Architectures: Design Patterns for Component-Based Systems

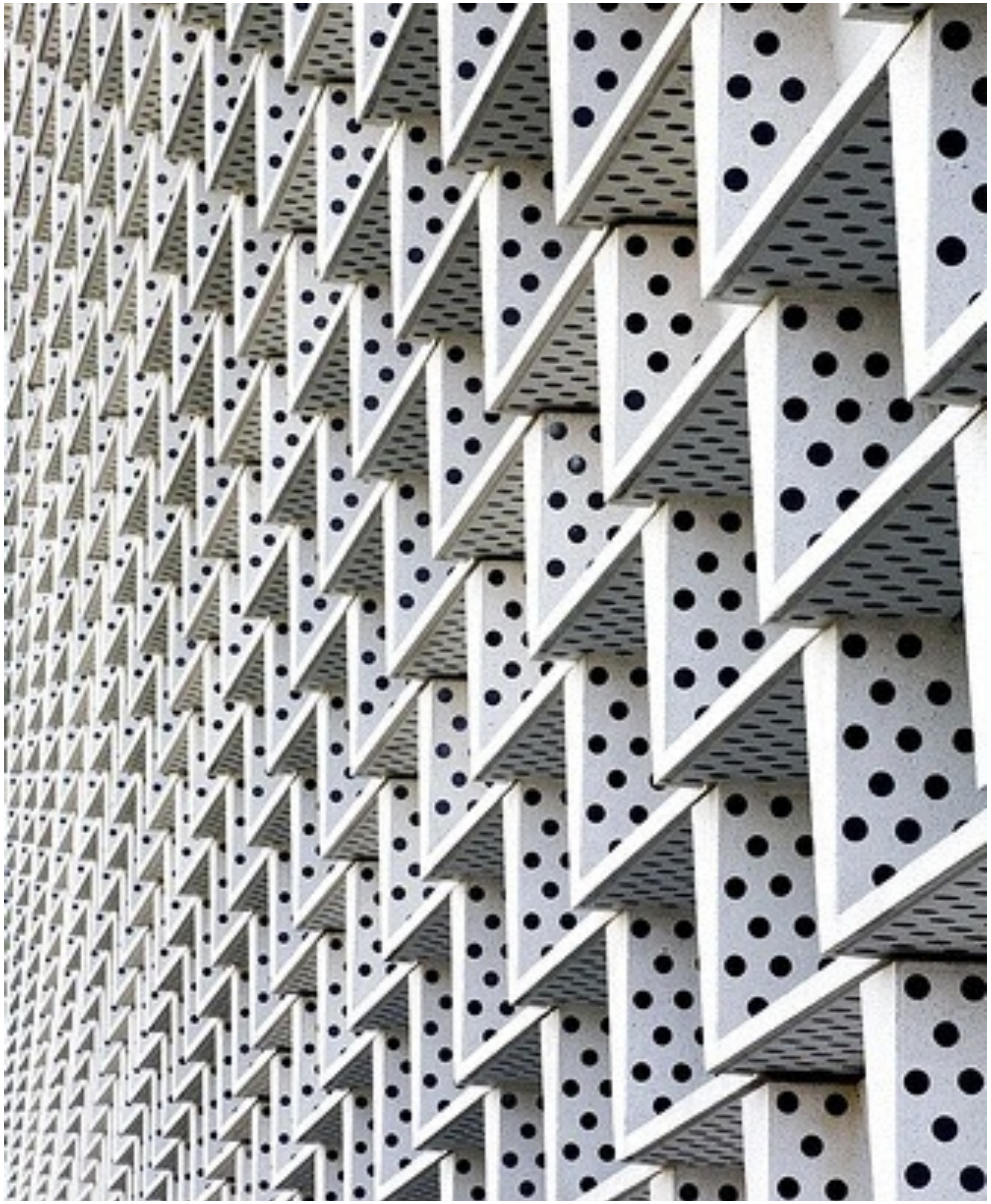
Short talk @ TGC, 5th of September, 2014

Simon Bliudze
École polytechnique fédérale de Lausanne

Joint work with: Paul Attie, Eduard Baranov, Marius Bozga, Mohamad Jaber and Joseph Sifakis

Reusable design patterns

- Systems are not built from scratch
- Maximal re-use of building blocks
  (off-the-shelf components)
- Maximal re-use of solutions (libraries, design patterns, etc.)
- Express coordination constraints in declarative manner



Applications

- Concurrency:
  (a)synchronous, time-triggered, token-ring,
  mutual exclusion

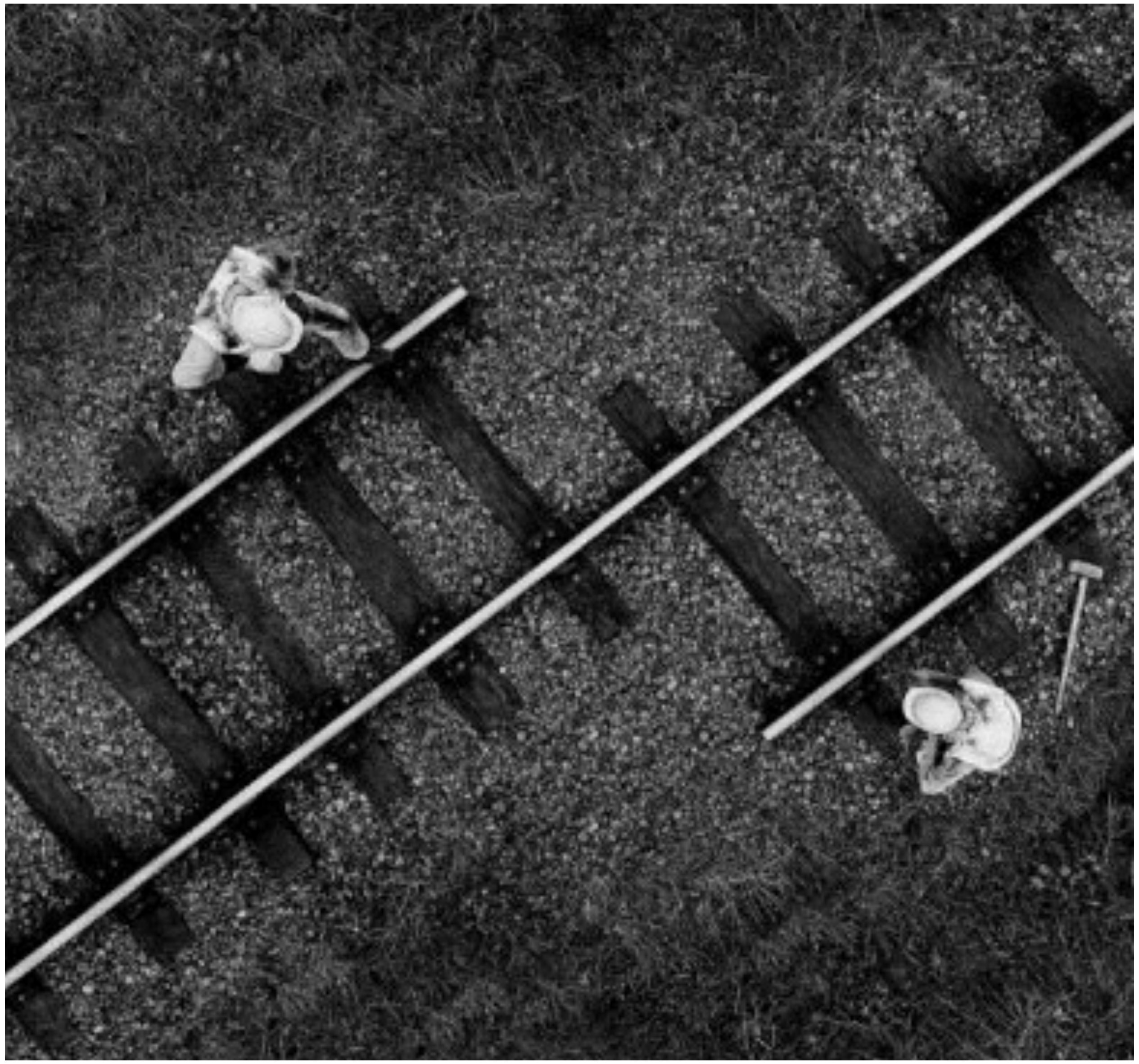
- Protocols:
  communication protocols, data access control,
  encryption, authentication

- Robustness:
  fault detection & recovery, resource management

- etc.

Theory of architectures

- How to model?
- How to specify?
- How to combine?
- How to implement efficiently?

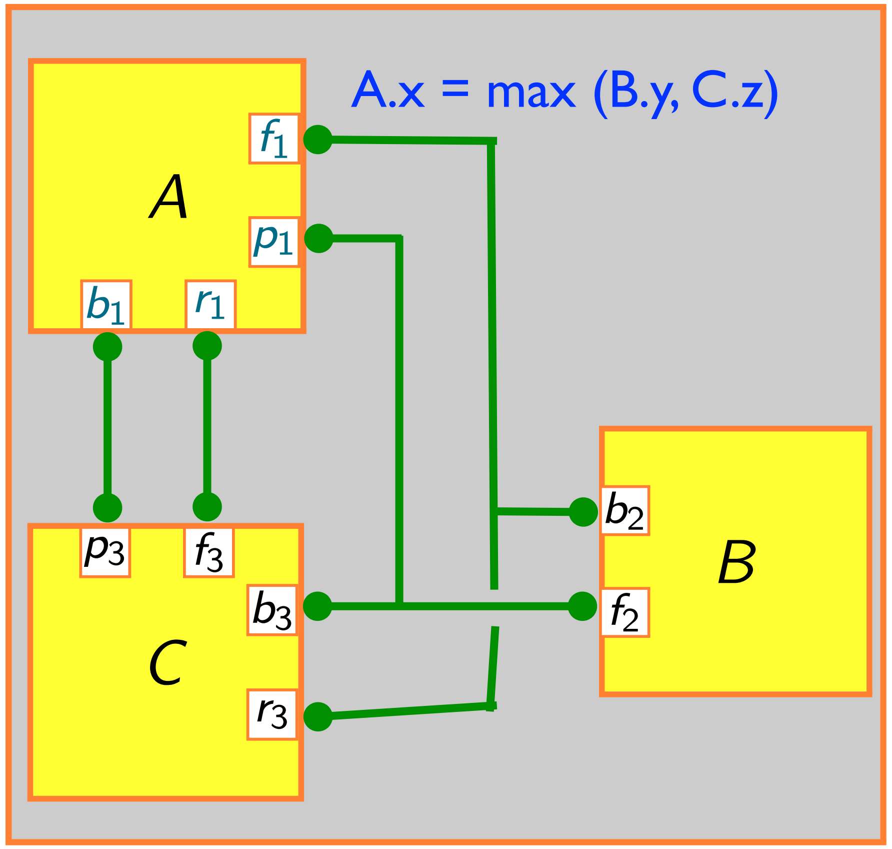


Architectures enforce characteristic properties. The crucial question is whether these are preserved by composition?

