DigitalWorld - NexComm 2023

Leveraging Collaborative Metaprogramming for Sustainable Innovation and Co-Creation

HERWIG MANNAERT

APRIL, 2023

Universiteit Antwerpen

- Introduction
- On Evolvable Metaprogramming
- Co-Creating Software Applications
- Scaling Generative Programming
- Conclusion

LEVERAGING METAPROGRAMMING

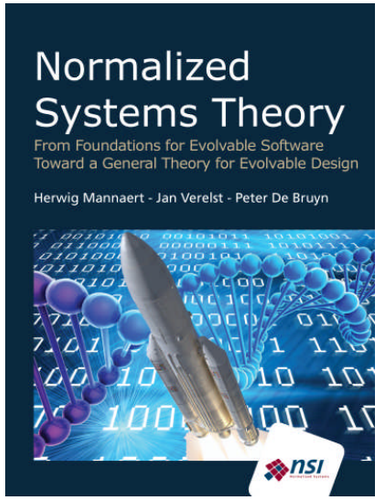
Overview

LEVERAGING METAPROGRAMMING

Overview

On myself, my colleagues, and our work

- Electronics engineer, PhD in computer vision
- Co-created *Normalized Systems Theory* on engineering and architecture of evolvable software systems, i.e., enabling software to cope with change
  - Books and papers (140 publications), and YouTube channel
  - Human adoption
    - Spin off company with 45 software engineers
    - > 60 software engineers at customers / partners
  - Software production
    - Suite of code generators and tools
    - Many software projects *AND* products, e.g.,
      - Energy monitoring suite
      - Privacy and security management suite
      - Command & Control suite for medical drone transports



- Full professor on University of Antwerp, not an esteemed researcher

On our goals

- We want to provide a theoretical framework and tools that enable the creation of software systems with unseen levels of evolvability
  - Able to cope with changing functional requirements and technologies
  - Defeating the *Law of Increasing Complexity* by Manny Lehman
- We want to achieve a critical mass in software systems to prove this
- As all real engineers, we want to contribute to our society

Our mission is to contribute to society. It is always about making things better.

- *Tim Minshall*

LEVERAGING METAPROGRAMMING

Overview

On Technology and Engineering

- The fundamental purpose of technology/engineering is to contribute to human *prosperity and well-being* by inventing *artifacts and techniques*
  - to increase productivity through *leverage effects*:
    - *Fishing:* manual ➔ spear ➔ fishing net
    - *Transport:* walking ➔ chariot ➔ car
    - *Digging:* manual ➔ shovel ➔ excavator
    - *Computing:* paper ➔ calculator ➔ computer
    - *Communicating:* courier ➔ letter ➔ message
  - to support and improve life in general:
    - Treatments and medicines to cure patients
    - Information systems to consolidate knowledge

On Technology and Engineering

- True realization of these leverage effects requires:
  - Scalability of production:
    - No huge efforts
    - No scarce resources
    - No rare capabilities
    - *Technical complexity of manufacturing*
  - Sustainability of production:
    - No depletion of resources
    - No production of poisonous substances
    - No jeopardizing of health
    - *Technical complexity of maintenance*

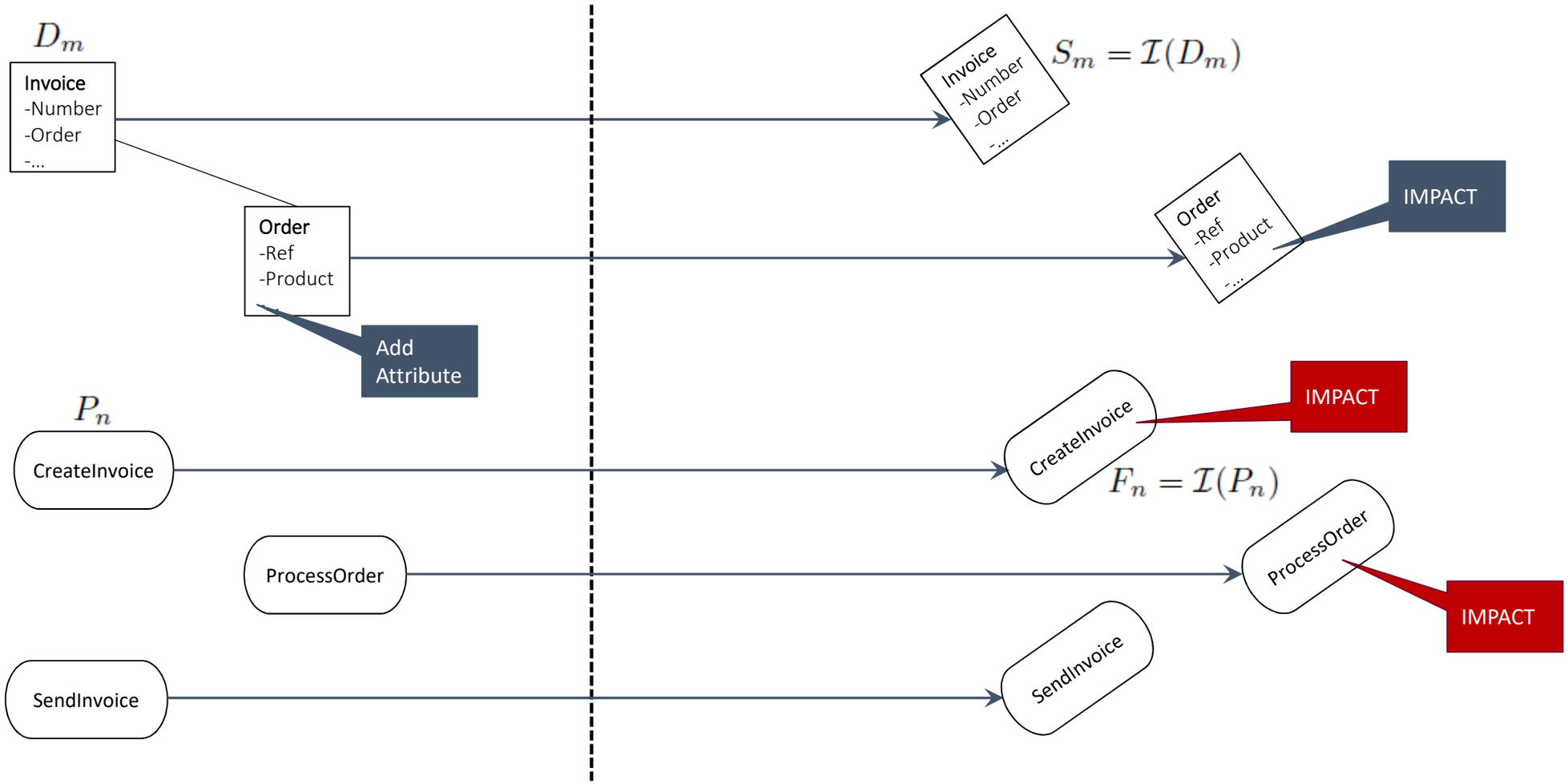
On Technology and Engineering

- In general, but *certainly in software*:
  - Scalability is related to *ability to collaborate*:
    - Control communication effort, e.g., “The Mythical Man-Month”
    - Unleash armies of volunteers, e.g., Open Source movement
    - *Technical complexity of collaborative manufacturing*
  - Sustainability is related to *ability to evolve*:
    - Upgrade and strengthen artifacts instead of destroying and producing new ones
    - Imagine evolving information systems instead of replacing legacy systems
    - *Technical complexity of evolutionary maintenance*

LEVERAGING METAPROGRAMMING

Overview

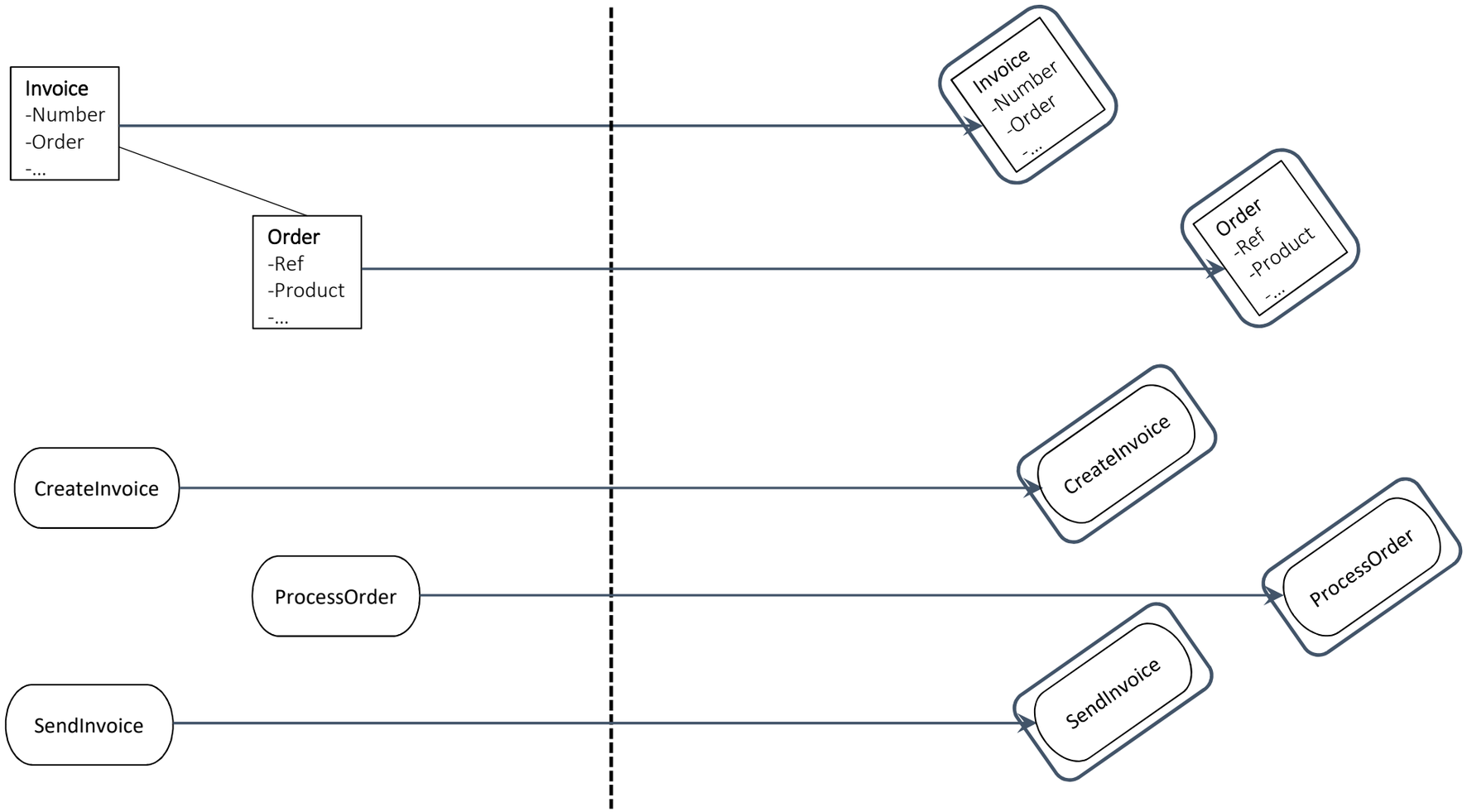
Change Ripples: A Basic Transformation



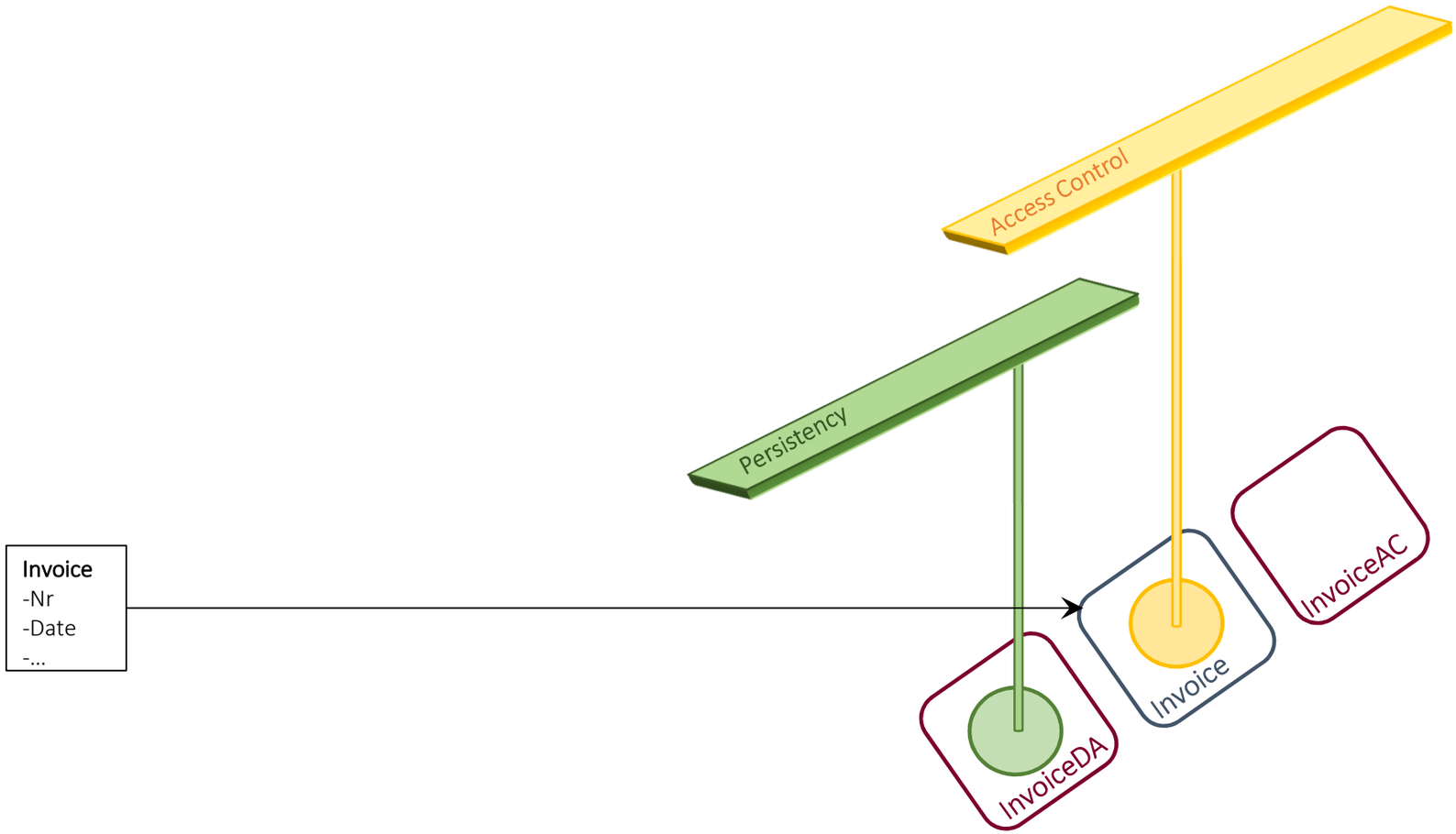
Design Theorems for Stable Software

- In order to avoid dynamic instabilities in the software design cycle, the *rippling of changes needs to be depleted or damped: a = 0*
- As these ripples create *combinations of multiple changes* for every functional change, we call these instabilities ***combinatorial effects***
- Demanding systems theoretic stability for the software transformation, leads to the derivation of ***principles*** in line with existing heuristics
- Adhering to these principles avoids dynamic instabilities, meaning that these *principles are necessary, not sufficient for systems stability*

