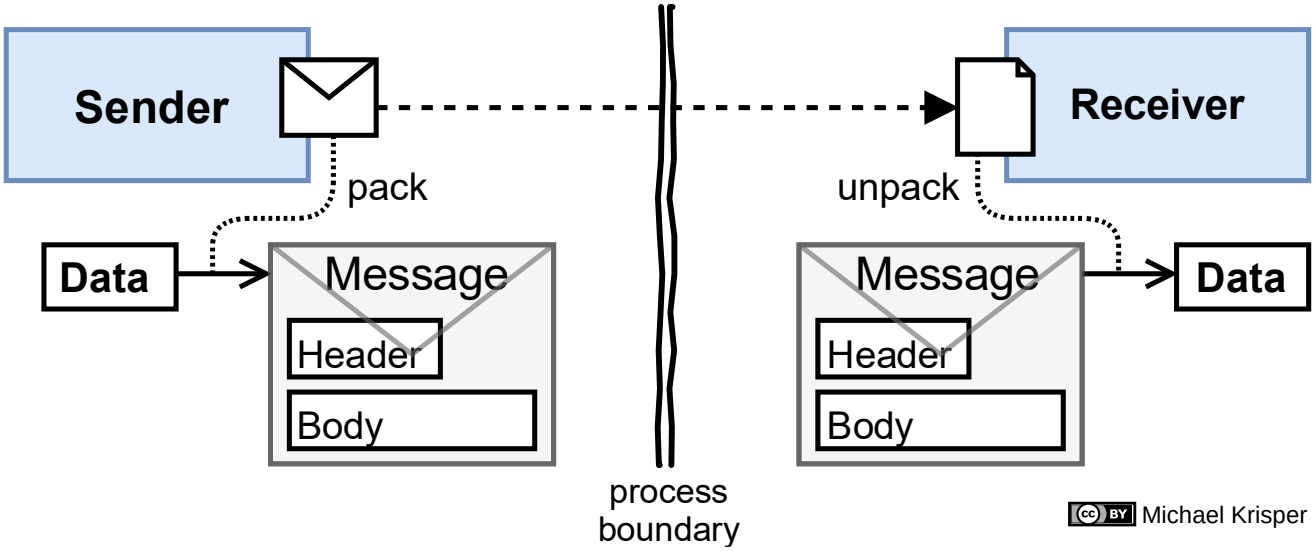
Store distributed service-providers and connect clients to them.



Messaging

Messages

Encapsulate information in a standardized way

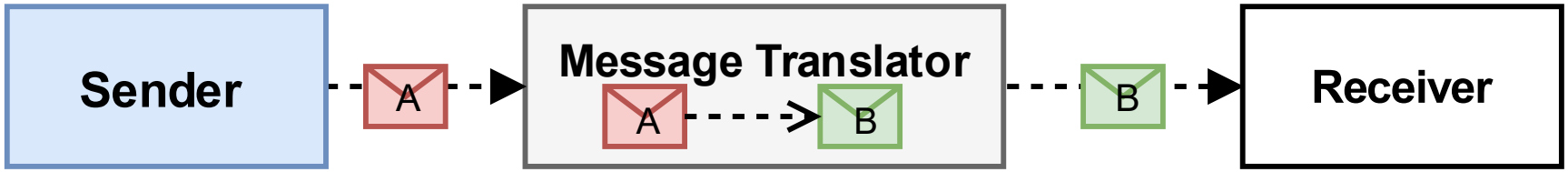


Message Endpoint

Provide functionality to send and receive messages

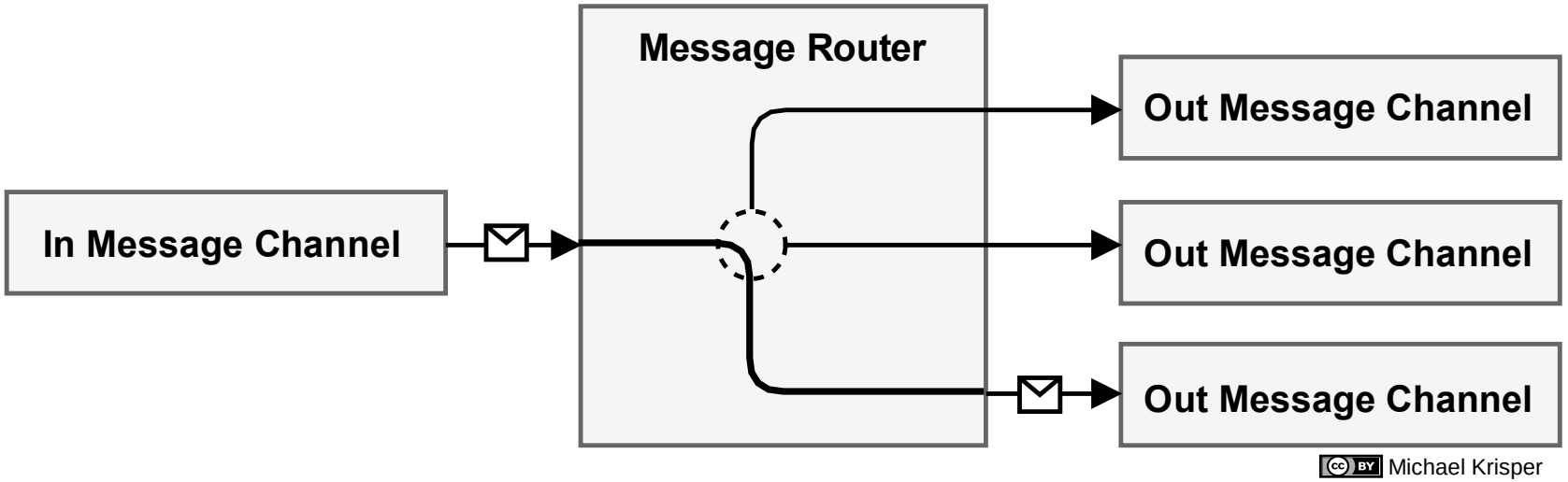
Message Translator *(Converter, Transformer)*