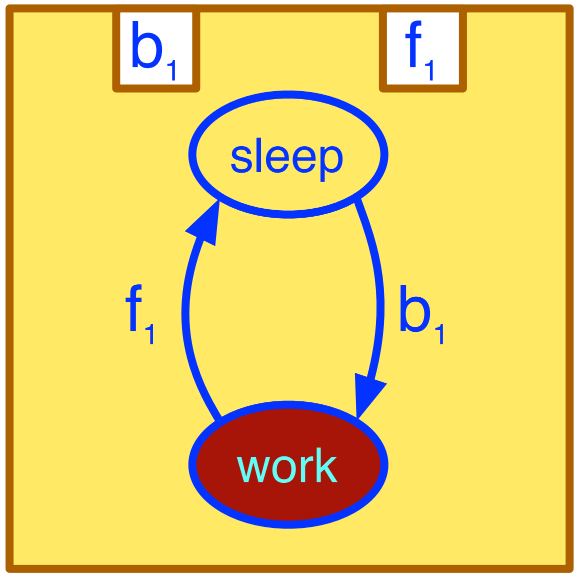
How to model?

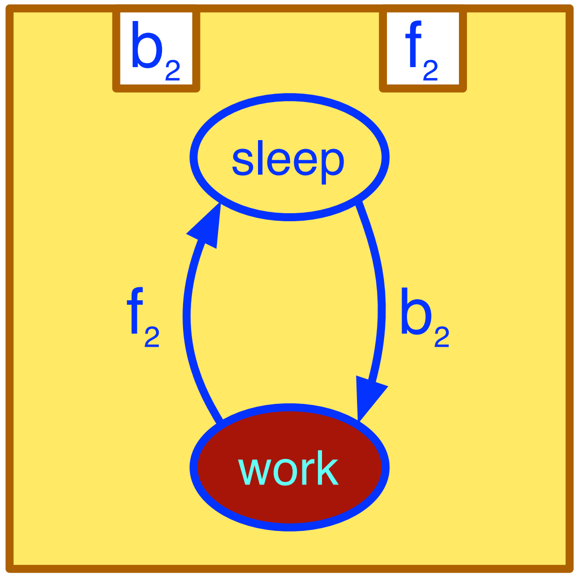
Component-based design

- Three layers
    - Component behaviour
    - Coordination
    - Data transfer

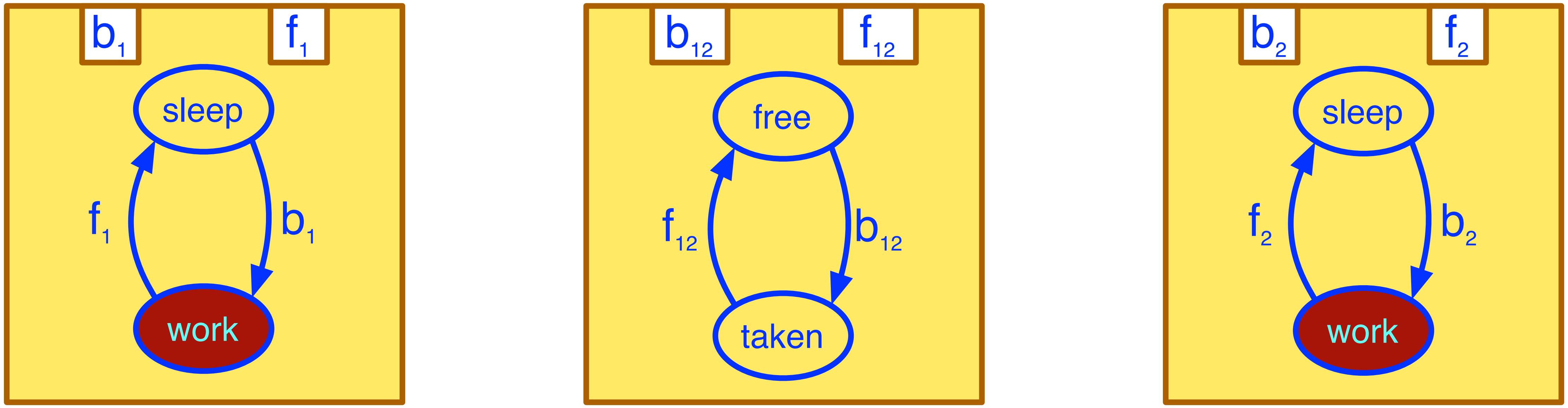


Example in BIP

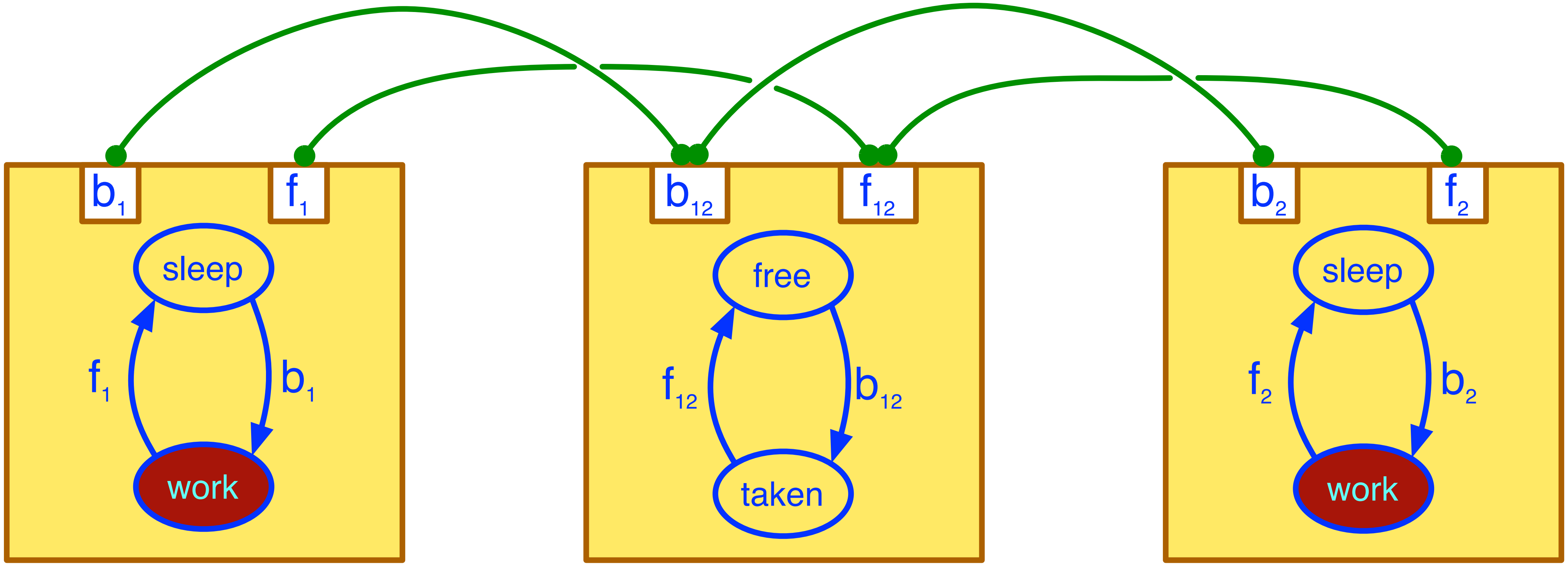




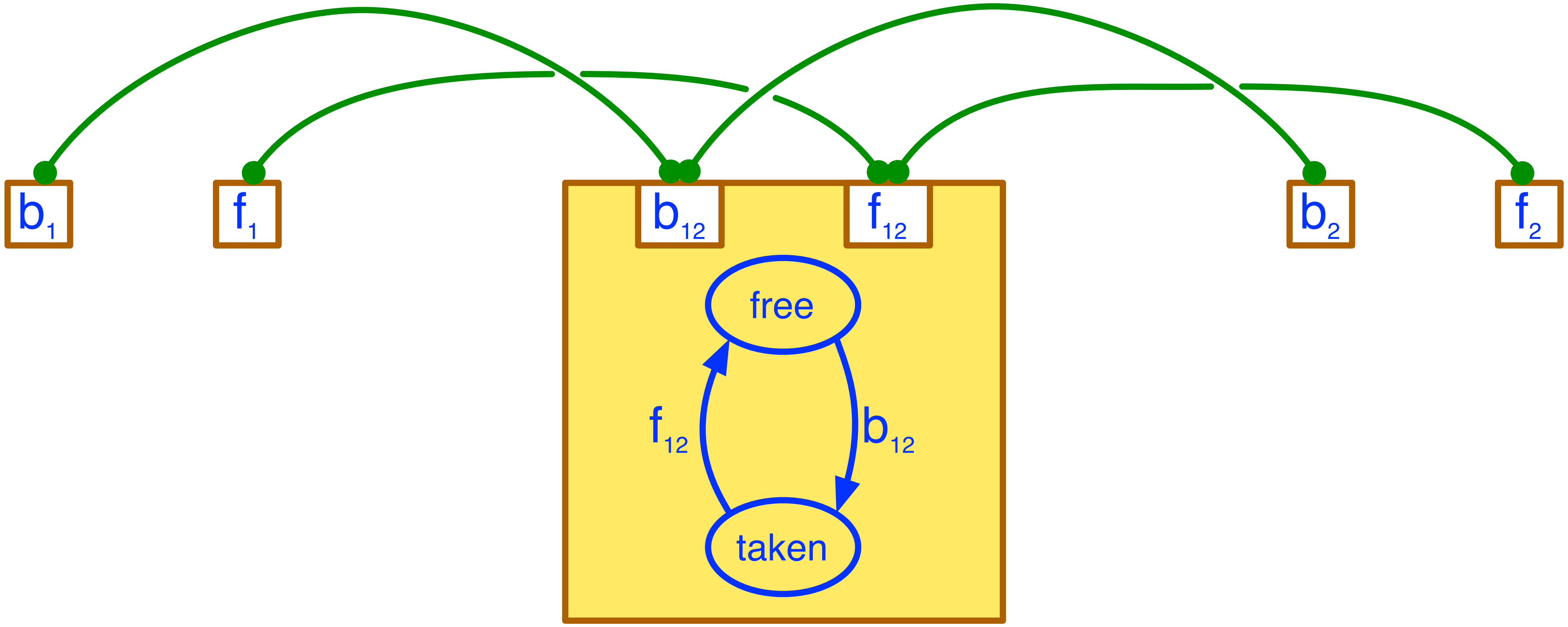
Example in BIP



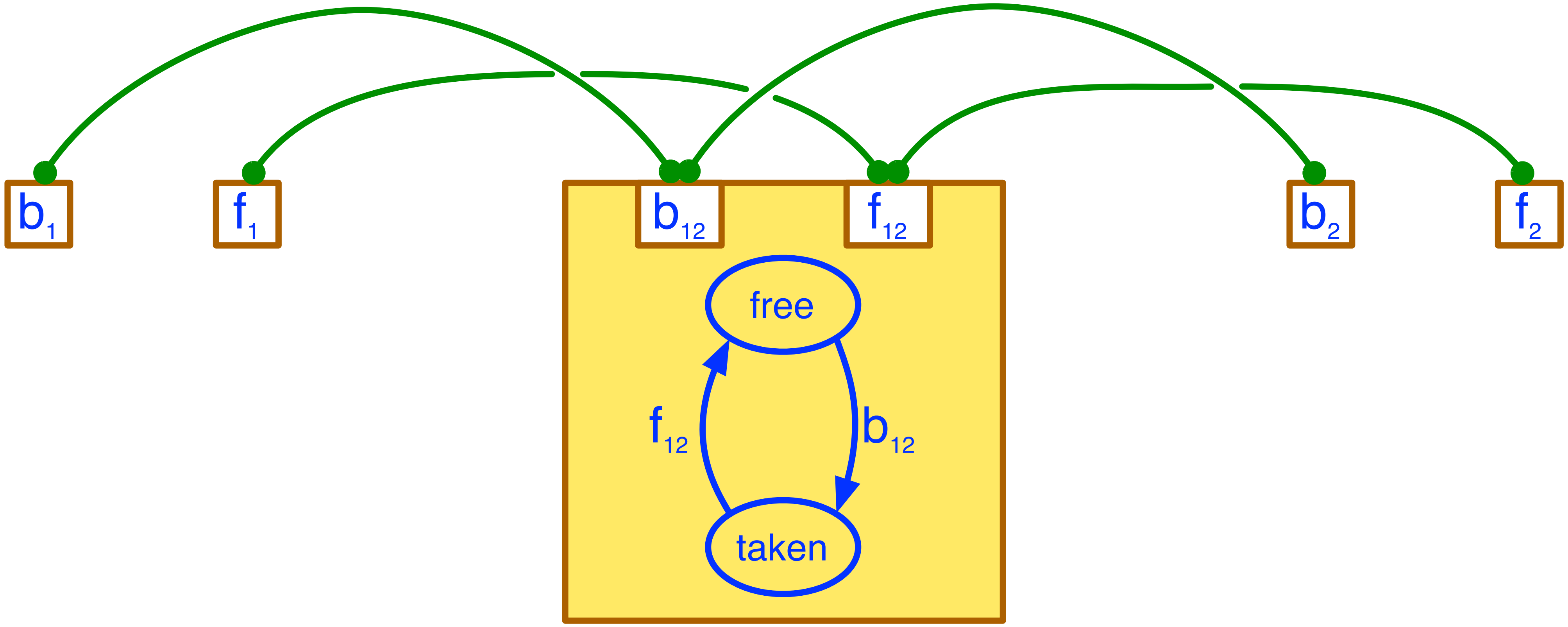