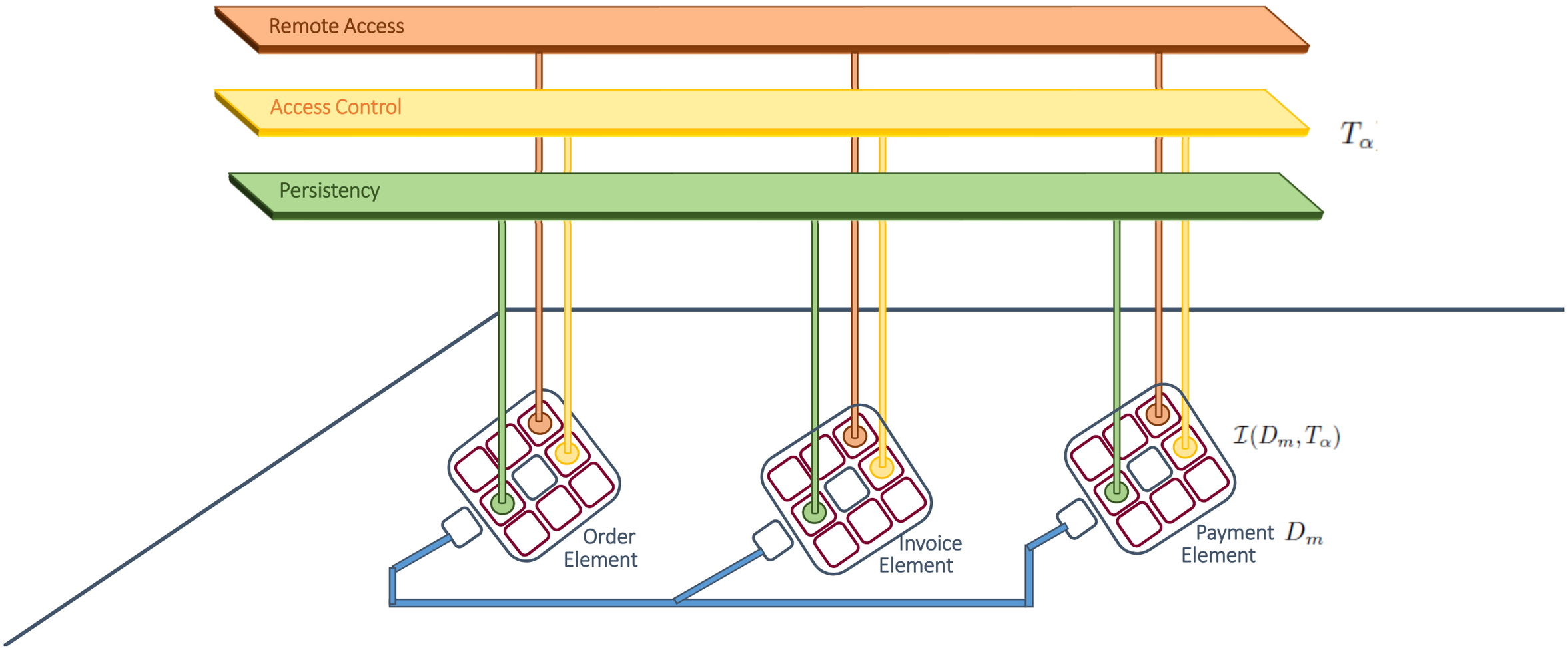
Encapsulating Basic Primitives



Separating Cross-Cutting Concerns

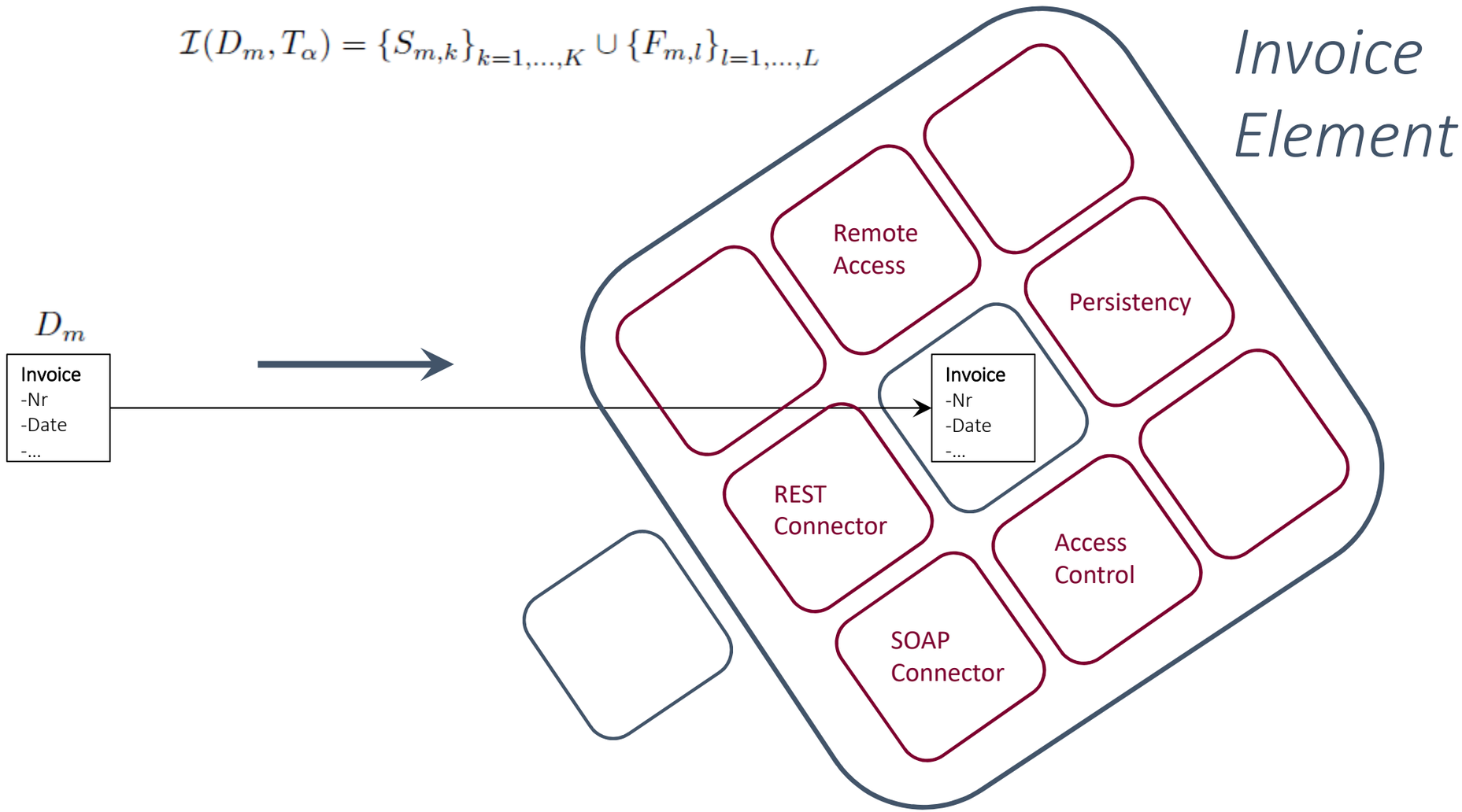


The Emergence of Elements

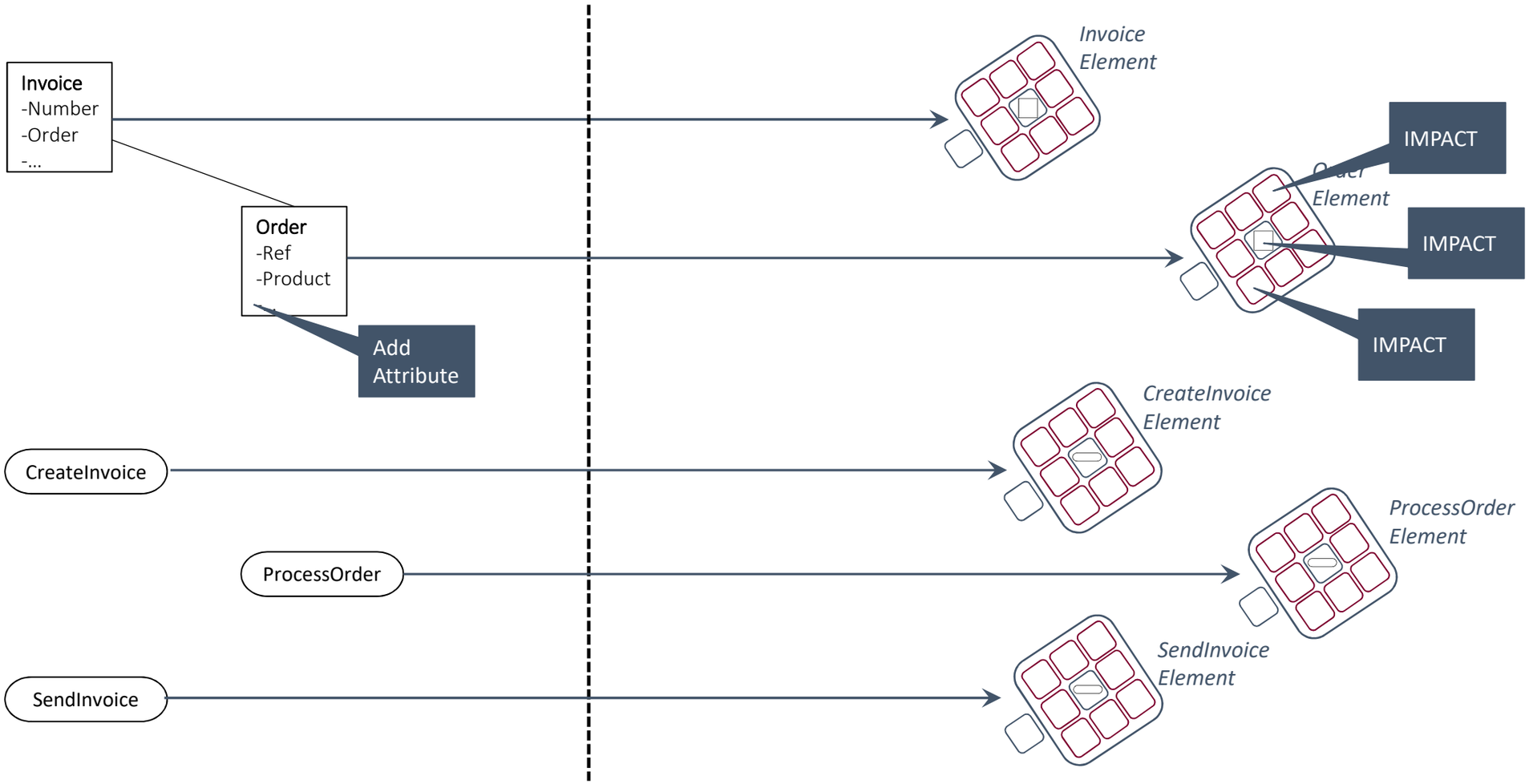


An Advanced Transformation

$$\mathcal{I}(D_m, T_\alpha) = \{S_{m,k}\}_{k=1,\ldots,K} \cup \{F_{m,l}\}_{l=1,\ldots,L}$$



An Advanced Transformation



Normalized Systems Elements

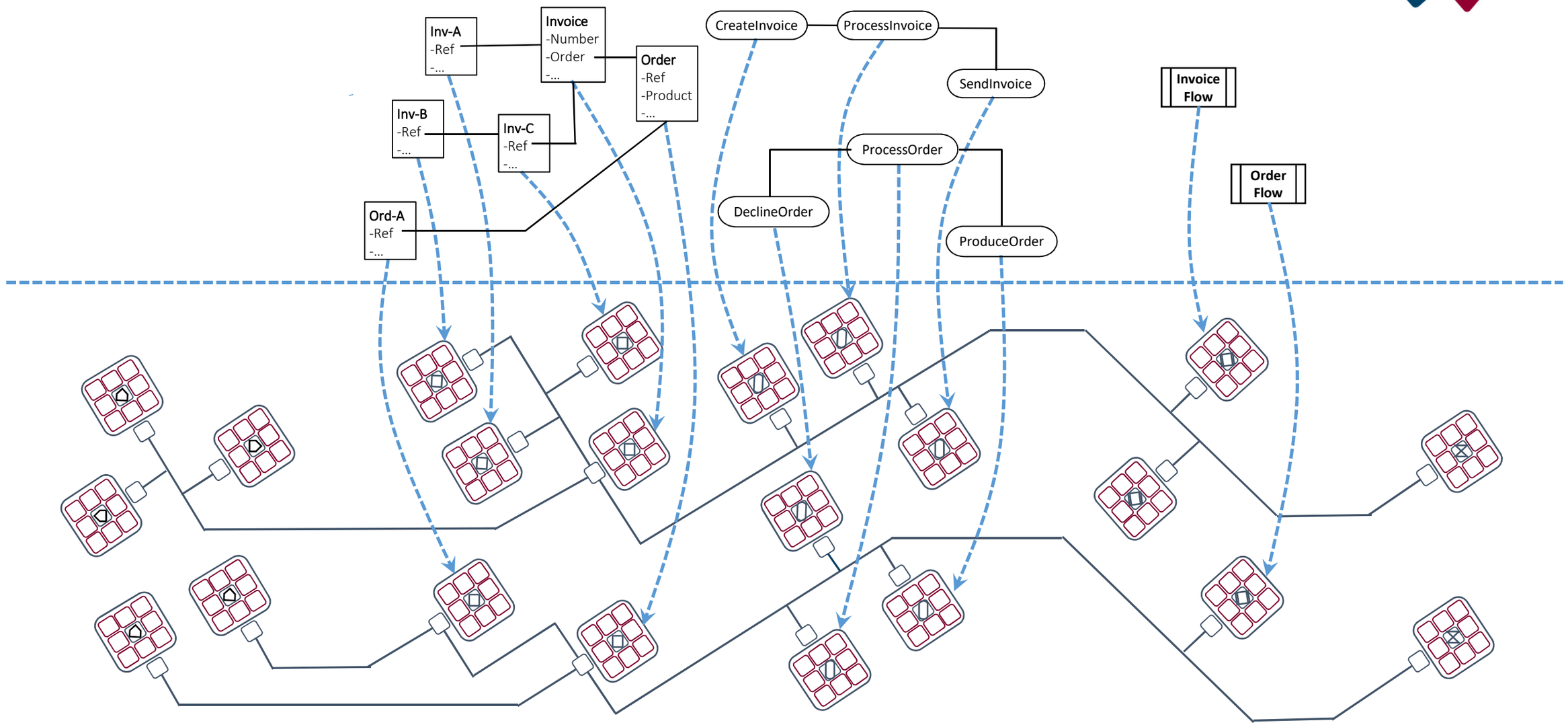
- Element structures are needed *to interconnect with CCC solutions*
- NS defines 5 types of elements, aligned with basic software concepts:
  - *Data elements*, to represent data variables and structures
  - *Task elements*, to represent instructions and/or functions
  - *Flow elements*, to handle control flow and orchestrations
  - *Connector elements*, to allow for input/output commands
  - *Trigger elements*, to offer periodic clock-like control
- It seems obvious to use code generation techniques to create instances of these recurrent element structures
- Due to its simple and deterministic nature, we refer to this process as *expansion*, and to the generators as *expanders*

- Introduction
- On Evolvable Metaprogramming
  - Stable software elements
  - Expansion and rejuvenation
  - Collaboration and meta-circularity
- Co-Creating Software Applications
- Scaling Generative Programming
- Conclusion

LEVERAGING METAPROGRAMMING

Overview

Expansion of Stable Elements



On Updating Recurring Structure

- Structure should be recurring, as variations:
  - increase complexity of codebase
  - decrease consistency in behaviour
- Recurring structure may need to vary over time:
  - new insights
  - discovery of flaws
  - changes in technologies

Structural changes may need to be applied with retroactive effect, but the efforts increase with the frequency of change.

