Co-Design of Autonomous Systems: From Hardware Selection to Control Synthesis

European Control Conference 2021

G. Zardini, A. Censi, and E. Frazzoli

Institute for Dynamic Systems and Control (IDSC)
ETH Zürich

gzardini@ethz.ch - http://gioele.science

The pain of engineering complex systems

An autonomous robot = actuation, hardware, sensing, computation, energetics, software, localization, control, perception, communication, behavior, interaction, mapping, planning, learning, regulations, coordination, social acceptance, liability

So many **components** (hardware, software, ...), so many choices to make!

Nobody can understand the **whole** thing!

We forget why we made some **choices**, and we are afraid to make **changes**...

These "computer" thingies are not helping us that much for design...

anthropomorphization of 21st century engineering malaise →

"My dear, it's simple: you lack a proper theory of co-design!"

Co-design of autonomous systems: from hardware selection to control synthesis

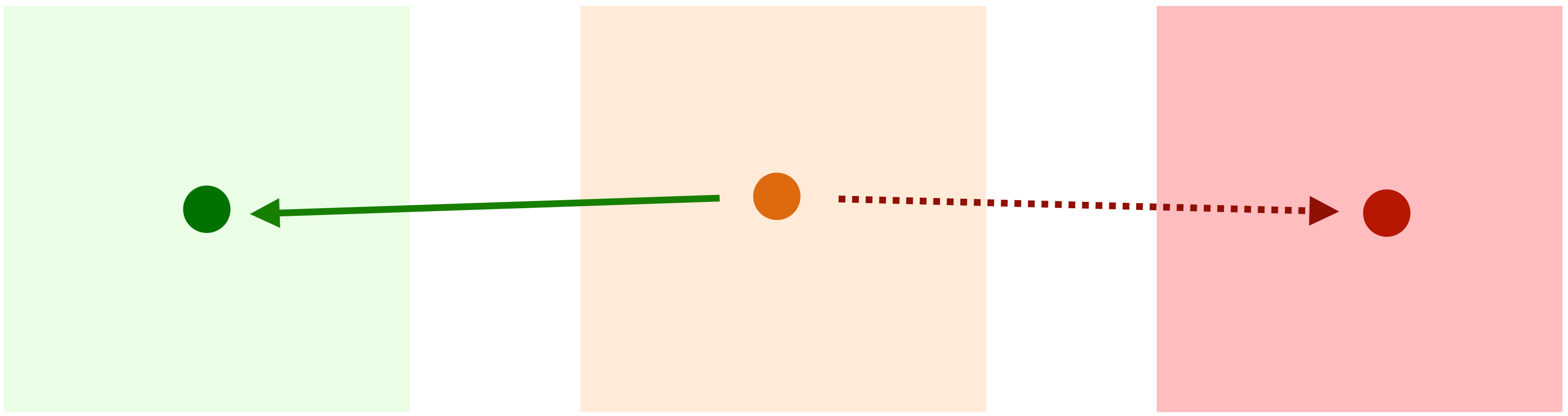
An autonomous robot =

software behavior coordination
hardware
localization planning social
actuation sensing interaction acceptance
control
computation learning
mapping liability
perception
energetics communication regulations

- **Takeways** of this talk**:**
  - Using co-design, it is easy to **embed** the synthesis of **controllers** into the co-design problem of the whole **autonomous robot**
  - Very **intuitive** modeling approach (no “acrobatics” needed)
  - **Rich modeling capabilities**: analytic models, catalogues, simulations
  - **Compositionality** and **modularity** allow **interdisciplinary collaboration**
  - Co-design produces **actionable information** for designers to **reason** about their problems