Example in BIP



Example in BIP



Example in BIP



$$\gamma_{12} = \{\emptyset,\ b_1 b_{12},\ b_2 b_{12},\ f_1 f_{12},\ f_2 f_{12}\}$$

Architectures in BIP

$$A = (\mathcal{C}, P_A, \gamma)$$

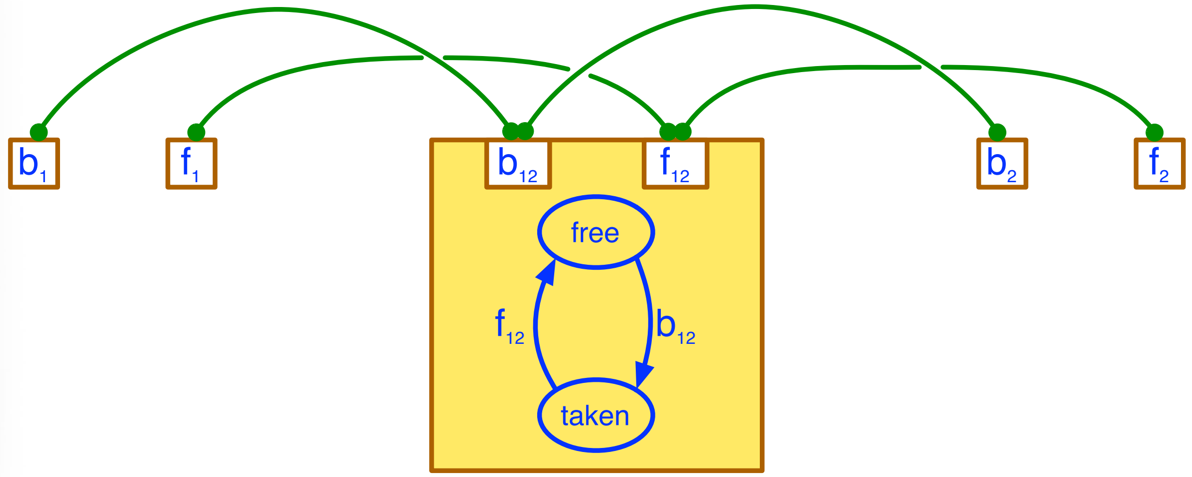
Set of coordinating behaviours

Interface (ports)

Interaction model

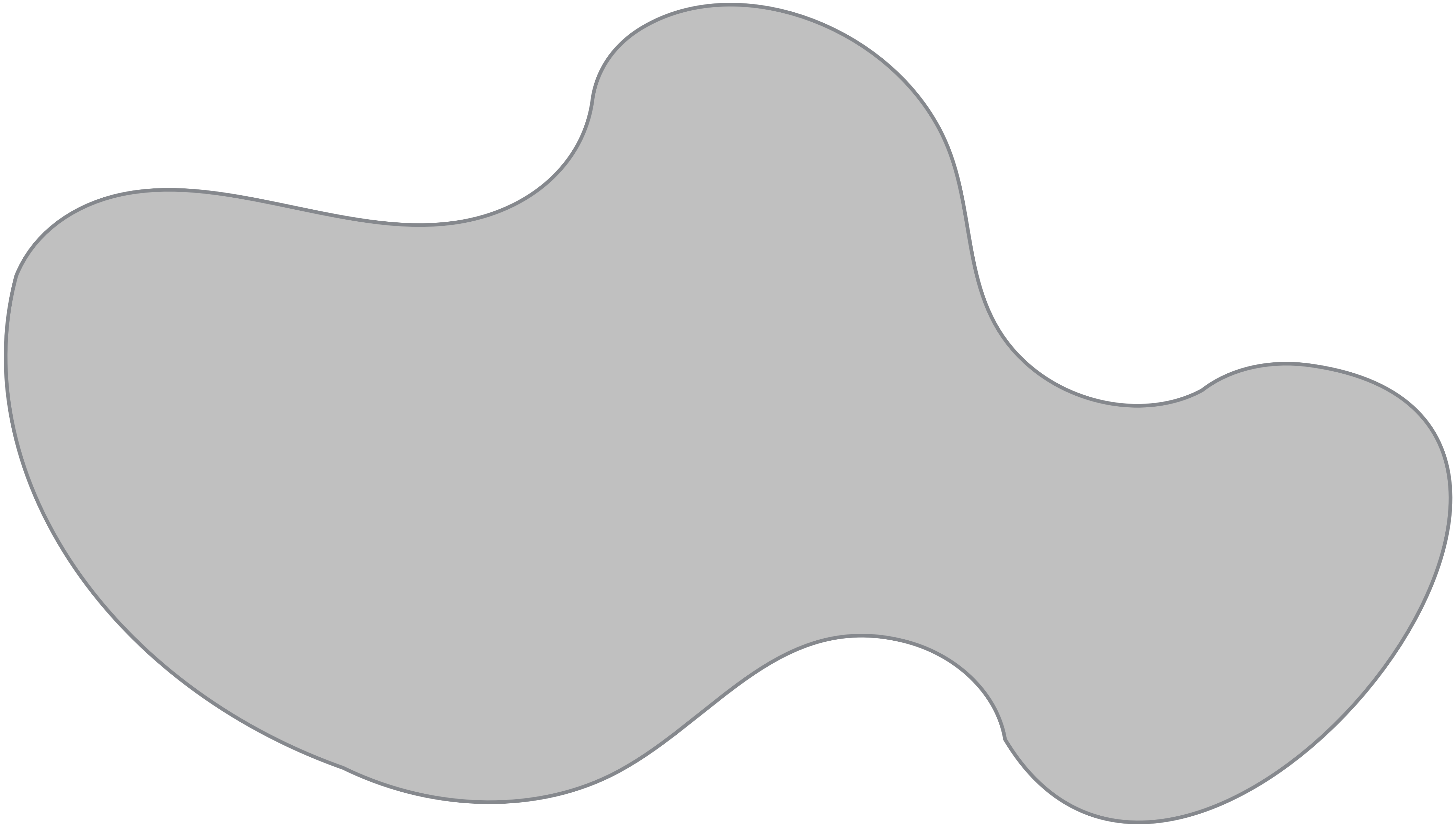
The interface includes all ports of the coordinator components