N instances, update every K ➔ ***#updates*** = $\frac{N(N+K)}{2K}$

K=50	K=20	K=10	K=5
			5
		10	10
			15
	20	20	20
			25
		30	30
			35
	40	40	40
			45
50		50	50
			55
	60	60	60
			65
		70	70
			75
	80	80	80
			85
		90	90
			95
100	100	100	100
150	**300**	**550**	**1050**

N=100

K	Total
100	100
50	150
20	300
10	550
5	1050
2	2550
1	5050

On Updating Recurring Structure

- Recurrent stable structures are required to limit complexity and to guarantee consistency
- Recurrent stable structures need to be able to adapt over time, to overcome flaws and technology changes
- Additional custom code is inevitable and needs to be maintained across updated stable structures

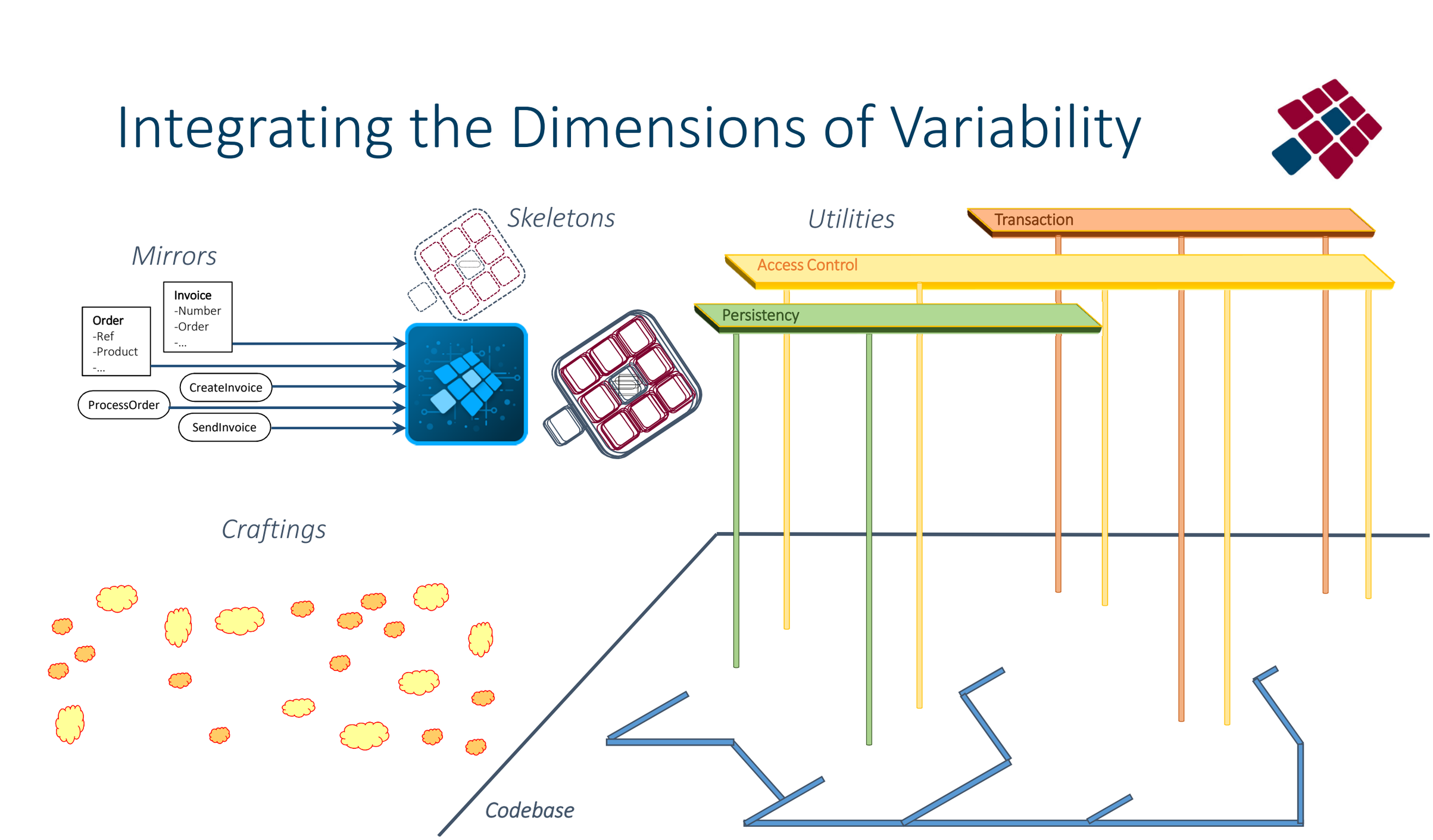
An automated mechanism is required,
providing both code generation or expansion,
and regeneration with harvesting and injection.

Variability Dimensions and Expansion

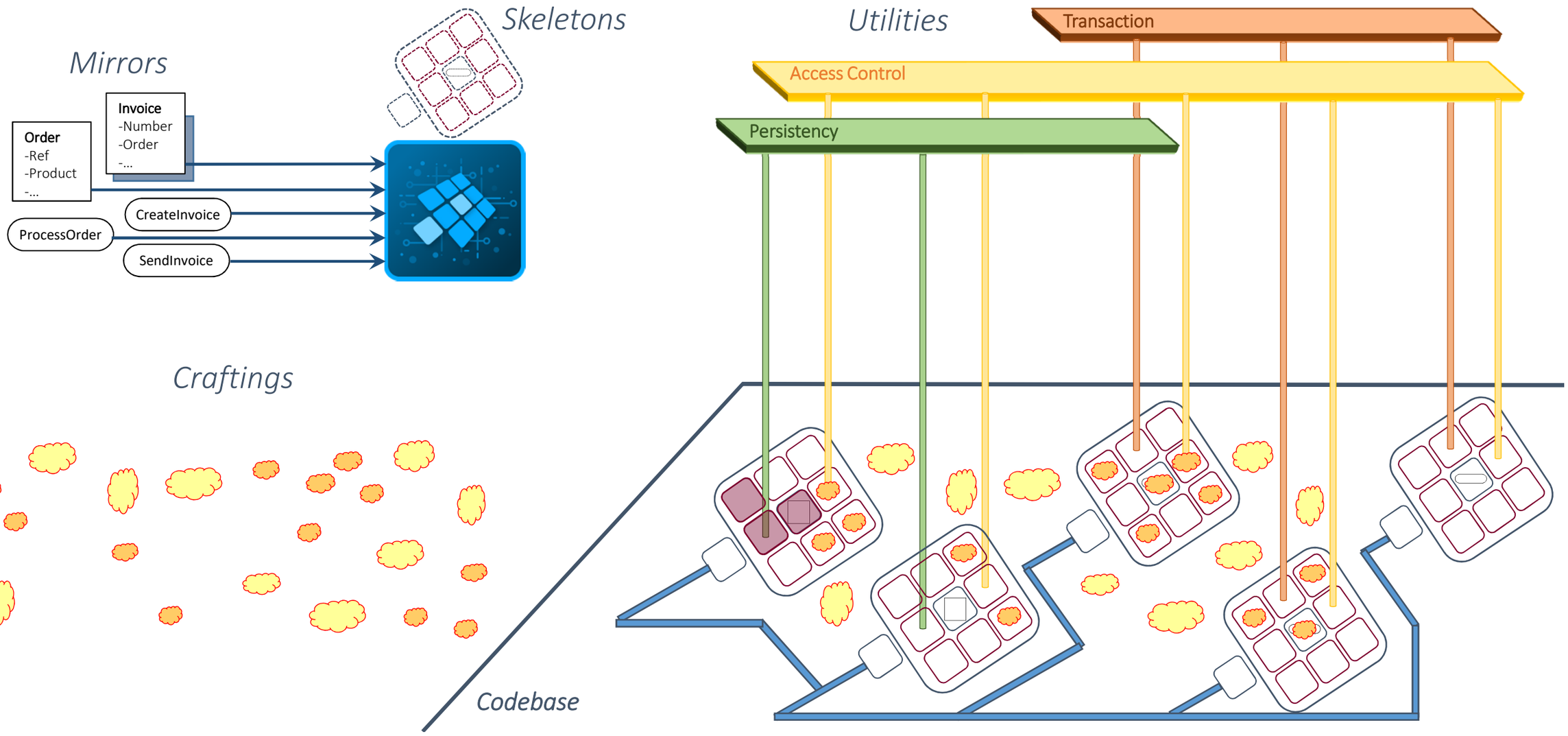
- We identify four dimensions of variability:
  - **M**odels or *mirrors*, new data attributes/relations, new elements
  - **E**xpanders or *skeletons*, new or improved implementations of concerns
  - **I**nfrastructure or *utilities*, new frameworks to implement various concerns
  - **C**ustom code or *craftings*, new or improved implementations of tasks, screens
- *If separated and well encapsulated*
  - Number of versions to maintain is *additive*: $\#V = \#M + \#E + \#I + \#C$
  - Number of versions available is *multiplicative*: $\#V = \#M \times \#E \times \#I \times \#C$
  - Where the same holds within any individual dimensions,

    e.g., infrastructure dimension: $\#I = \#G \times \#P \times \#B \times \#T$

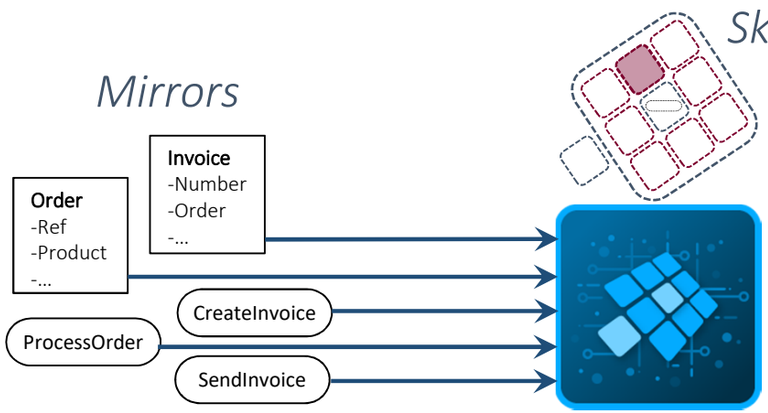
Integrating the Dimensions of Variability