An abstract view of design problems

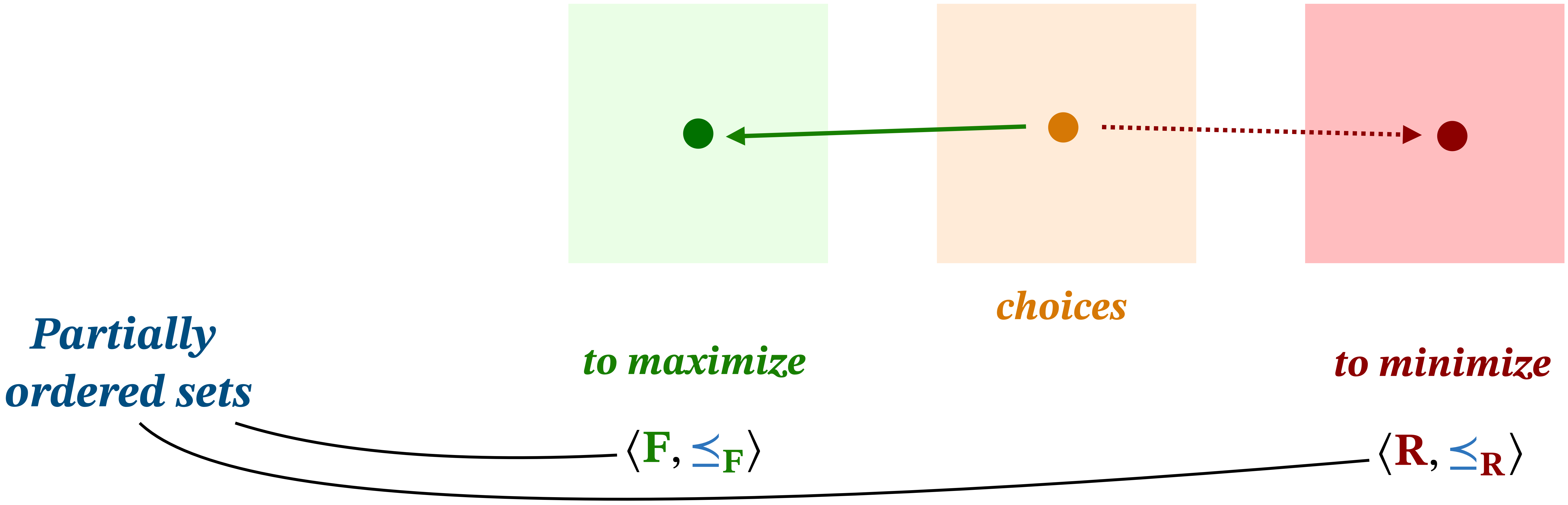
- Across fields, design or synthesis problems are defined with 3 spaces:
  - **implementation space**: the options we can choose from;
  - **functionality space**: what we need to provide/achieve;
  - **requirements/costs space**: the resources we need to have available;



functionality *(provided)*	***implementations***	***costs, resources*** *(required)*
desired behavior	*choices*	
specifications	*plans*	*requirenents*
objectives	*blueprints*	*dependencies*
guarantees	*decision variables*	
“function”	*“form”*	*“function”*
“conclusions”	*“proof”*	*“assumptions”*

An abstract view of design problems

- Across fields, design or synthesis problems are defined with 3 spaces:
  - **implementation space**: the options we can choose from;
  - **functionality space**: what we need to provide/achieve;
  - **requirements/costs space**: the resources we need to have available;



Partial orders allow to model various trade-offs

Definition. A *poset* is a tuple $\langle P, \preceq_P \rangle$, where P is a set and $\preceq_P$ is a partial order, defined as a reflexive, transitive, and antisymmetric relation.

- All **totally ordered sets** are particular cases of **partially ordered sets:**

$$\langle \mathbb{R}_{\geq 0}, \leq \rangle \qquad \langle \mathbb{N}, \leq \rangle$$

Partial orders allow to model various trade-offs

Definition. A *poset* is a tuple $\langle P, \preceq_P \rangle$, where P is a set and $\preceq_P$ is a partial order, defined as a reflexive, transitive, and antisymmetric relation.