$$\bigcup_{C \in \mathcal{C}} P_C \subseteq P_A$$

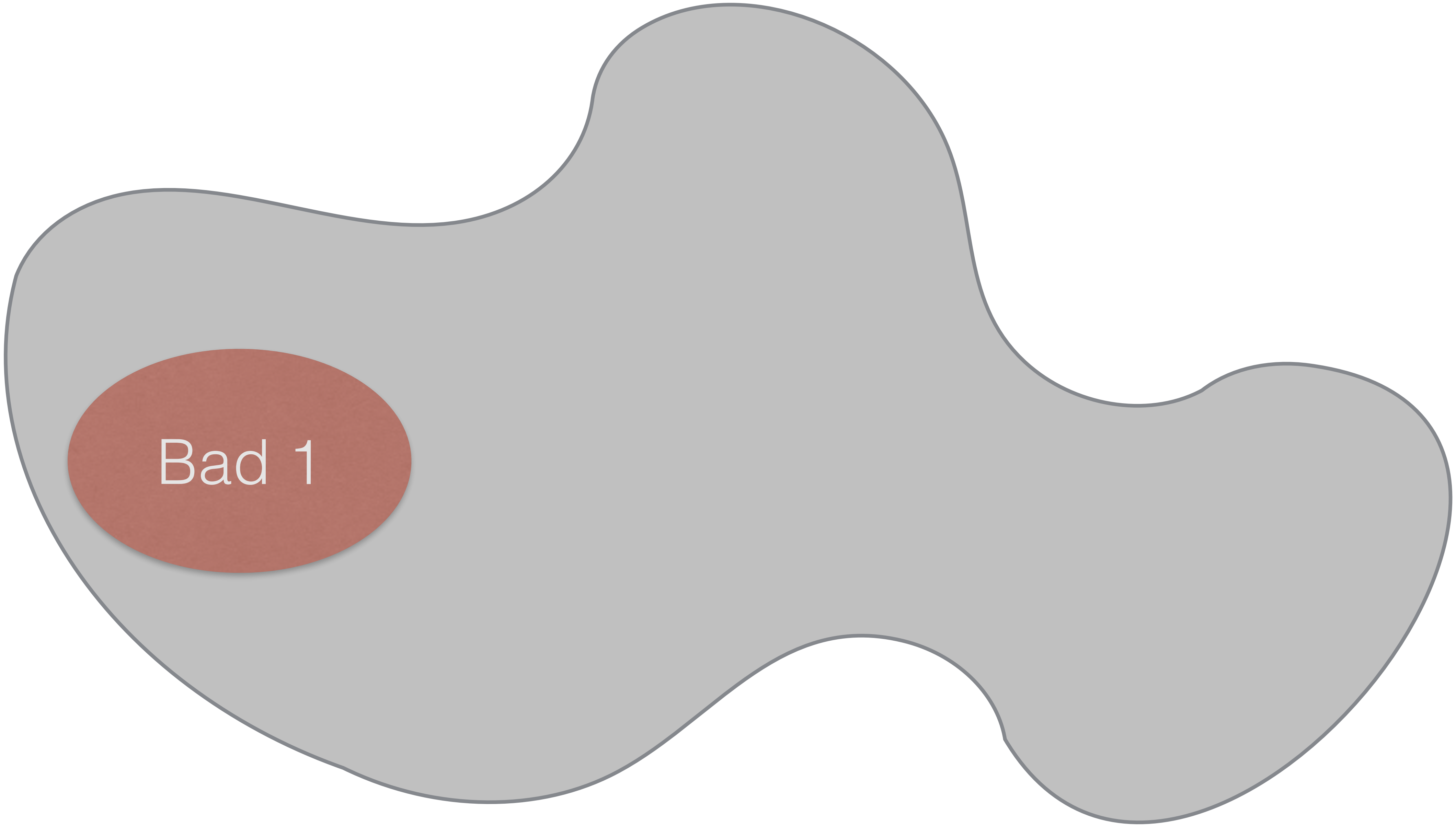


How to combine?