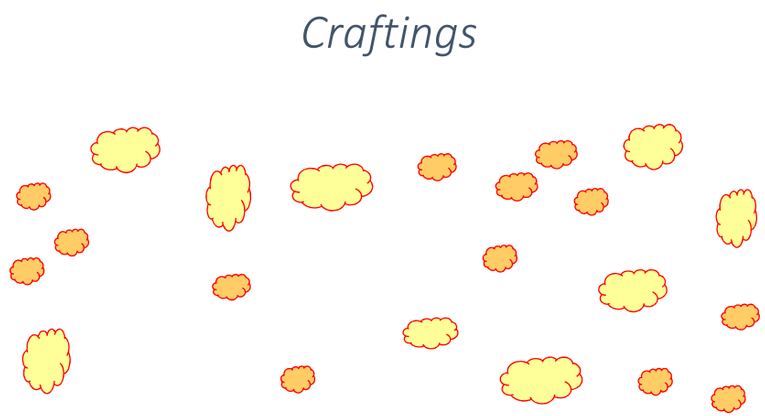


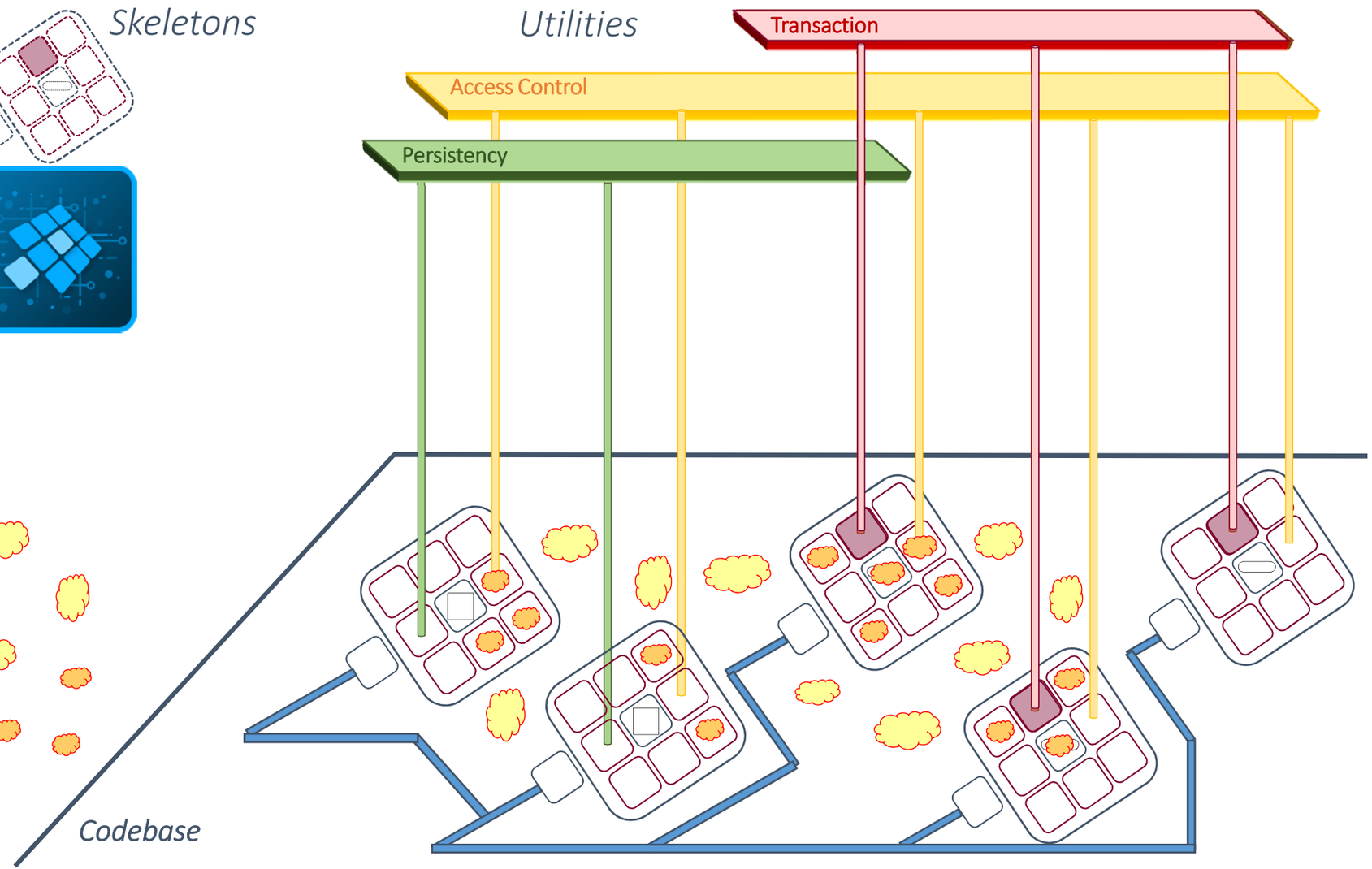
Change Dimension 1: The Mirrors



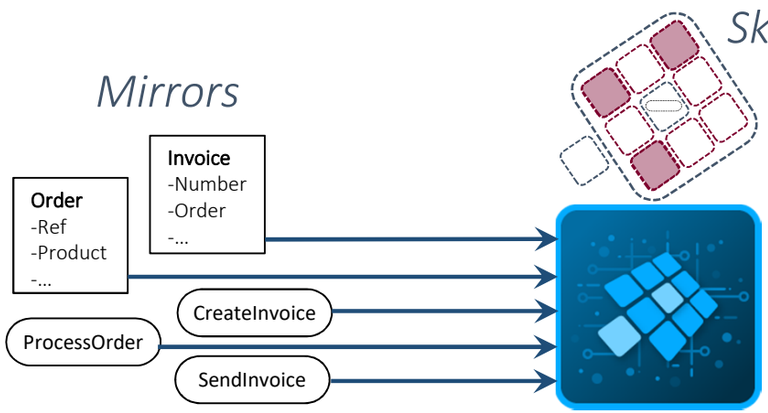
Change Dimension 2: The Utilities

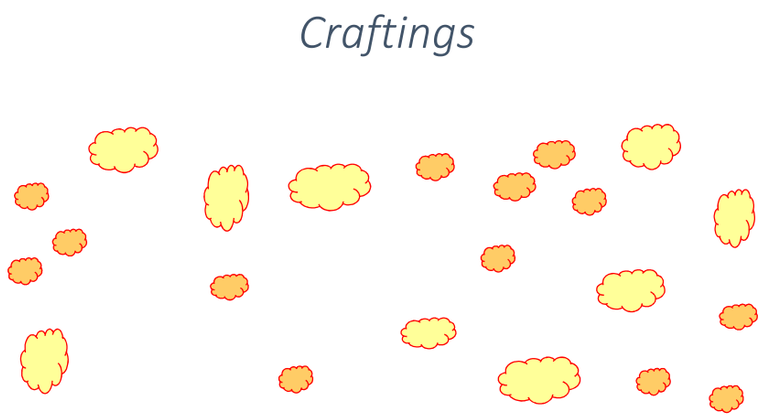


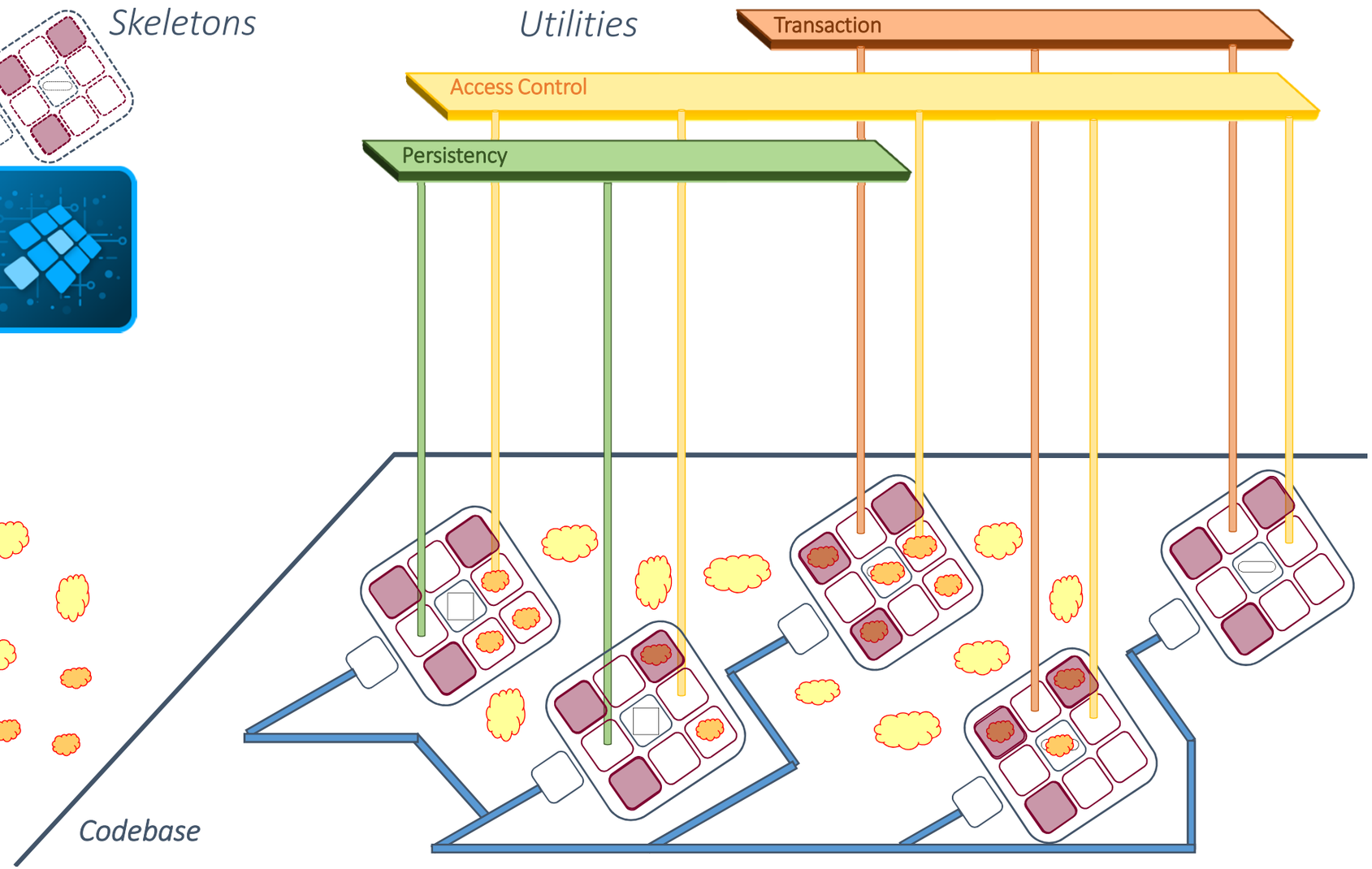




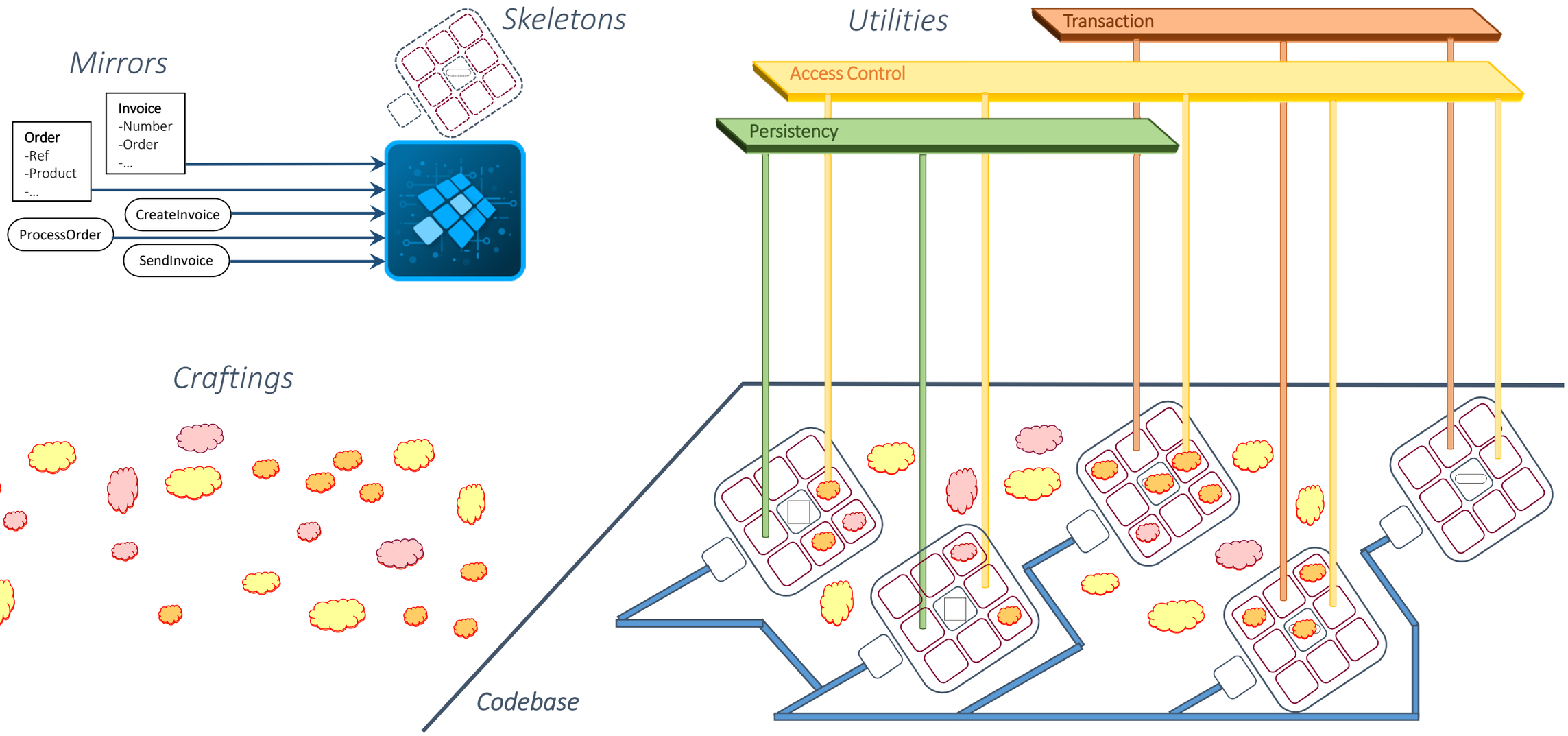
Change Dimension 3: The Skeletons



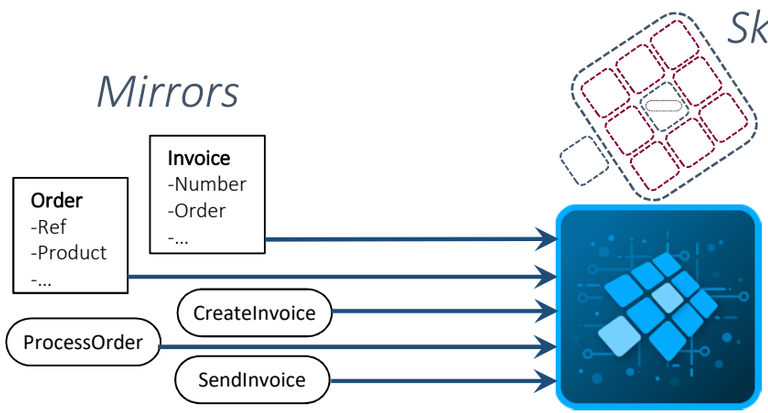


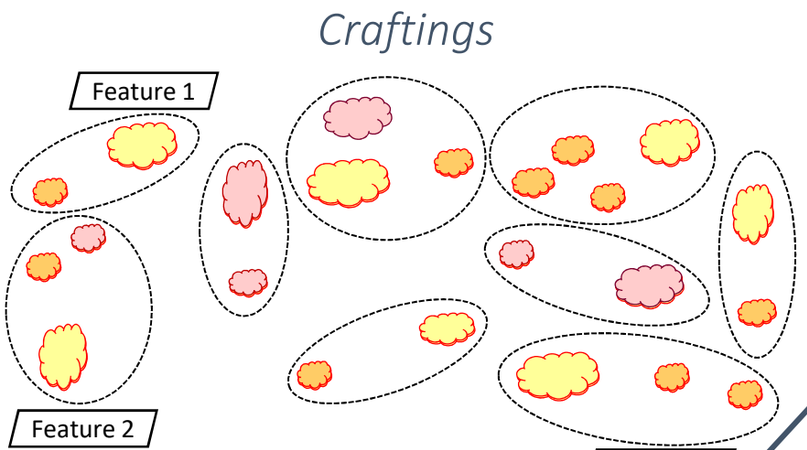


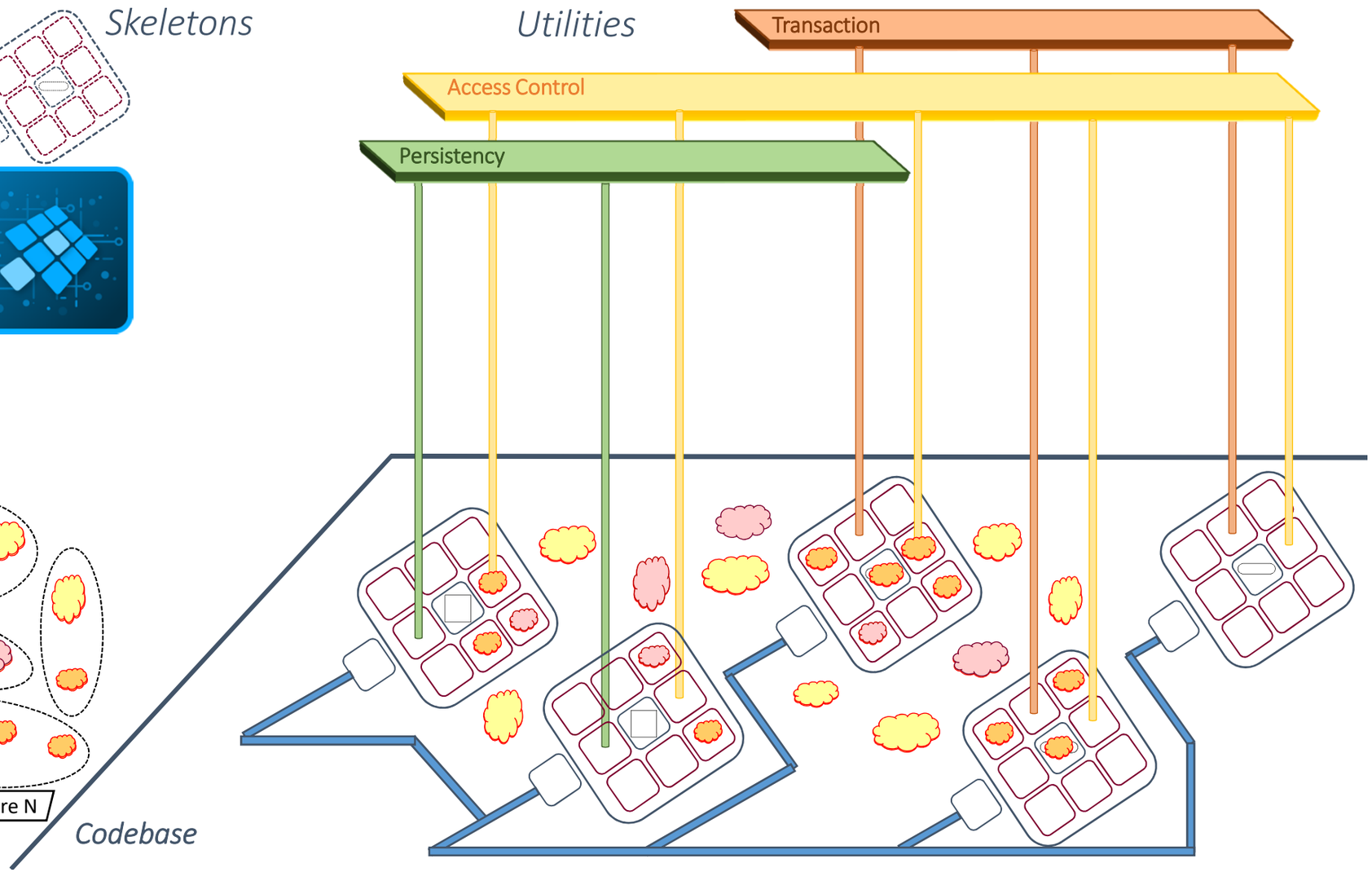
Change Dimension 4: The Craftings



Change Dimension 4: The Craftings

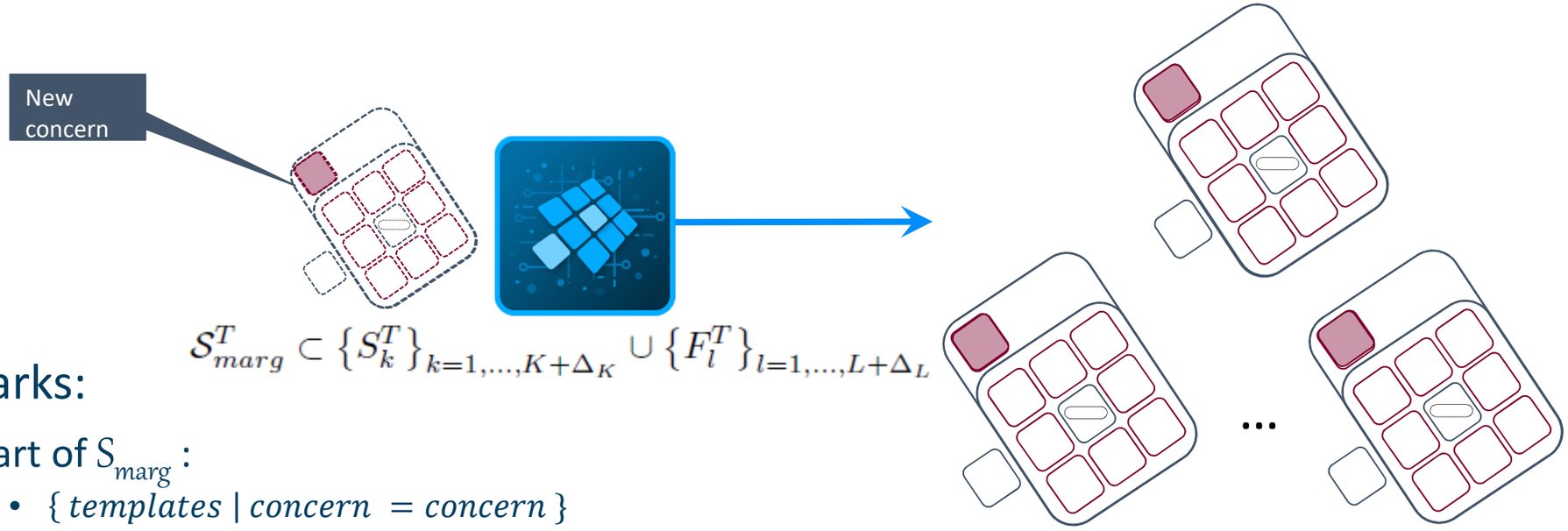






Sustaining an Evolving Utility Landscape

Ex Ante 11 – Expansion An additional concern for an of element can be made available for all information systems in a stable way.