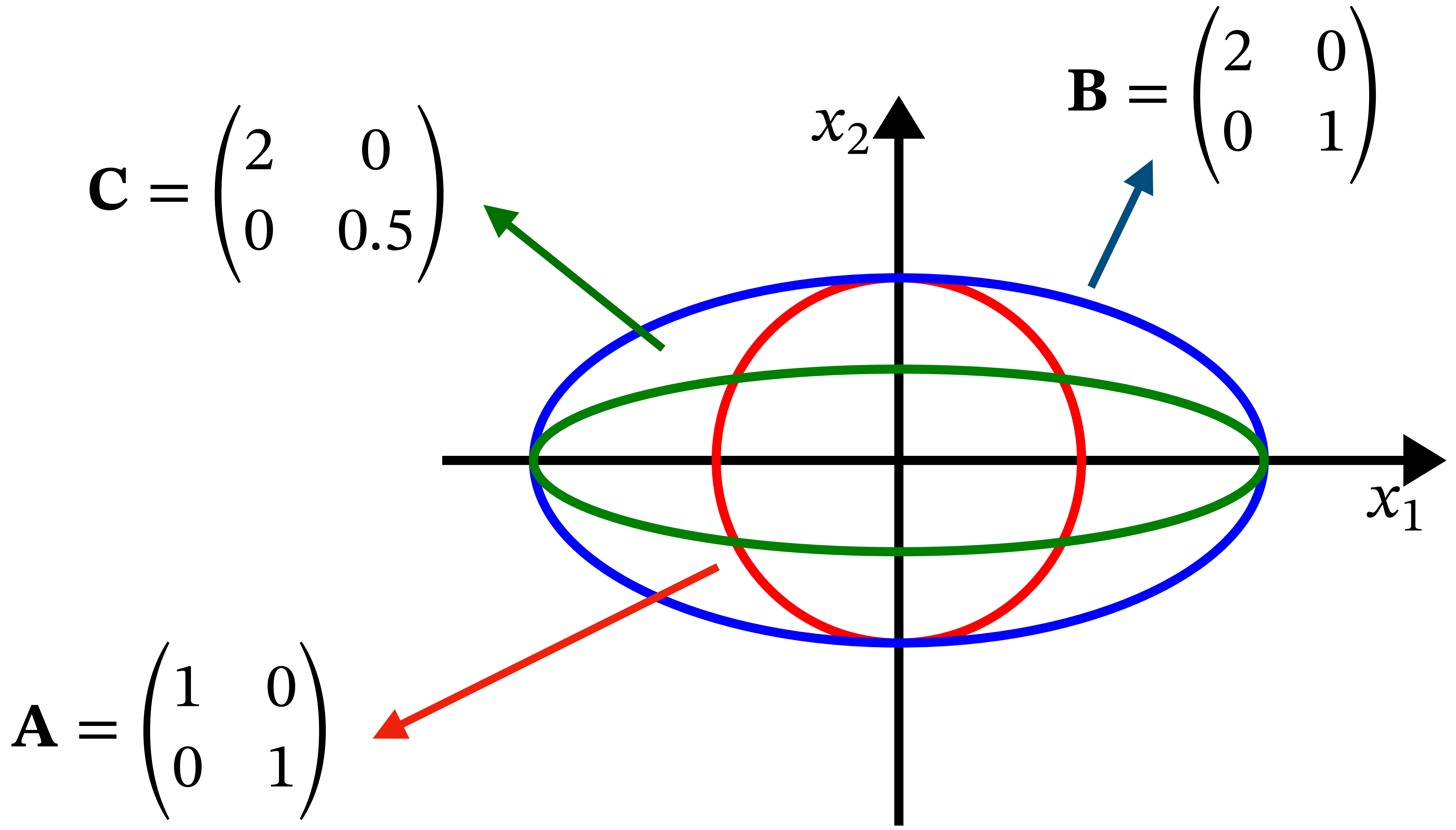
- All **totally ordered sets** are particular cases of **partially ordered sets:**

$$\left\langle \mathbb{R}_{\geq 0}, \leq \right\rangle \qquad \langle \mathbb{N}, \leq \rangle$$

- In this work, among others, we consider the poset of **positive semi-definite matrices**

Definition. A symmetric matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ is *positive semi-definite* if $x^\intercal \mathbf{M} x \geq 0$ for all non-zero $x \in \mathbb{R}^n$. We call the set of all such matrices $\mathcal{P}^n$.