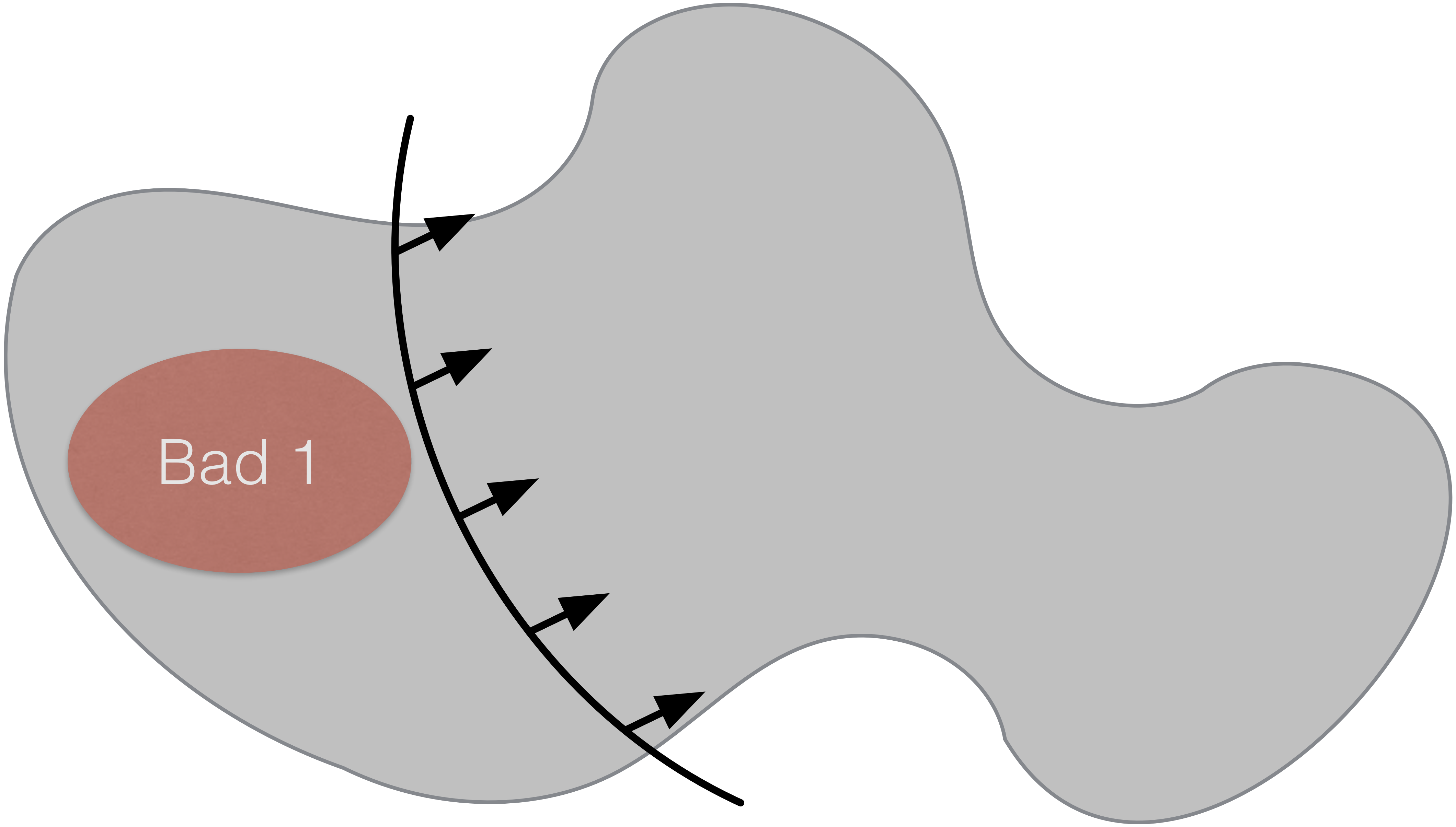
Constraints intuition



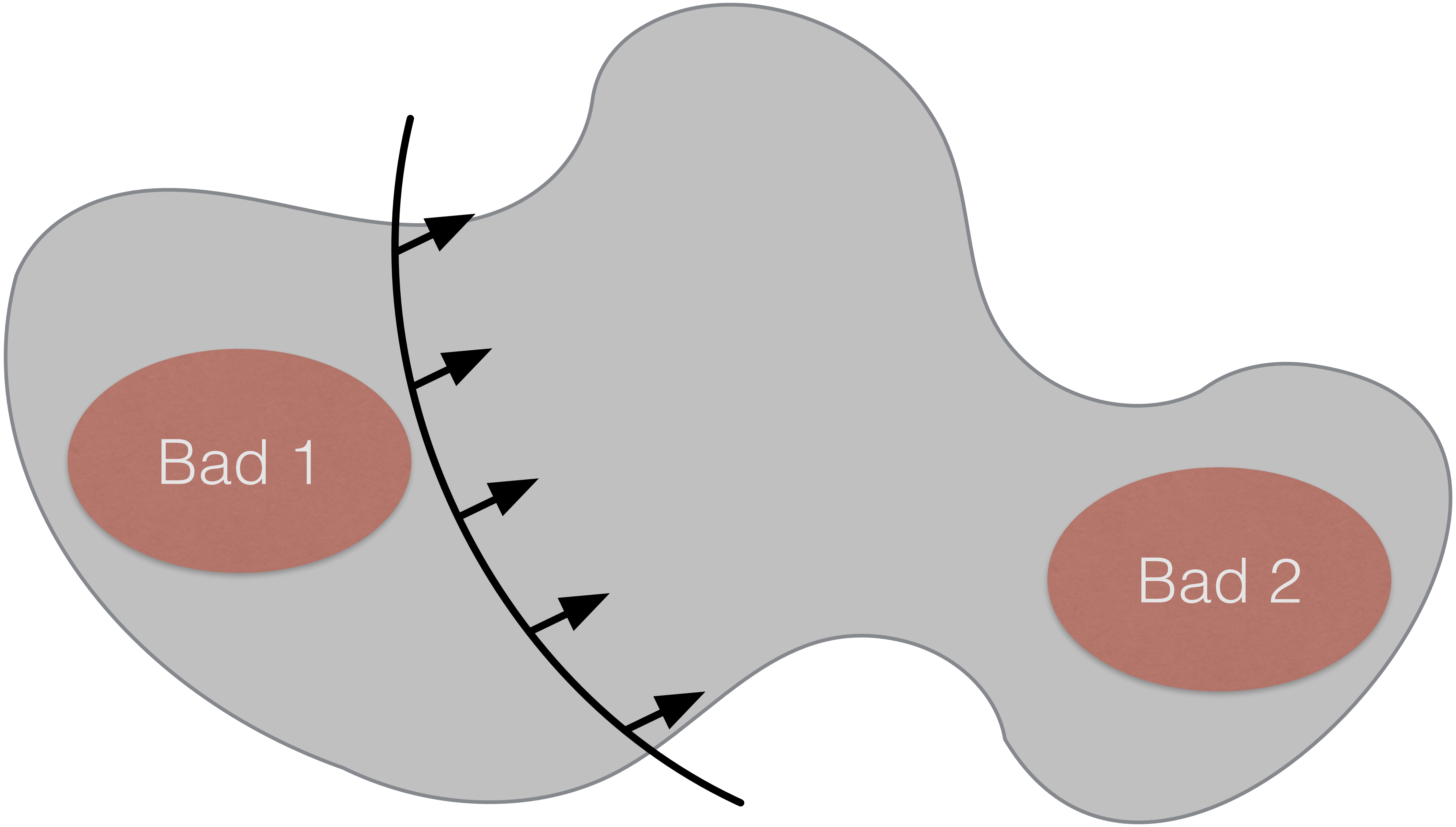
Constraints intuition



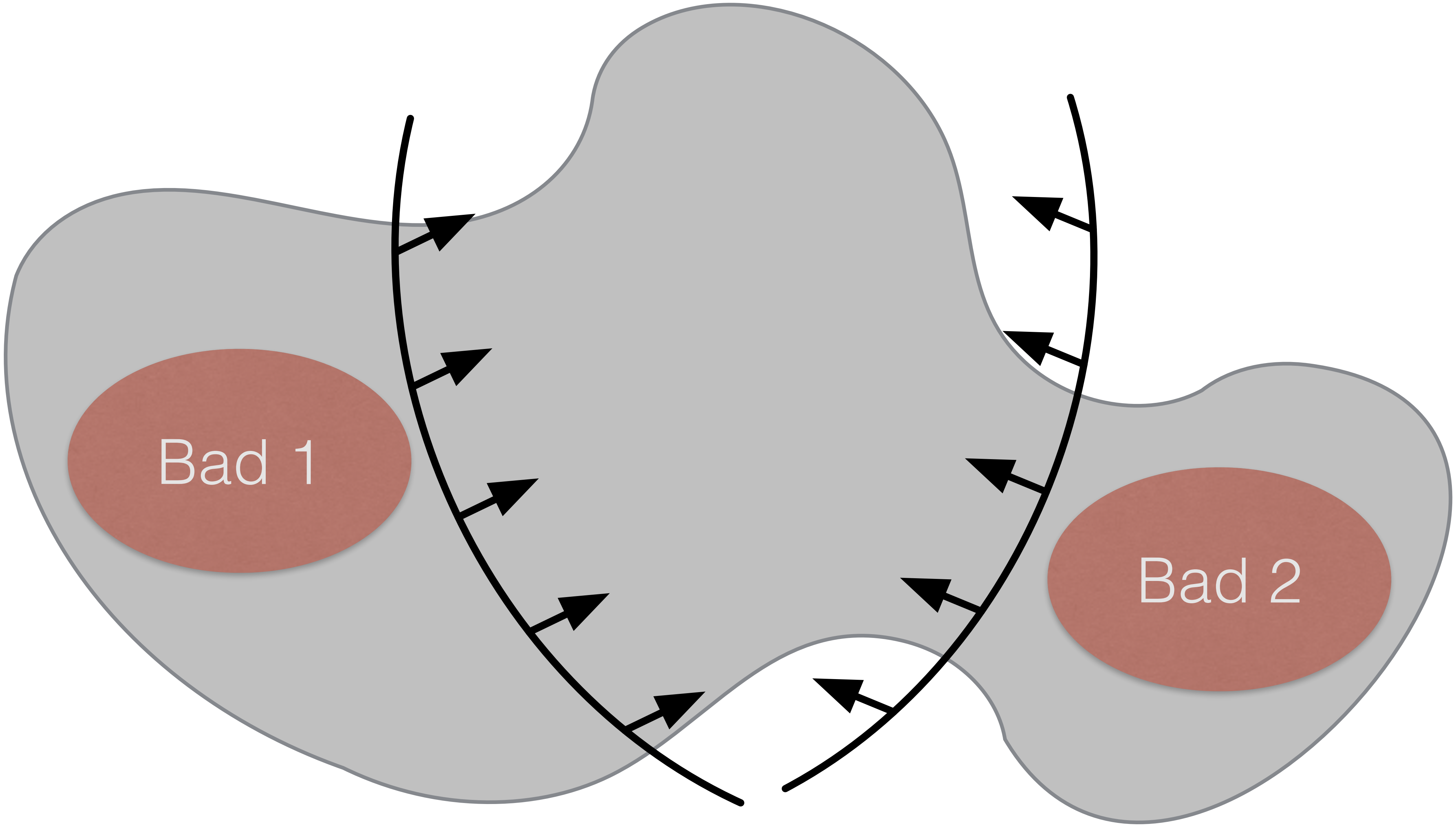
Constraints intuition



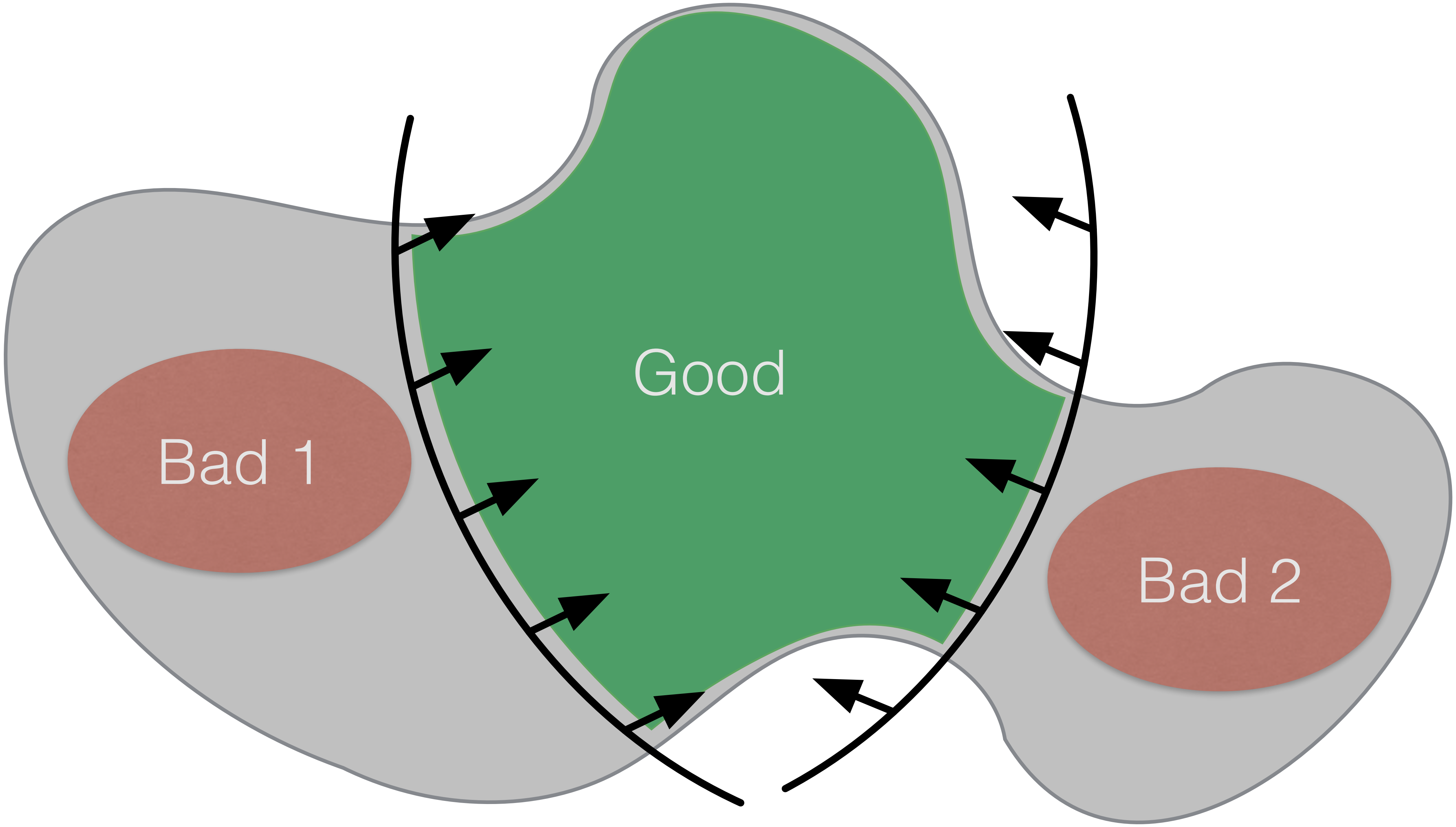
Constraints intuition



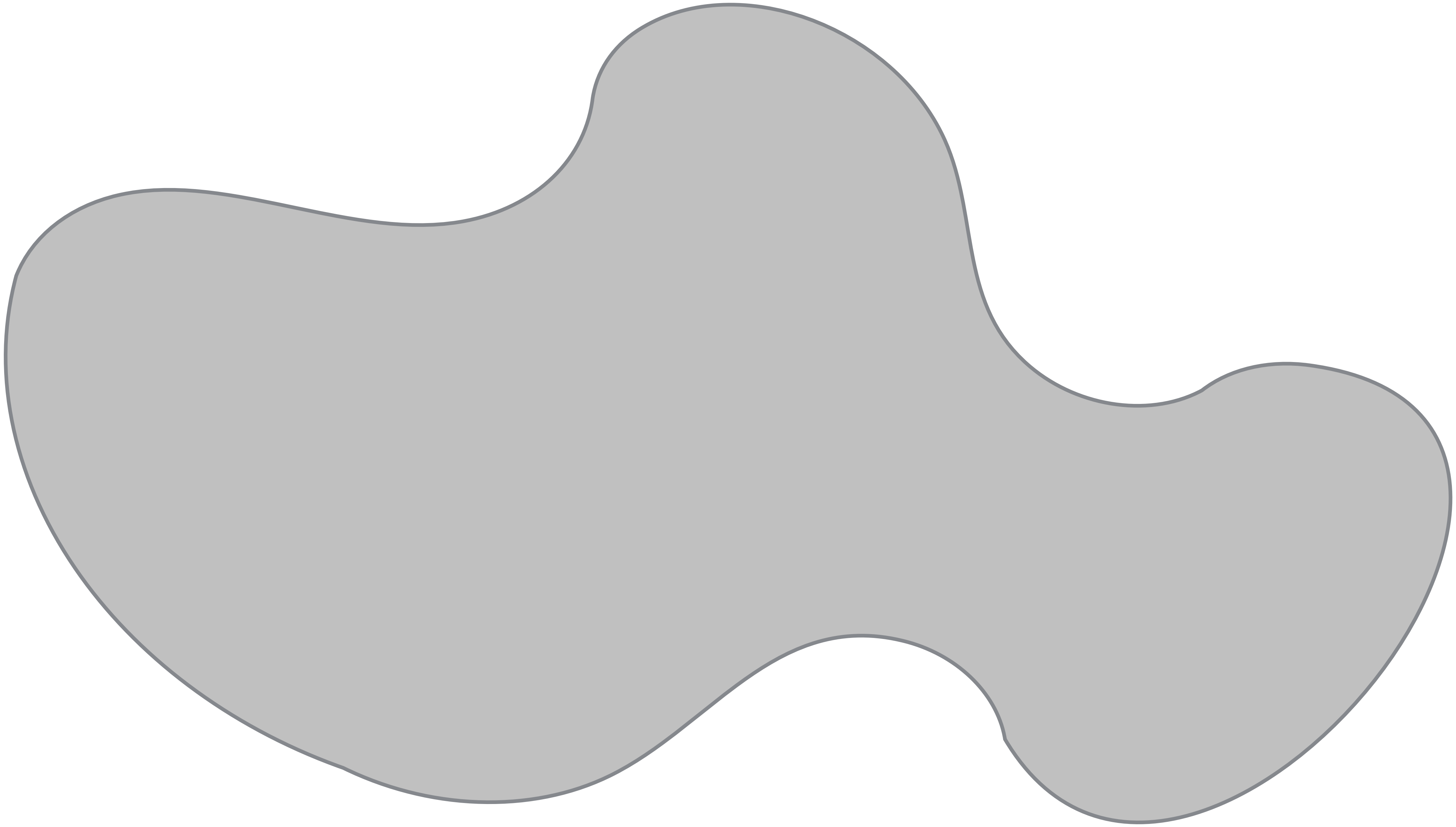
Constraints intuition



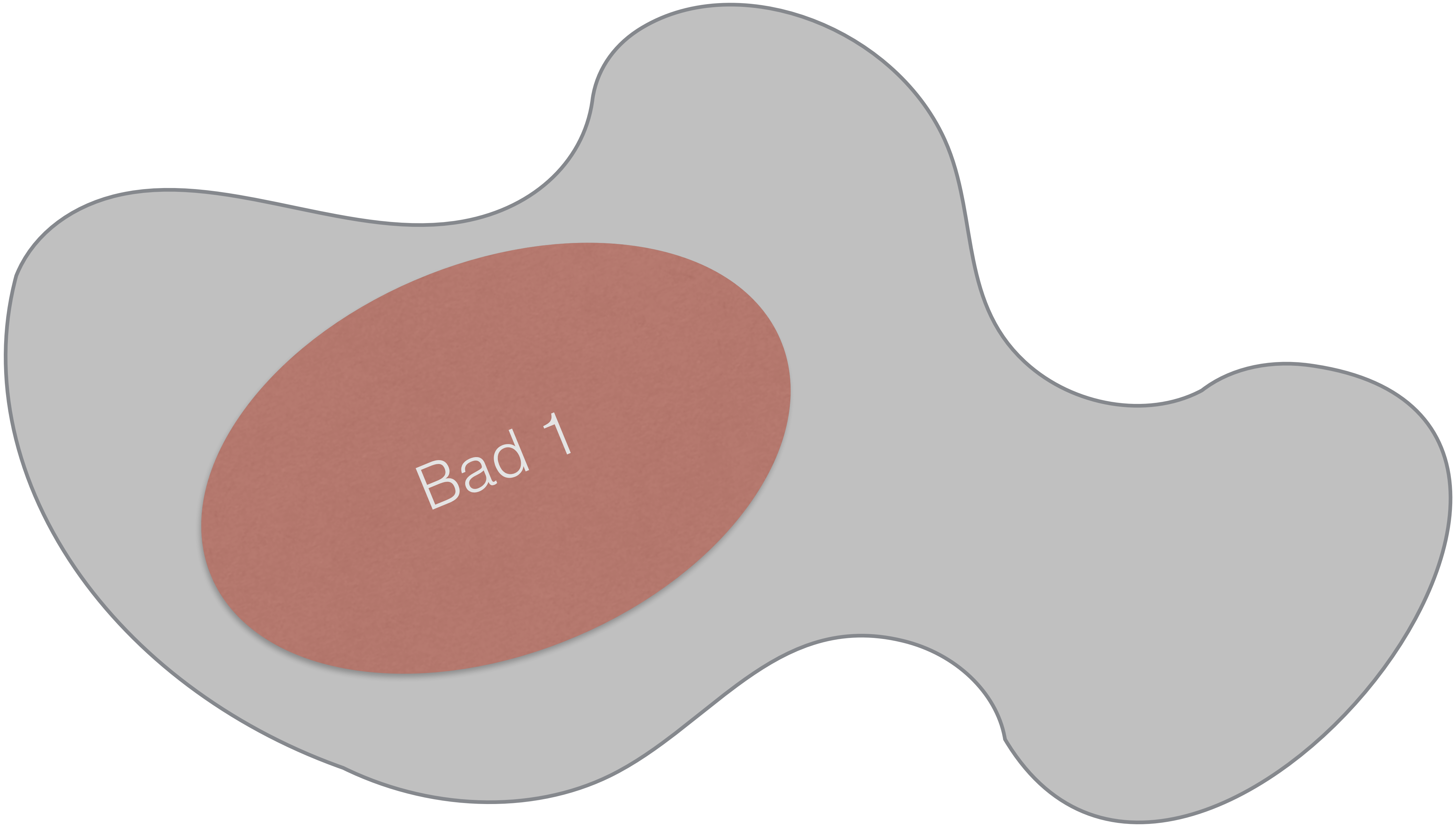
Constraints intuition



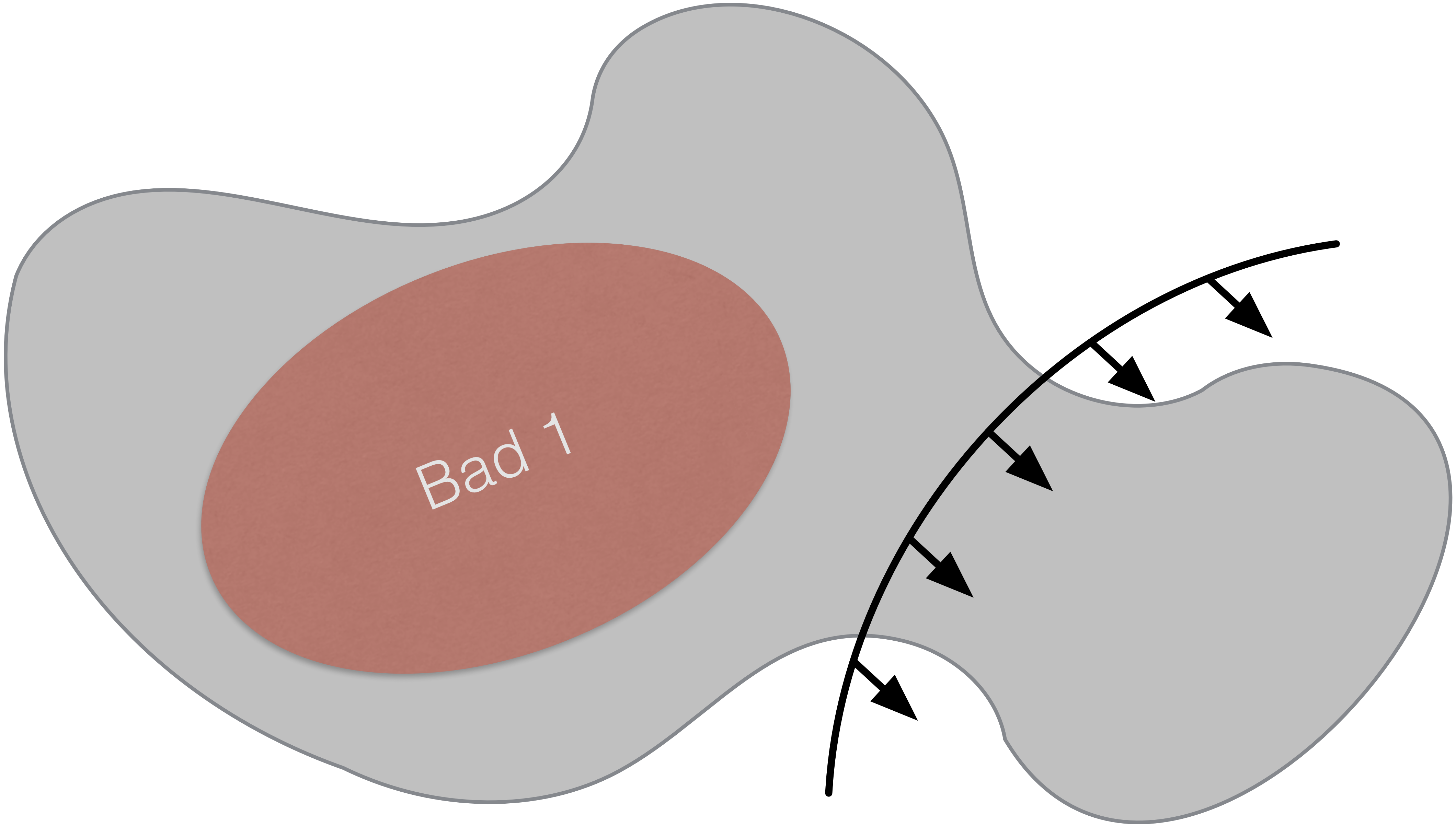
Limits of white magic



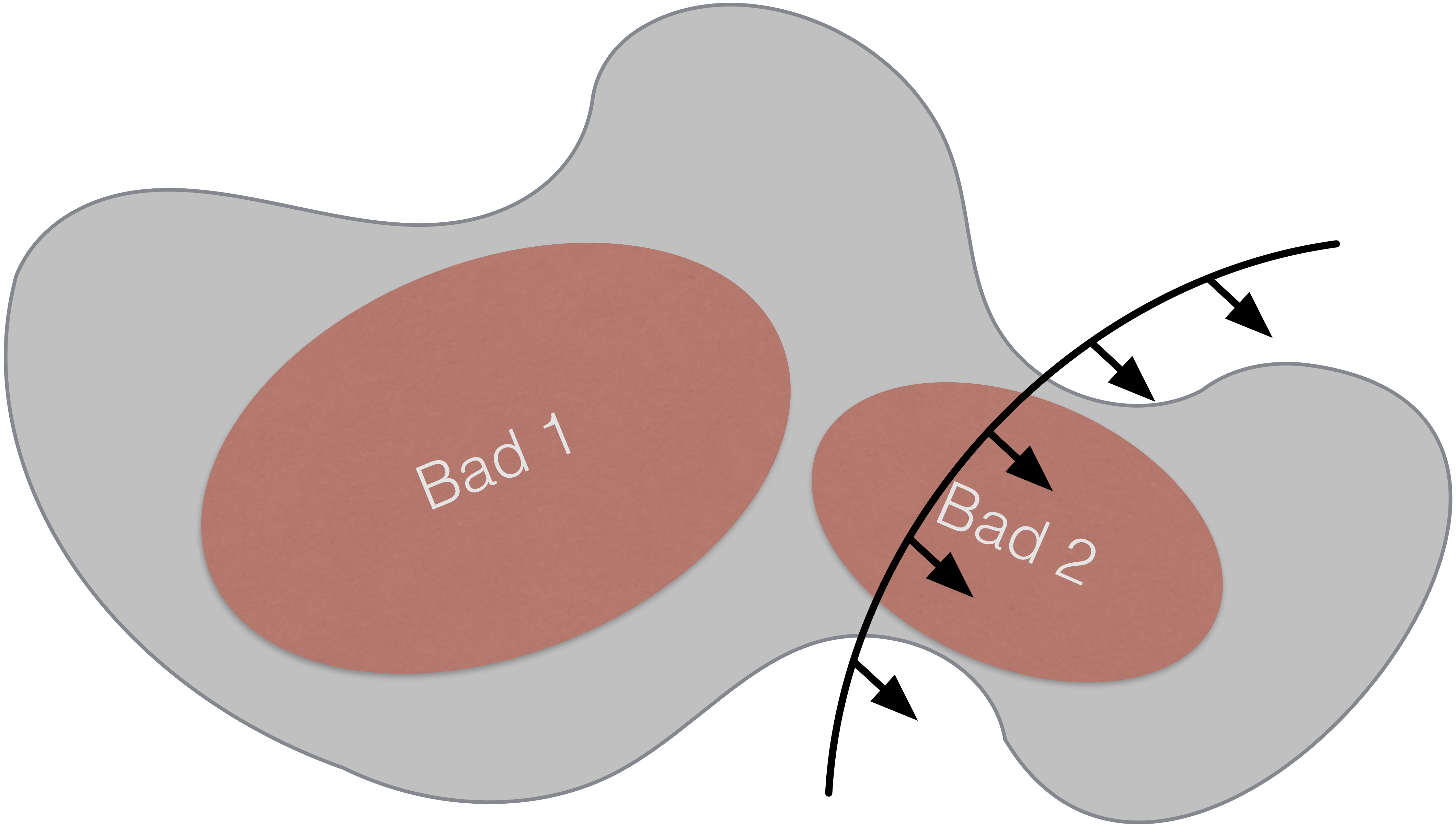
Limits of white magic



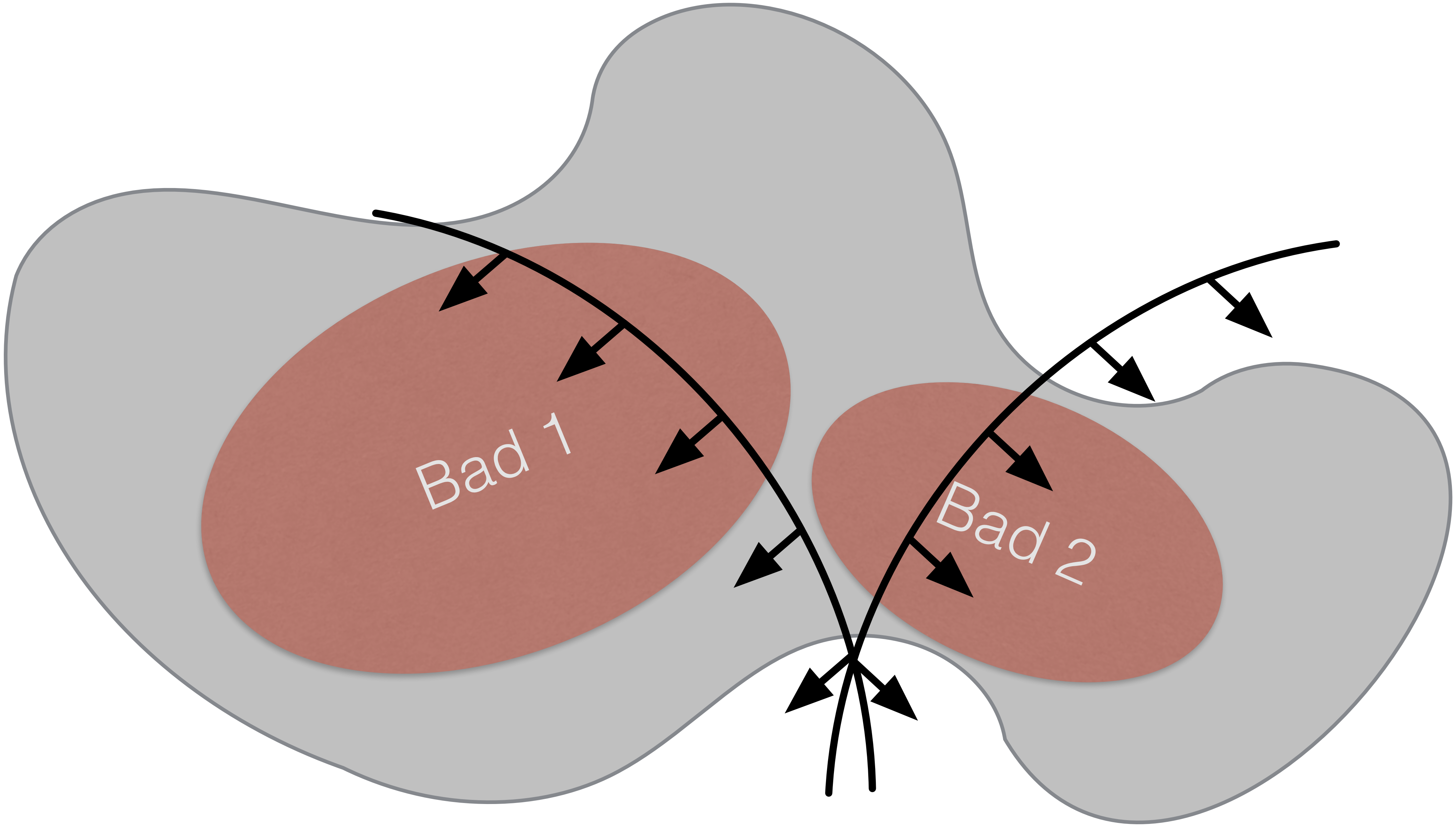
Limits of white magic



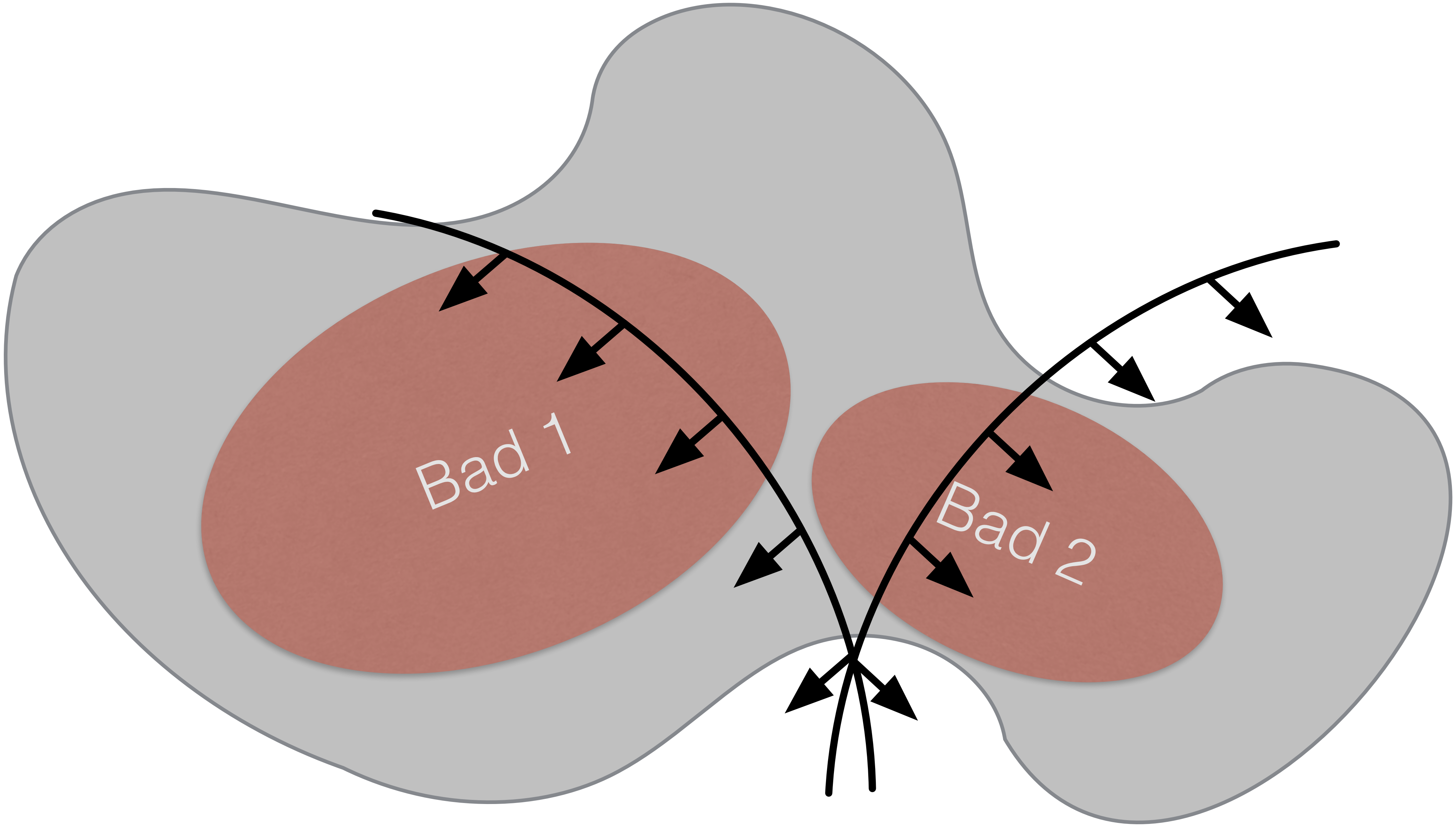
Limits of white magic



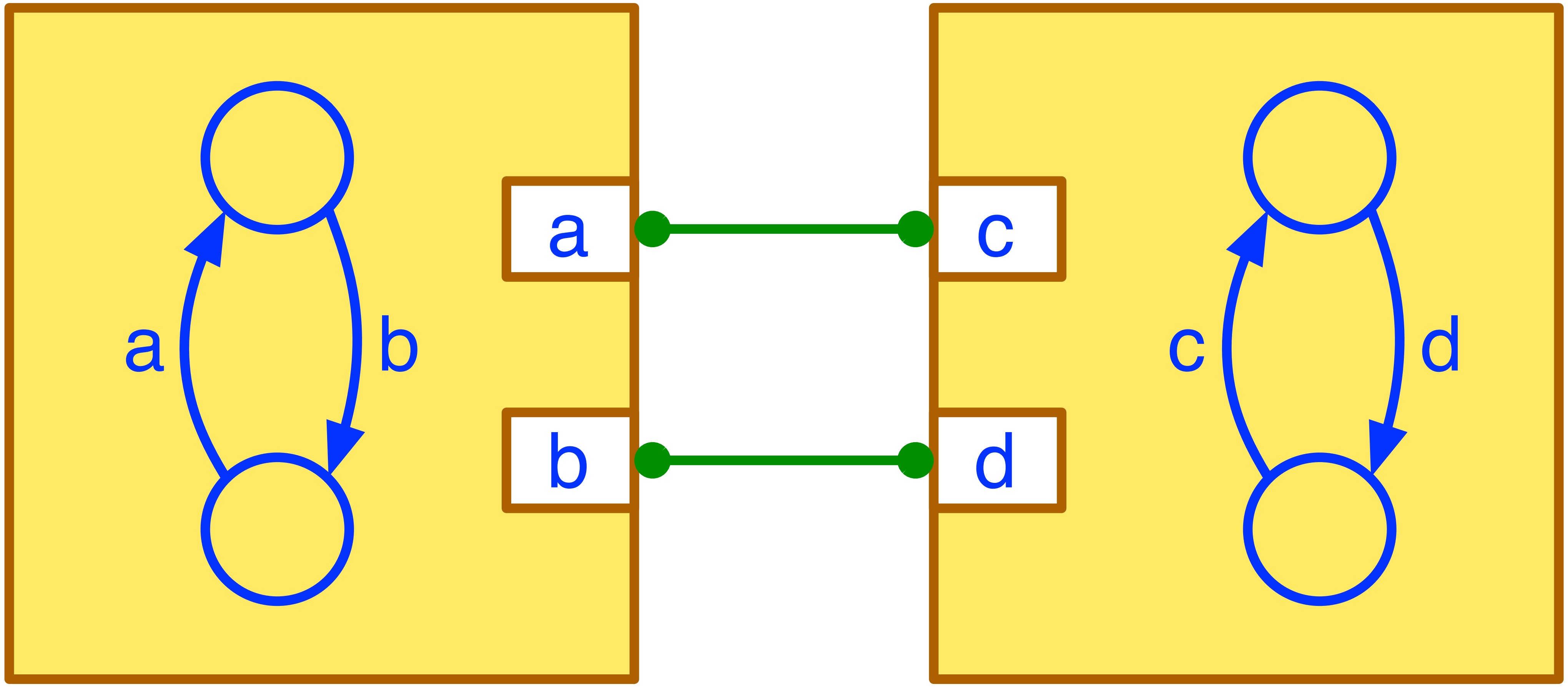