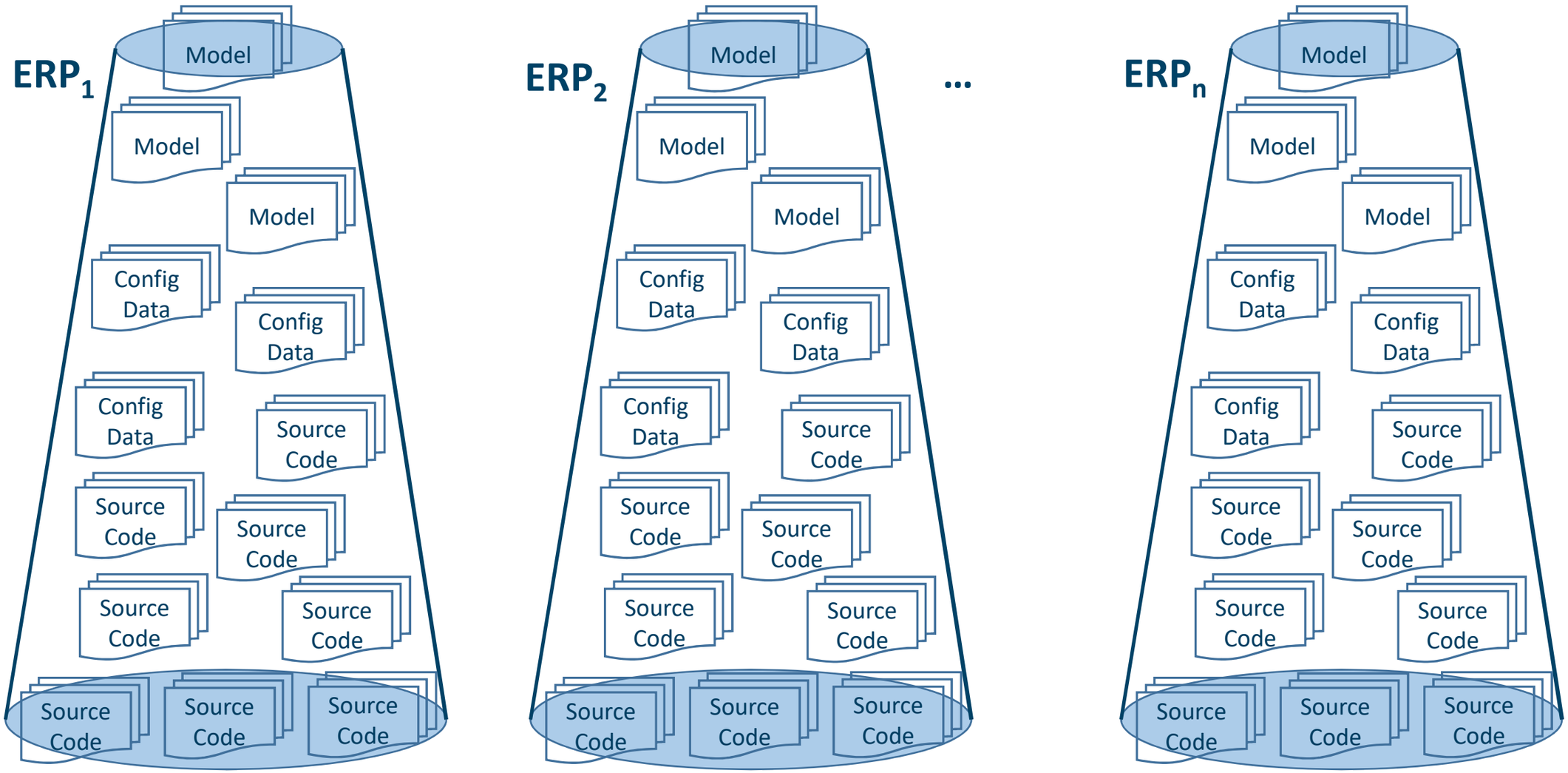


- Remarks:
  - Part of S_{marg} :
    - $\{\, templates \mid concern \;= concern \,\}$
  - Configuration :
    - Define setting or option

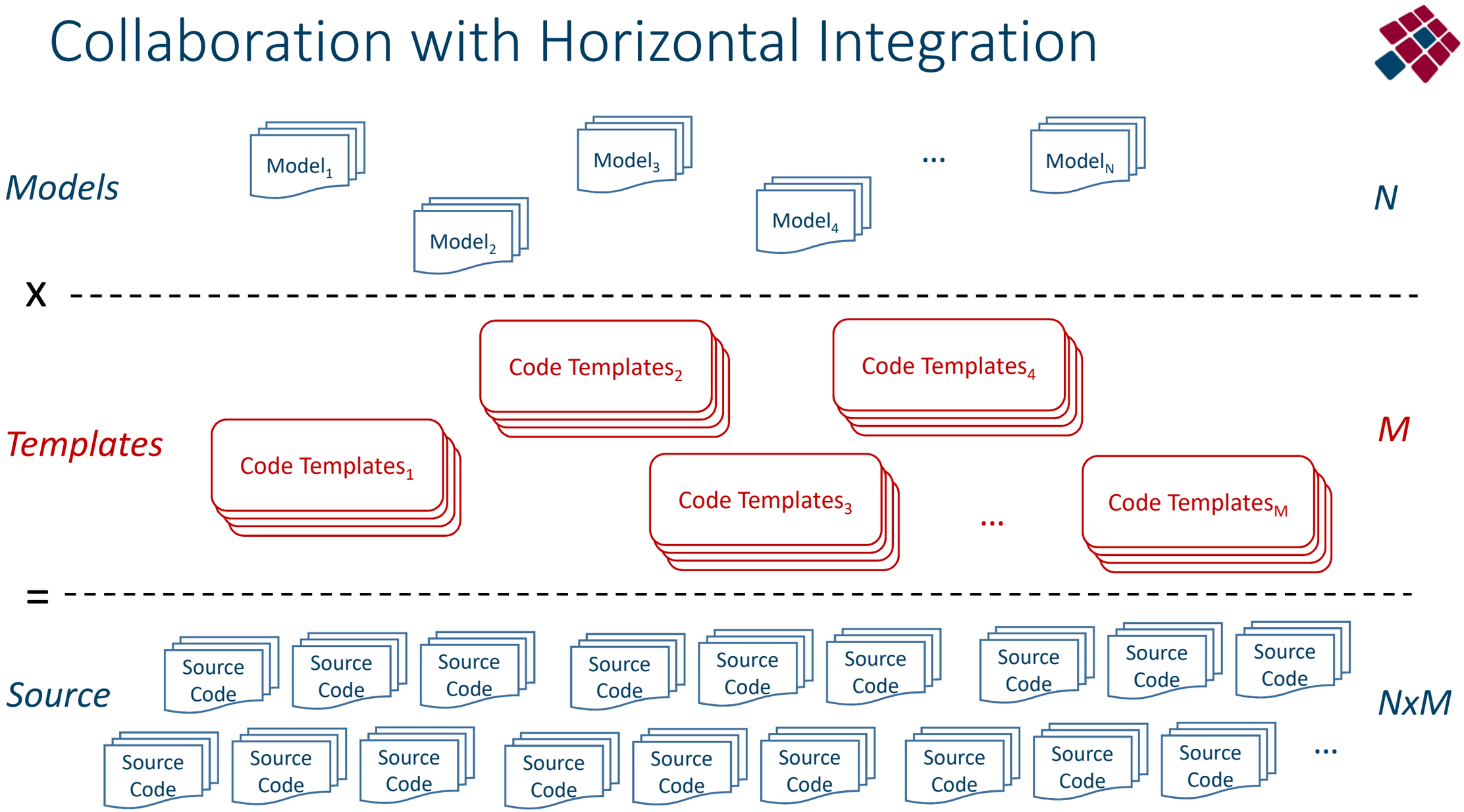
LEVERAGING METAPROGRAMMING

Overview

Collaboration with Vertical Integration



Collaboration with Horizontal Integration

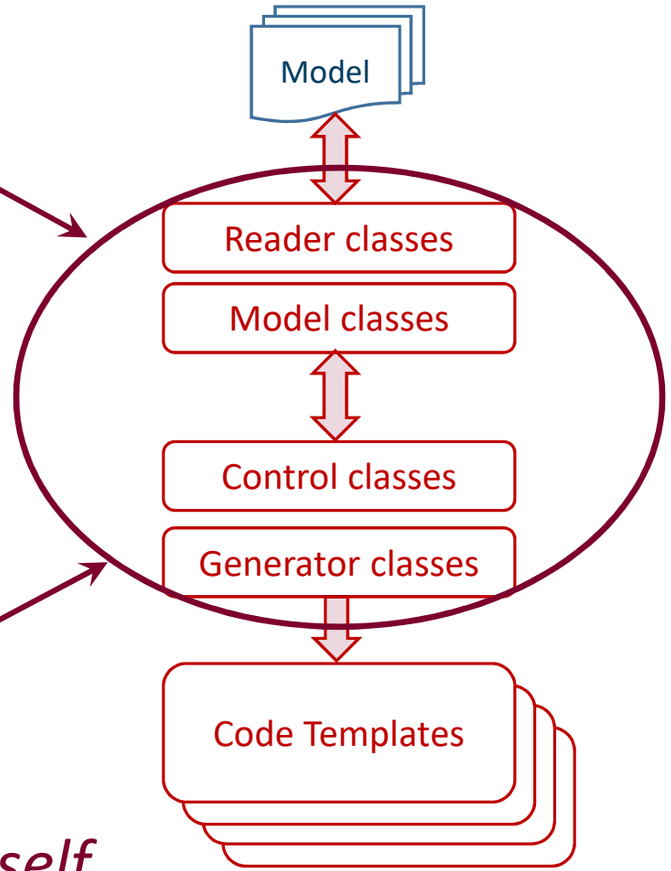


Collaboration with Horizontal Integration

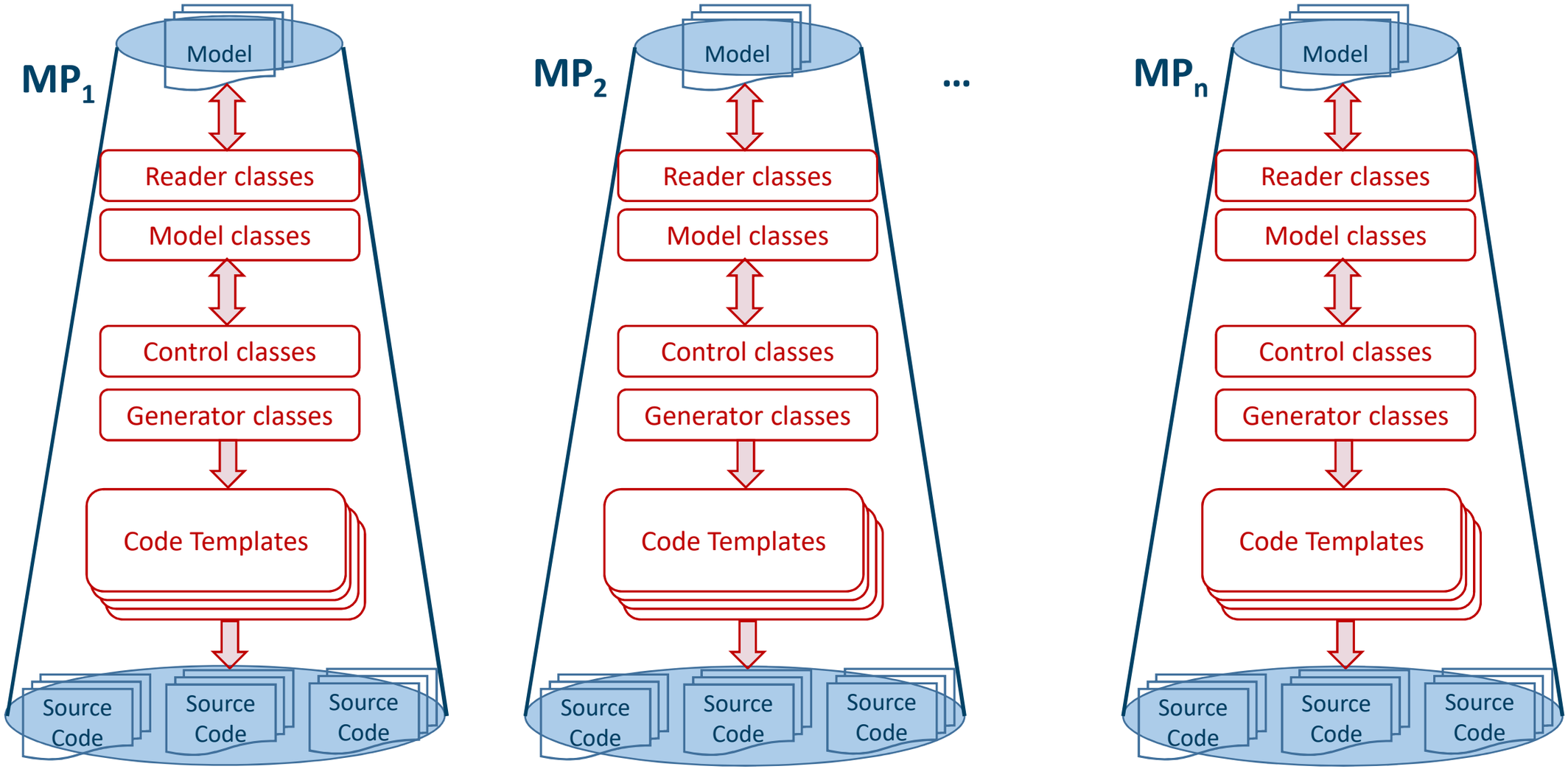
- Modelers would collaborate on domain models
  - Improving model versions and variants
  - Adding new functional business modules
- (Meta)programmers would collaborate on templates
  - Improving and integrating new insights
  - Adding and improving cross-cutting concerns
  - Supporting modified and new technologies
- Software systems would be based on metaprogramming
  - for a selected version of a domain model
  - using a specified set of coding templates
  - targeted at specific technology platform