Translate between different message formats

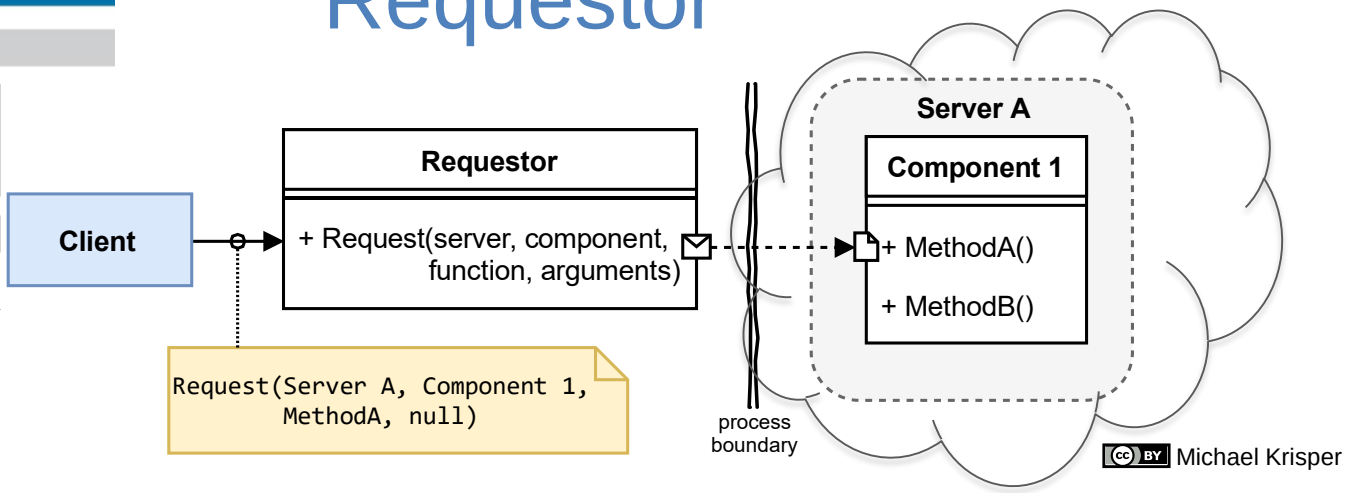


Message Router

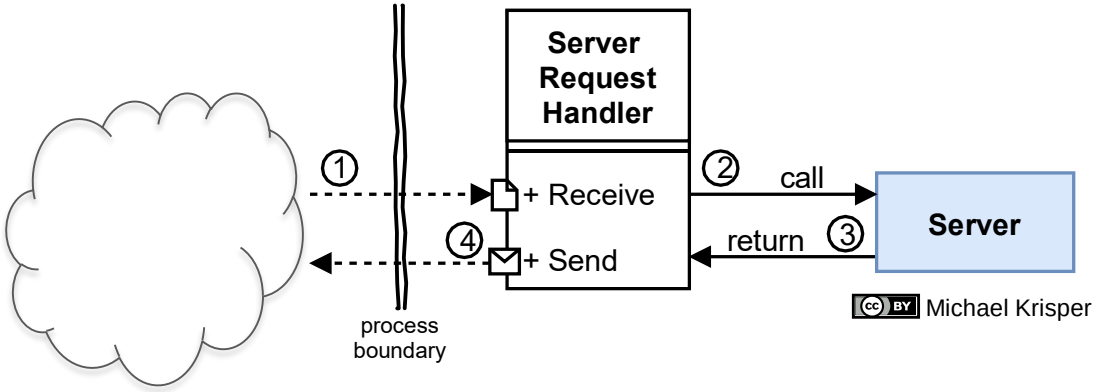
Route messages to the right receiver



Requestor



Request Handler



Summary

Communication Patterns:

- Mediator
- Blackboard
- Microkernel
- Bridge
- Broker
- Messages
- Message Endpoint
- Message Translator
- Message Router
- Request Handler
- Requestor