Limits of white magic



Limits of white magic



How to implement efficiently?



Architecture-based

External: component behaviour is not modified

Task 1:

```
while (true) {
   free(S1);
   take(S2);
   do-a;
   free(S1);
   take(S2);
   do-b;
}
```

Task 2:

```
while (true) {
   take(S1);
   free(S2);
   do-c;
   take(S1);
   free(S2);
   do-d;
}
```

Architecture-agnostic

Internal: the coordination primitives mix-up with the functional behaviour of the components

Two sides of the same design

- Architecture-based
  - External
  - Declarative
  - Highly abstract
  - Relies on an execution engine
  - Easy to understand, analyse and manipulate

- Architecture-agnostic
  - Internal
  - Imperative
  - Detailed
  - Relies on low-level primitives
  - Efficient

Two sides of the same design

- Architecture-based
  - External
  - Declarative
  - Highly abstrac
  - Relies on an execution engine
  - Easy to understand, analyse and manipulate

- Architecture-agnostic
  - Internal
  - I ive
  -
  - R s on low-level primitives
  - Efficient

Internalisation

Conclusion

- Architectures solve coordination problems by enforcing characteristic properties.

- First steps toward the study of a rigorous concept of architecture and its effective use for achieving correctness by construction in a system design flow.
  - A notion of architecture in BIP
  - Safety-preserving architecture composition operator
    - [SEFM 2014] *A General Framework for Architecture Composability*
  - Architecture internalisation using Top/Bottom component model
    - [CBSE 2014] *Architecture internalisation in BIP*

Future work

- Modelling
    - Study and classification of real-life architectures in various domains (Embedded systems, web services, enterprise integration, etc.)
    - Versatility of the model
    - Dynamic architectures
- Specification
    - Development of a simple powerful language for specifying architectures and their characteristic properties
- Efficiency & distribution
    - Optimisation of internalised architectures
    - Implementation based on message-passing mechanisms (e.g. AKKA)

Thank you for your attention