Metaprogramming with Vertical Integration

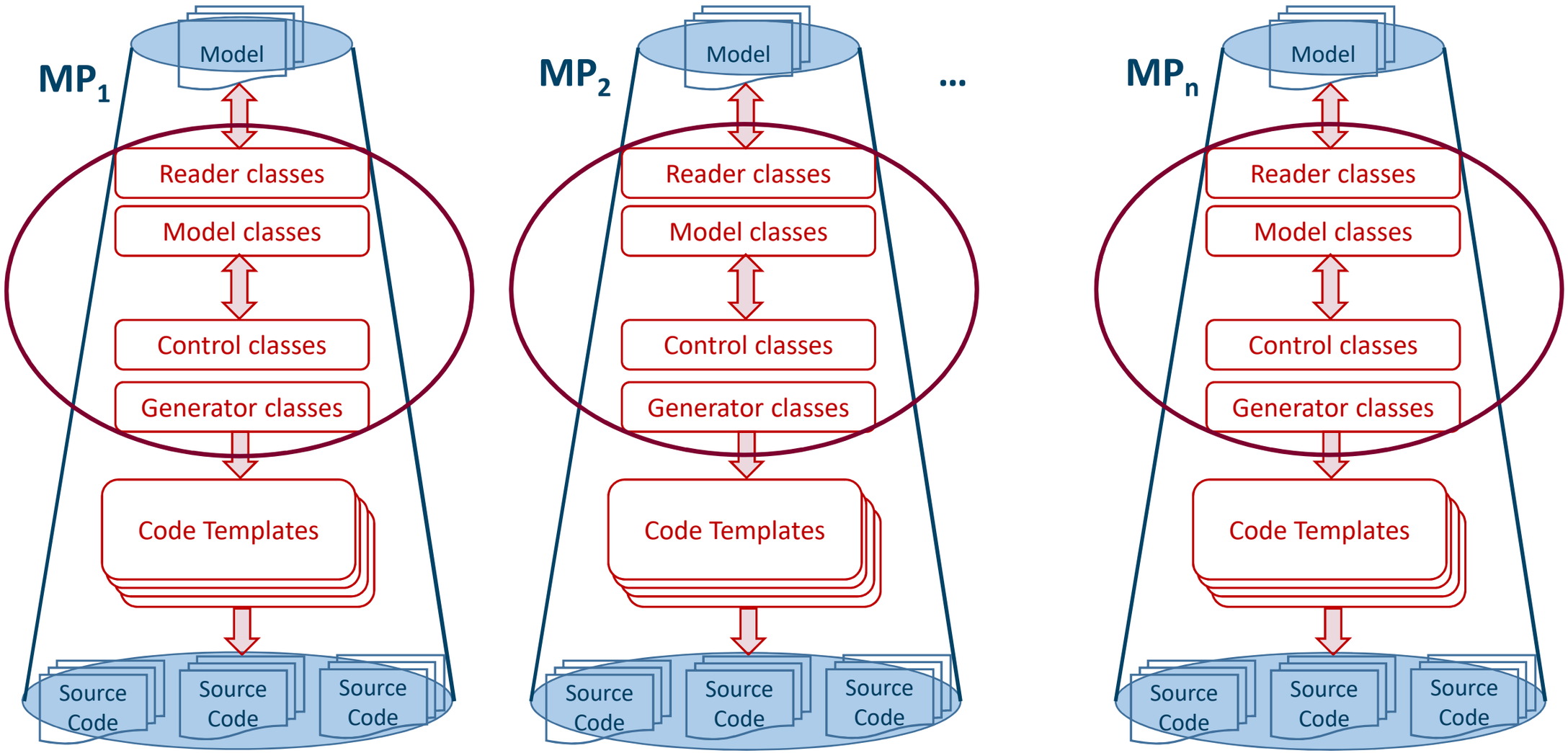
- You also have to maintain the meta-code
  - Consists of several modules
  - Is in general not trivial to write
- Will face growing number of implementations:
  - Different versions
  - Multiple variants
  - Various technology stacks
- Will have to adapt itself to:
  - Evolutions of its underlying technology
    - Which even may become obsolete
- *Meta-Circularity: meta-code that (re)generates itself*



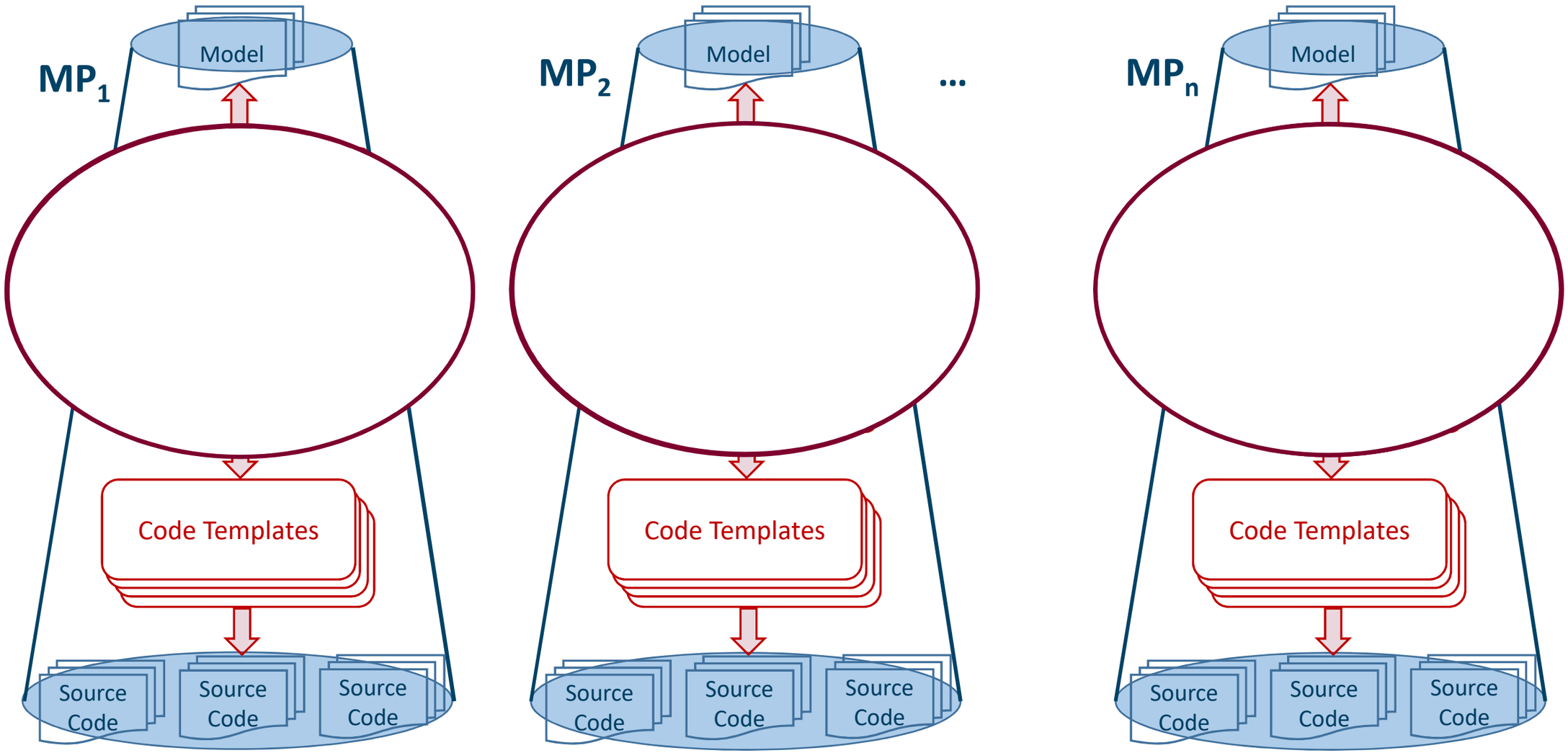
Metaprogramming with Vertical Integration



Metaprogramming with Vertical Integration



Metaprogramming with Vertical Integration



Metaprogramming with Horizontal Integration

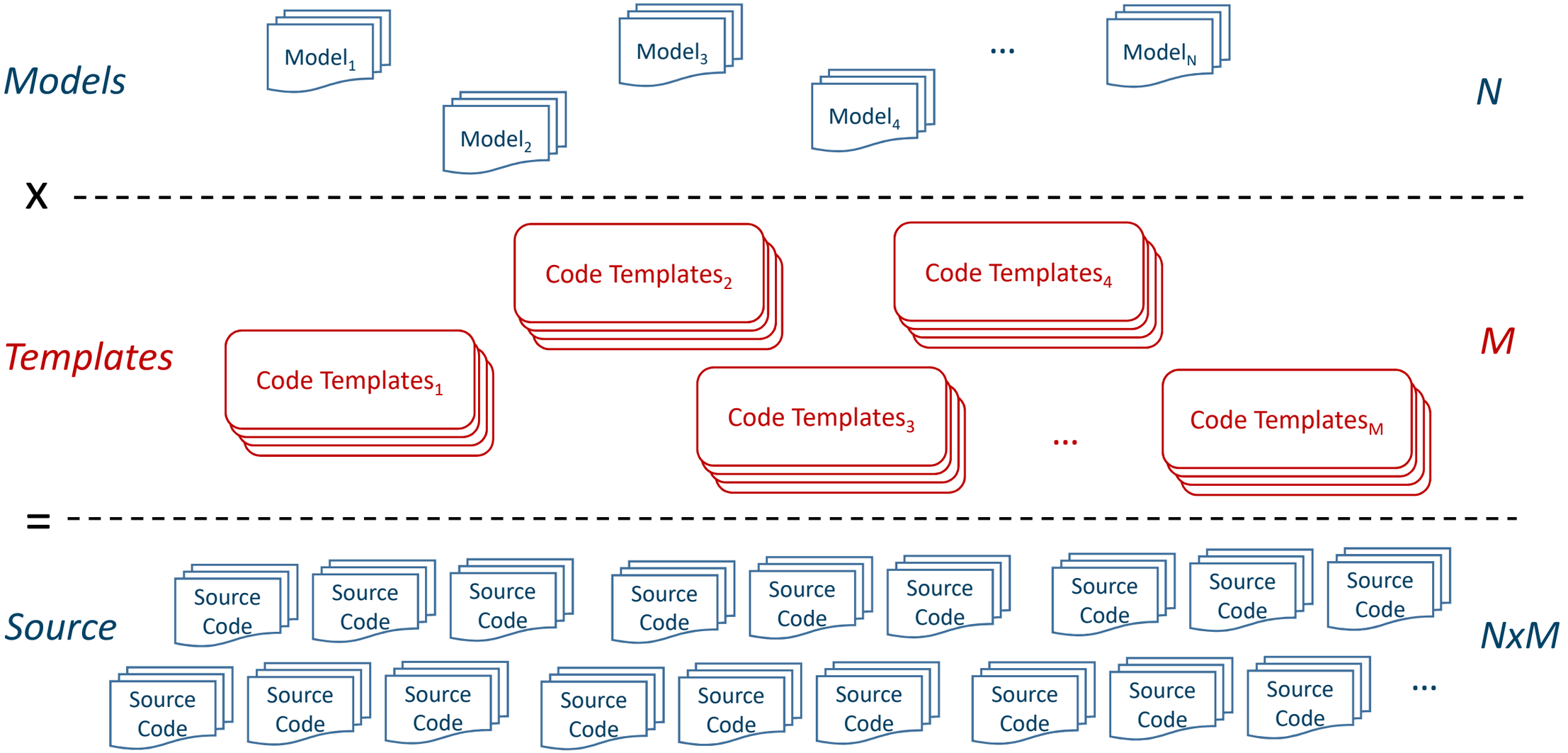
- Just like the programming code, the *metaprogramming code needs to be rejuvenated* and adapted to new technologies
- Just like the application skeletons, *the structure of the models itself needs to be able to evolve*

  → you need the models and the generative software to evolve

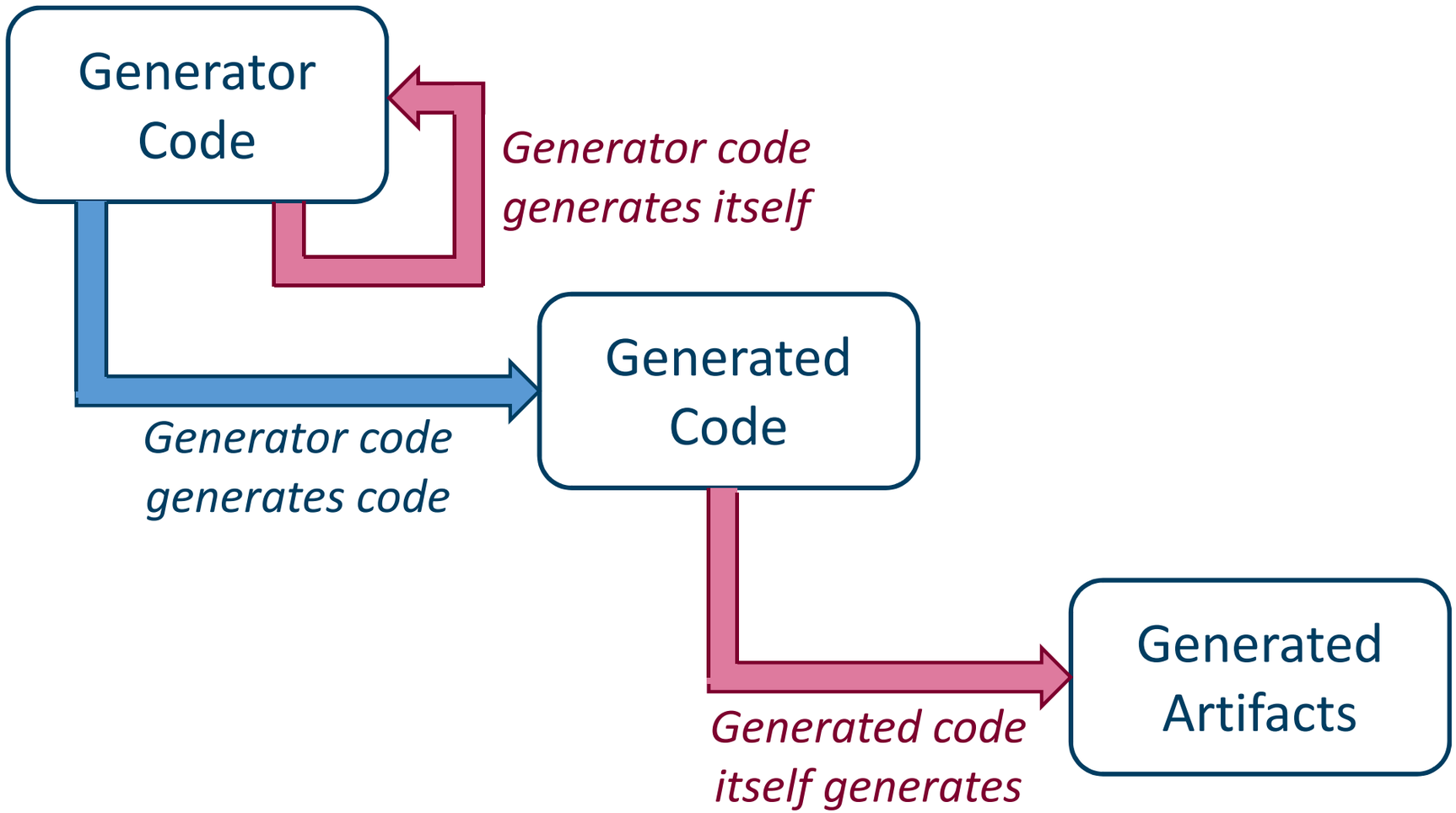
  → you need *meta-circular metaprogramming*

    - *i.e. generative programming that includes the generative code*

On Horizontal Integration in Metaprogramming



Bootstrapping Meta-Circular Metaprogramming



- Introduction
- On Evolvable Metaprogramming
- Co-Creating Software Applications
  - The case for co-creation
  - Some real-world examples
- Scaling Generative Programming
- Conclusion

LEVERAGING METAPROGRAMMING

Overview

The Case for Application Co-Creation

- The creation of a software application and/or product requires a large amount of domain knowledge
- The creation of a software application and/or product requires a large amount of software know-how and infrastructure
- Starting from a domain company, the software can either be
  - Outsourced to an IT services company
    - Can be quite *expensive*
  - Insourced by hiring software professionals
    - Requires a critical mass and can be *vulnerable*
- Both scenarios easily lead to external capital, restricting various degrees of freedom such as the *time frame* and the founders' *original goals*

The Case for Application Co-Creation

- We envisage a long-term co-creation partnership featuring:
  - a *shared center for software* know-how and infrastructure
  - intense *collaboration on the models* between domain experts and software engineers supported by the metaprogramming environment
  - continuous *rejuvenation of the software application* based on the based on the re-generation capabilities of the metaprogramming environment
  - build-up over time of a dedicated software team at the domain company to *consolidate knowledge* and balance the shared software center
  - *slow start of costs* and expenditures to allow for organic business development, including reduced rates and partial conversion to stocks
  - maximum internal capacity of starting co-creation partnerships at 20%

- Introduction
- On Evolvable Metaprogramming
- Co-Creating Software Applications
  - The case for co-creation
  - Some real-world examples
- Scaling Generative Programming
- Conclusion

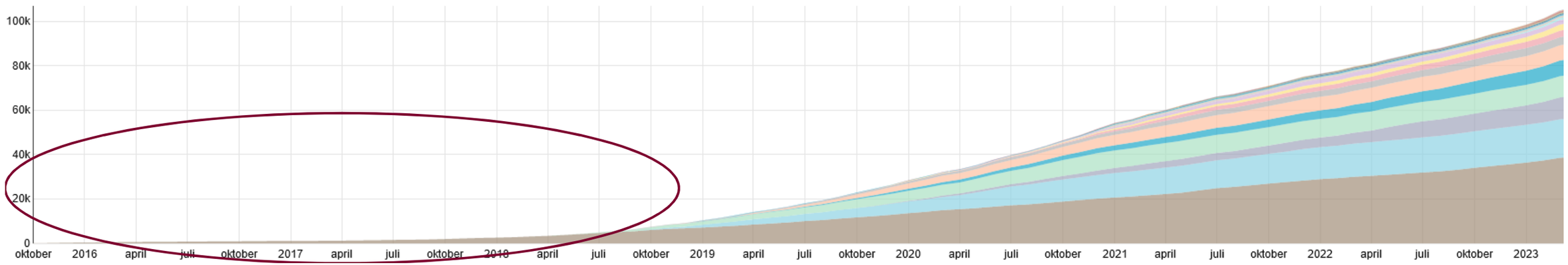
LEVERAGING METAPROGRAMMING

Overview

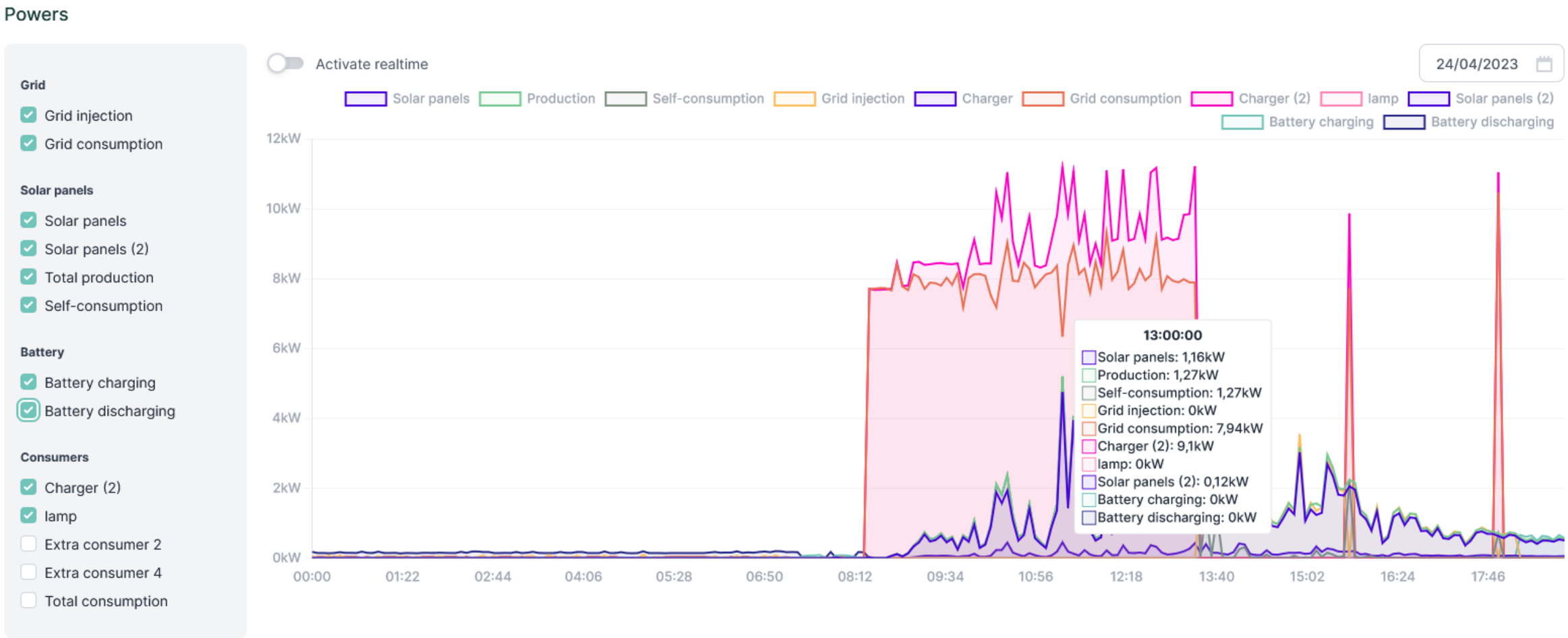
Case 1: Sustainable Energy Monitoring

- Monitoring and management of residential energy installations, such as solar panels, heat pumps, batteries and charging stations
- Starting in 2015 with a market taking off extremely slow
- Taking off with 1 domain expert, 1 software engineer in shared center
- Evolving to 2 domain experts, 3 software engineers in shared center
- First software engineers will now be insourced

cast4all TECHNOLOGIES



Case 1: Sustainable Energy Monitoring



Case 2: Medical Transport via Drones

- Command & Control center for medical transport via drones, integrating hospital systems, air traffic control systems, and drone control systems
- Starting in 2015 before even the required legislation was put in place
- Taking off with 1 domain expert, 1 software engineer in shared center
- Evolving to 3-5 domain experts, 4 software engineers in shared center
- First software engineers will now be insourced

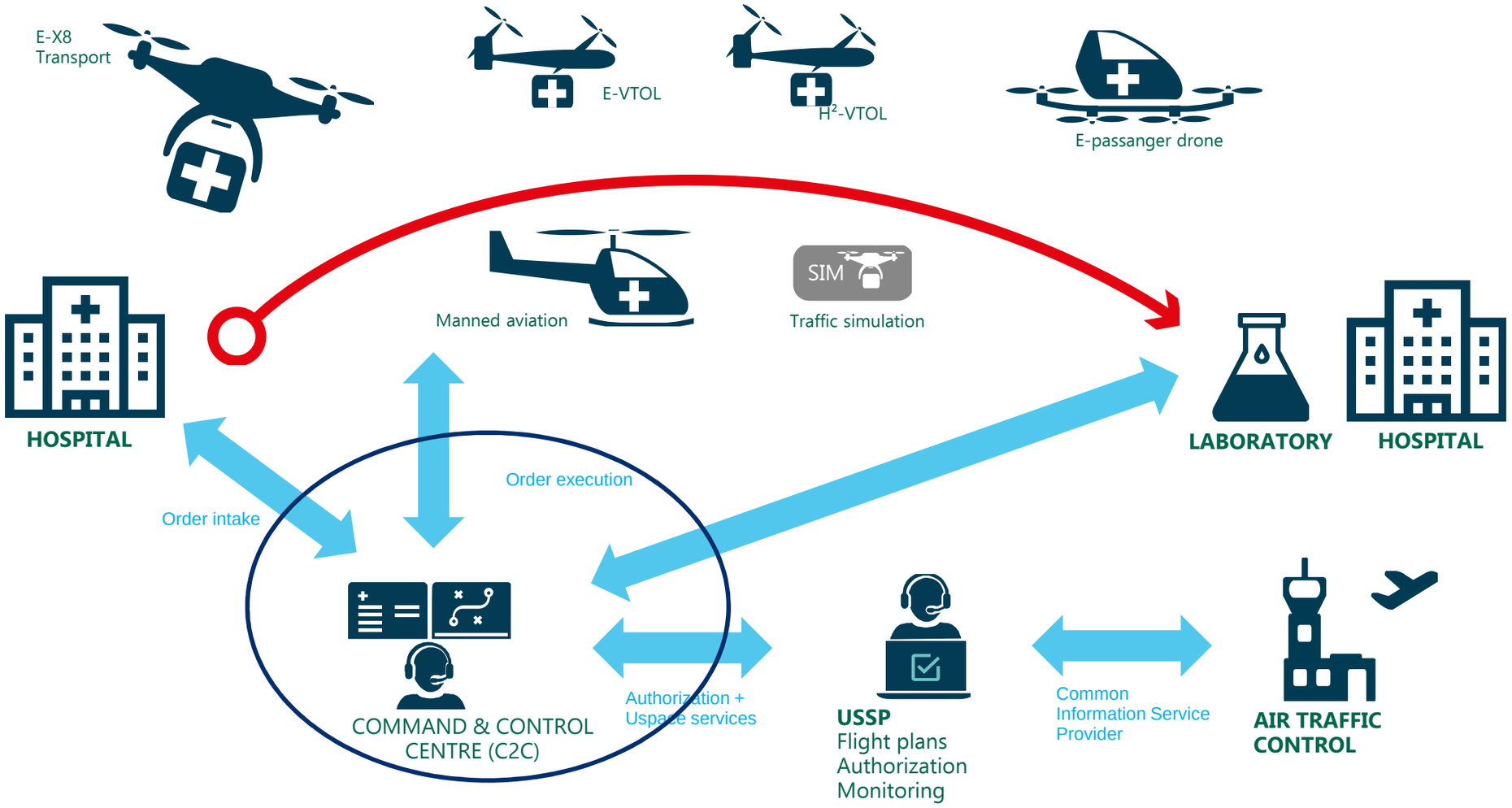
HELICUS

SAFIR-Med wins Digital European Sky Award

Winner: Helicus coordinated EU project receives validation and recognition from EU aviation actors through prestigious award.

09/03/2023

Case 2: Medical Transport via Drone



Case 3: Privacy Management Software

- Privacy management software suite with international ambitions
- Starting in 2019 with very limited resources
- Taking off with 2-3 domain expert, 1 software engineer in shared center
- Evolving to 3-5 domain experts, 4 software engineers in shared center
- First software engineers are now being insourced

RESPONSUM

Belgium-based RESPONSUM raises €1M from Volta Ventures, can be extended up to €2.5M

Case 3: Privacy Management Software

The all-in-one privacy management software solution

Simplify and automate your Privacy compliance challenges with an all-in-one Privacy Management software. Minimize risks for your organization and turn Privacy into a competitive advantage.

Get a free demo

Case 4: Referral Platform for Healthcare

- Referral platform for healthcare providers and primary care
- Starting in 2020 with extremely limited resources
- Taking off with 1 domain expert, 1 software engineer in shared center
- Currently continues in low expenditure mode

zipster


KONINKLIJKE ARTSENVERENIGING VAN ANTWERPEN

CERTIFICAAT

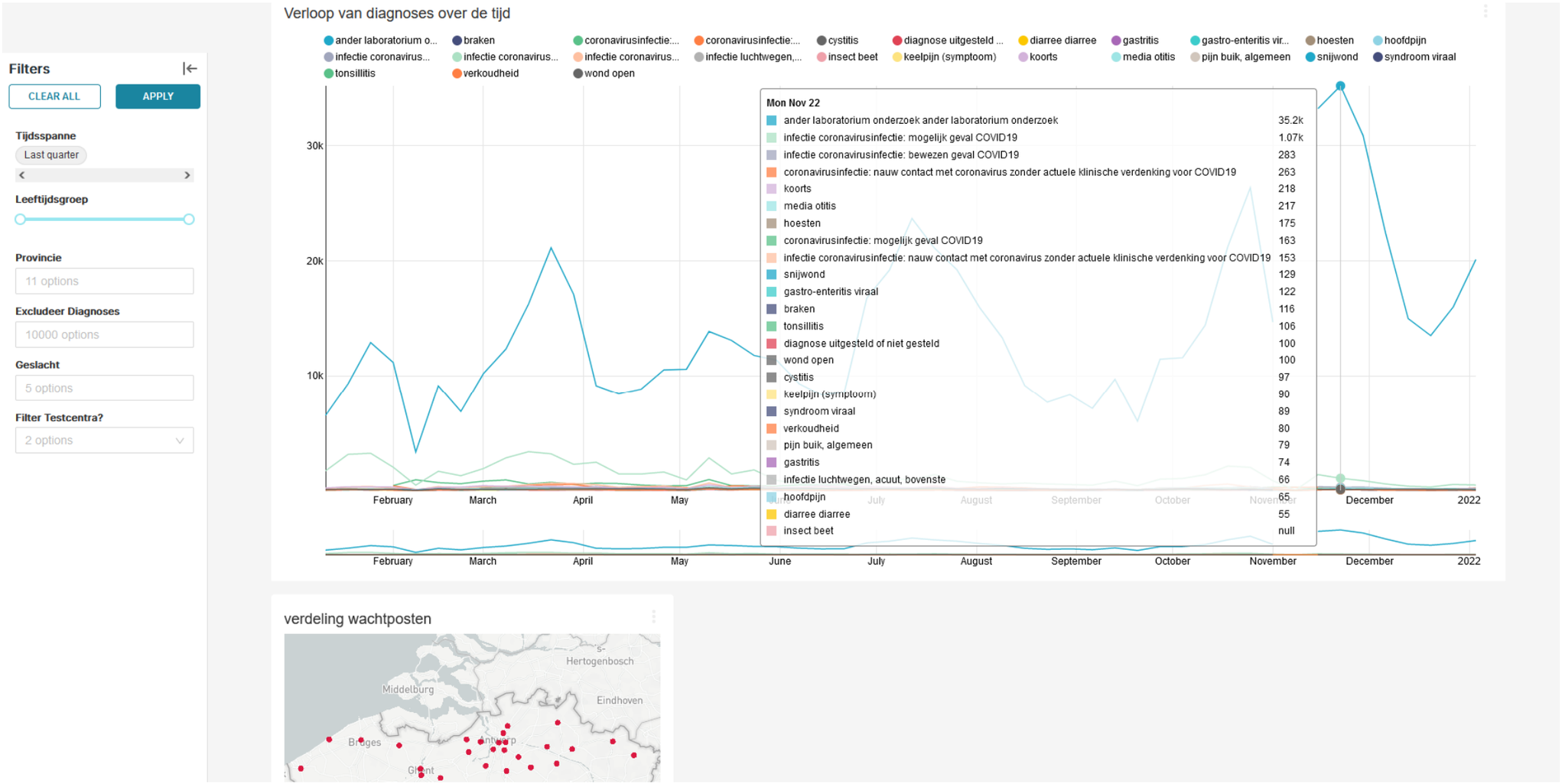
WINNAAR KARVA-PRIJS 2022

De KARVA-prijs bekroont een bijzondere bijdrage aan een optimale relatie en samenwerking tussen huisartsen, specialisten en andere gezondheidswerkers over de grenzen van vakgebieden heen.

ZIPSTER

uitgereikt te Antwerpen, 15 september 2022

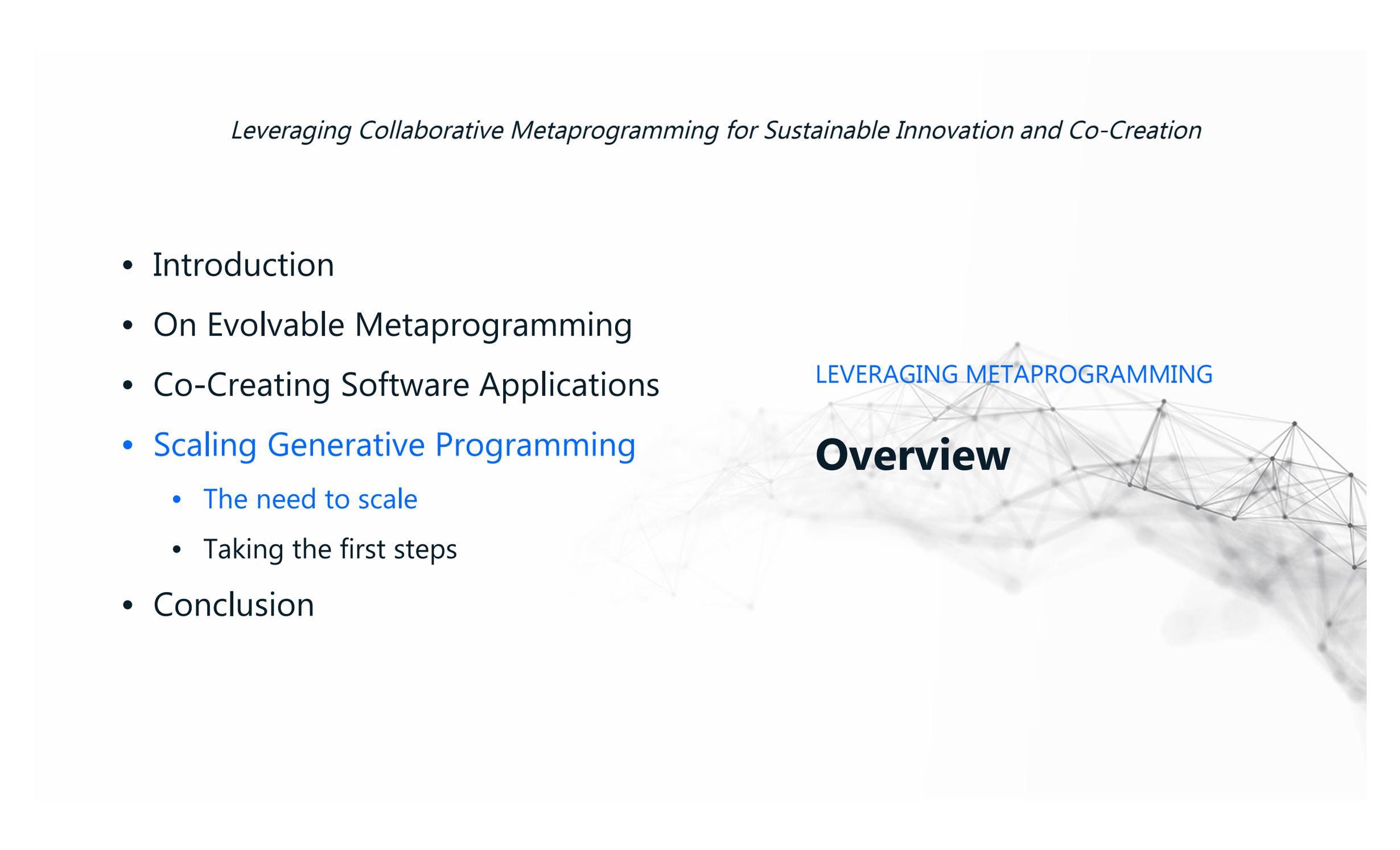
Case 4: Referral Platform for Healthcare



- Introduction
- On Evolvable Metaprogramming
- Co-Creating Software Applications
- Scaling Generative Programming
  - The need to scale
  - Taking the first steps
- Conclusion

LEVERAGING METAPROGRAMMING

Overview



Need to Scale Generative programming

- Automatically generating source code is as old as programming itself
  - Called *automatic programming*, *generative programming*, *metaprogramming*
- The issues that generative programming is supposed to address/solve are as relevant and acute as ever:
  - Software is growing in size and importance
  - Shortage of tens of thousands of programmers
  - Multi-trillion lines of code with billions of defects
  - Gigantic IT development and maintenance budgets
- For programming, interfaces have enabled *scalable collaboration*:
  - Within companies, across companies, in open source communities
  - Resulting in rich application offering and versatile hardware support

The Field of Generative Programming

- Better known through names/trends like:
  - Model-Driven Architecture (MDA)
  - Model-Driven Engineering (MDE)
  - Model-Driven Software Development (MDSD)
  - Low-Code Development Platforms (LCDP)
  - No-Code Development Platforms (NCDP)
- The various trends share the *use of models to structure requirements* and/or *to represent domain knowledge*:
- The field is still evolving and facing challenges and criticisms:
  - Suitability for large-scale and mission-critical enterprise systems
  - Lack of intermediate representation, pervasive concepts for DSL reuse
  - Either a *conceptual gap* toward code, *or tied to* a technological solution

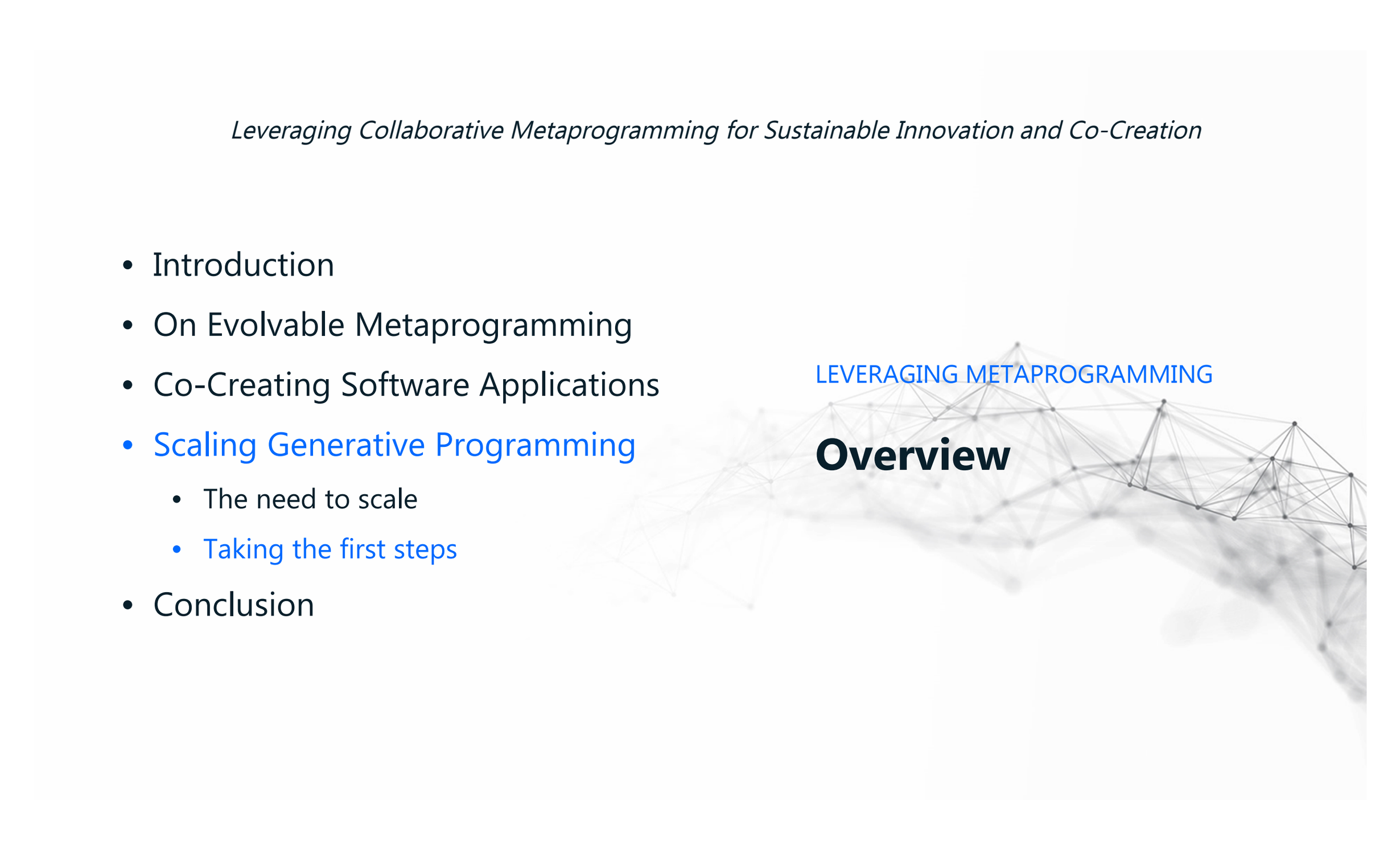
Two-Sided Interfaces for Metaprogramming

- Generative programming performs a *transformation*
  - From domain and/or intermediate models
  - To code generators and programming code
- Need to define open *interfaces at both ends*
  - To add or extend domain models
  - To add or replace code generators
- A meta-circular architecture as proposed
  - Simplifies the definition of the interfaces
  - Allows for a horizontal integration architecture
  - Avoids the *non-scalable burden* on the meta-code
    - To integrate, or at least accommodate, ever more extensions at both ends

- Introduction
- On Evolvable Metaprogramming
- Co-Creating Software Applications
- Scaling Generative Programming
  - The need to scale
  - Taking the first steps
- Conclusion

LEVERAGING METAPROGRAMMING

Overview



Exchanging Code Generators with Partners

- Using the *standard web application meta-model*, generators for
  - Implementations of layers in other frameworks
    - e.g., Spring Boot, Angular
  - Additional application functionality
    - e.g., advanced search, improved data security
- Using *additional dedicated meta-models* with generators for
  - Message connectors
  - Specific types of screens
  - Event handling
- Using completely different meta-models
  - Documents, cloud deployment, simulations

Exchanging Code Generators with Partners

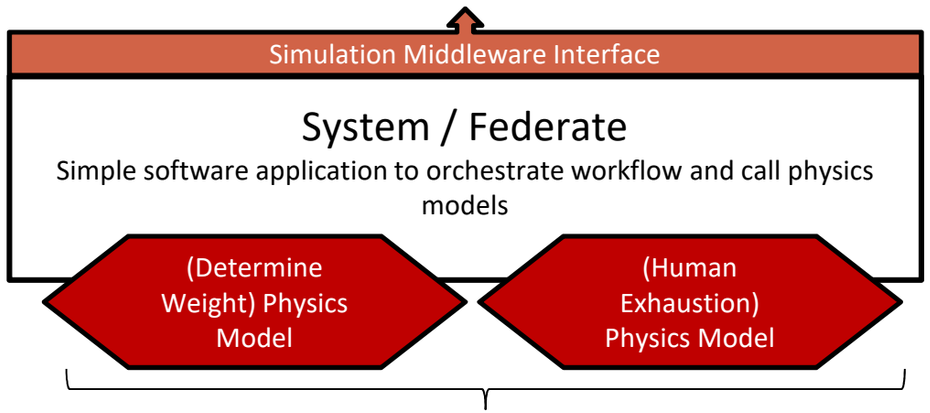
- Meta-models are based on a common basis of ERD / UML / OWL:
  - Entity classes or data entities
  - Attributes or datatype properties
  - References or object properties
  - *Instances* of these entities or classes
- After defining meta-models, tooling:
  - Generates the meta-circular stack for these entities, including:
    - classes representing model instances , XML readers and writers
    - view and control classes for create and manipulate models in a user interface
  - Provides native support
    - to enter models based on these meta-models
    - to invoke expanders defined for these models

Exchanging Code Generators with Partners

- Code generators are packaged as *Expansion Resources*, containing
  - Sets of individual *expanders*, each generating 1 artifact
  - Possibly a meta-model definition and/or transmuters
  - Possibly some run-time libraries
- Expansion resources can be made available to all partners
  - Willingness is abundant, technicalities challenging
- Developed expansion resources include
  - Some major bundles from Dutch Tax Office
    - Message connectors, view models
  - Several smaller bundles from partners like Cast4All, Responsum
    - Advanced search, data security

Integrating Another Metaprogramming System

- Integration with models from the High-Level Architecture for distributed simulations to combine federated simulations



Code generation template

- Software best practices via Normalized Systems Theories
- Connection
- Timing
- Coordination across models
- Database connection
- Threading
- etc.

- Introduction
- On Evolvable Metaprogramming
- Co-Creating Software Applications
- Scaling Generative Programming
- Conclusion

LEVERAGING METAPROGRAMMING

Overview

Conclusion

- We have argued that the ability to *collaborate and evolve is crucial for software to realize leverage effects* in a scalable and sustainable way
- We have argued that the application of metaprogramming, and even *meta-circular metaprogramming is crucial* to realize these abilities
- We have explained how these abilities can *enable the productive and sustainable co-creation of software* applications with several cases
- We have indicated that this *co-creation at the meta-level can even be more powerful*, and have explained our first elementary steps
- We invite everyone who is interested in making and exchanging modules for code generation, i.e., expansion resources, to join us

Some References

- Mannaert Herwig, McGroarty Chris, Gallant Scott, De Cock Koen, **Integrating Two Metaprogramming Environments : An Explorative Case Study :** ICSEA 2020 - ISSN 2308-4235 - IARIA, 2020, p. 166-172
- Mannaert Herwig, De Cock Koen, Uhnak Peter, **On the realization of meta-circular code generation : the case of the normalized systems expanders,** ICSEA 2019 - ISSN 2308-4235 - IARIA, 2019, p. 171-176
- De Bruyn Peter, Mannaert Herwig, Verelst Jan, Huysmans Philip, **Enabling normalized systems in practice : exploring a modeling approach**, Business & information systems engineering - ISSN 1867-0202 - 60:1(2018), p. 55-67.
- Mannaert Herwig, Verelst Jan, De Bruyn Peter, **Normalized systems theory : from foundations for evolvable software toward a general theory for evolvable design**, ISBN 978-90-77160-09-1 - Koppa, 2016, 507 p.
- Mannaert Herwig, Verelst Jan, Ven Kris, **Towards evolvable software architectures based on systems theoretic stability**, Software practice and experience - ISSN 0038-0644 - 42:1(2012), p. 89-116
- Mannaert Herwig, Verelst Jan, Ven Kris, **The transformation of requirements into software primitives : studying evolvability based on systems theoretic stability**, Science of computer programming - ISSN 0167-6423 - 76:12(2011), p. 1210-1222
- Mannaert Herwig , De Bruyn Peter, Verelst Jan, **On the Interconnection of Cross-Cutting Concerns within Hierarchical Architectures**, IEEE Transactions on Engineering Management - ISSN 1558-0040 - 69:6(2022), p. 3276-3291.
- **Normalized Systems Foundation Lectures** : https://www.youtube.com/c/normalizedsystems
- **Normalized Systems Documentation and Tooling :** https://foundation.stars-end.net


QUESTIONS ?
herwig.mannaert@uantwerp.be