- We can define a **partial order** as $\mathbf{A} \preceq \mathbf{B} \Leftrightarrow (\mathbf{B} - \mathbf{A}) \in \mathcal{P}^n, \quad \mathbf{A}, \mathbf{B} \in \mathcal{P}^n$
- Symmetric matrices have **real** eigenvalues
- Can be interpreted as **axes lengths** of **ellipsoids**
- Order is given by **ellipsoids inclusion**



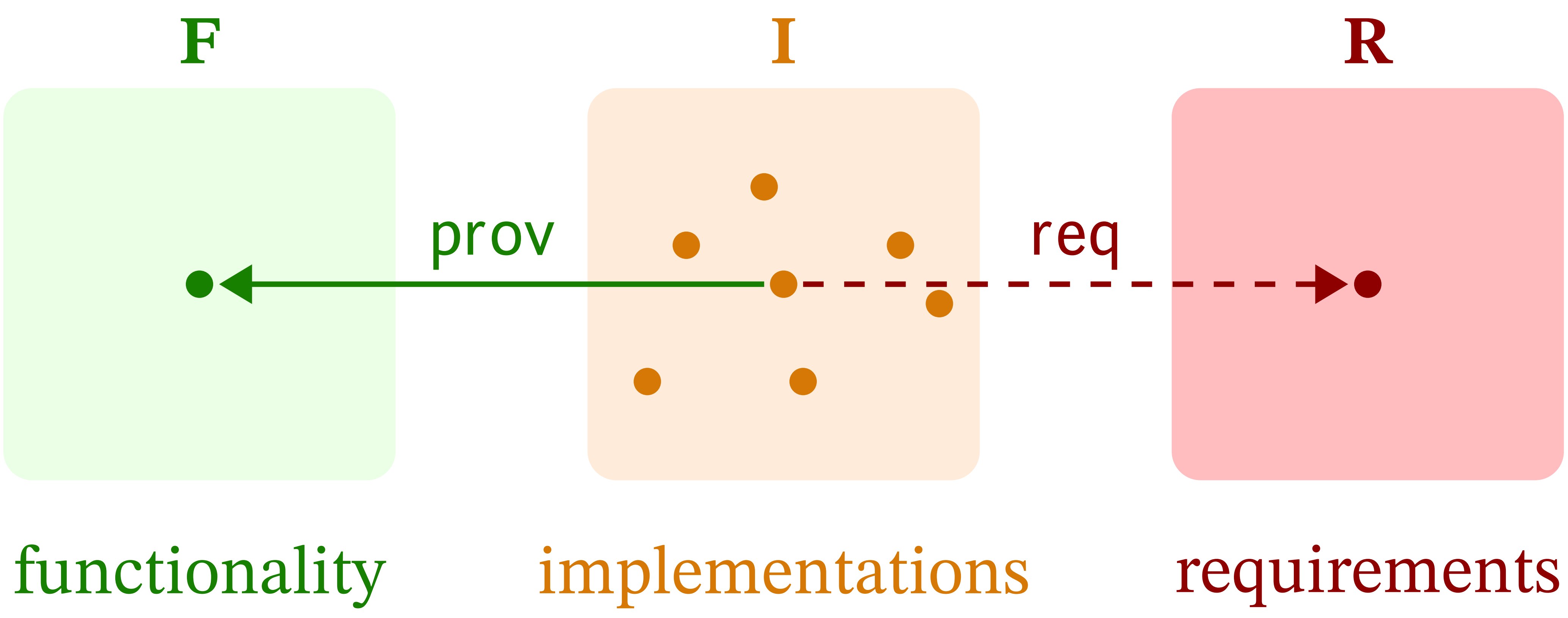
Design problem with implementation (DPIs)

Definition (Design problem with implementation). A *design problem with implementation* (DPI) is a tuple

$$\langle \mathbf{F}, \mathbf{R}, \mathbf{I}, \mathtt{prov}, \mathtt{req} \rangle ,$$

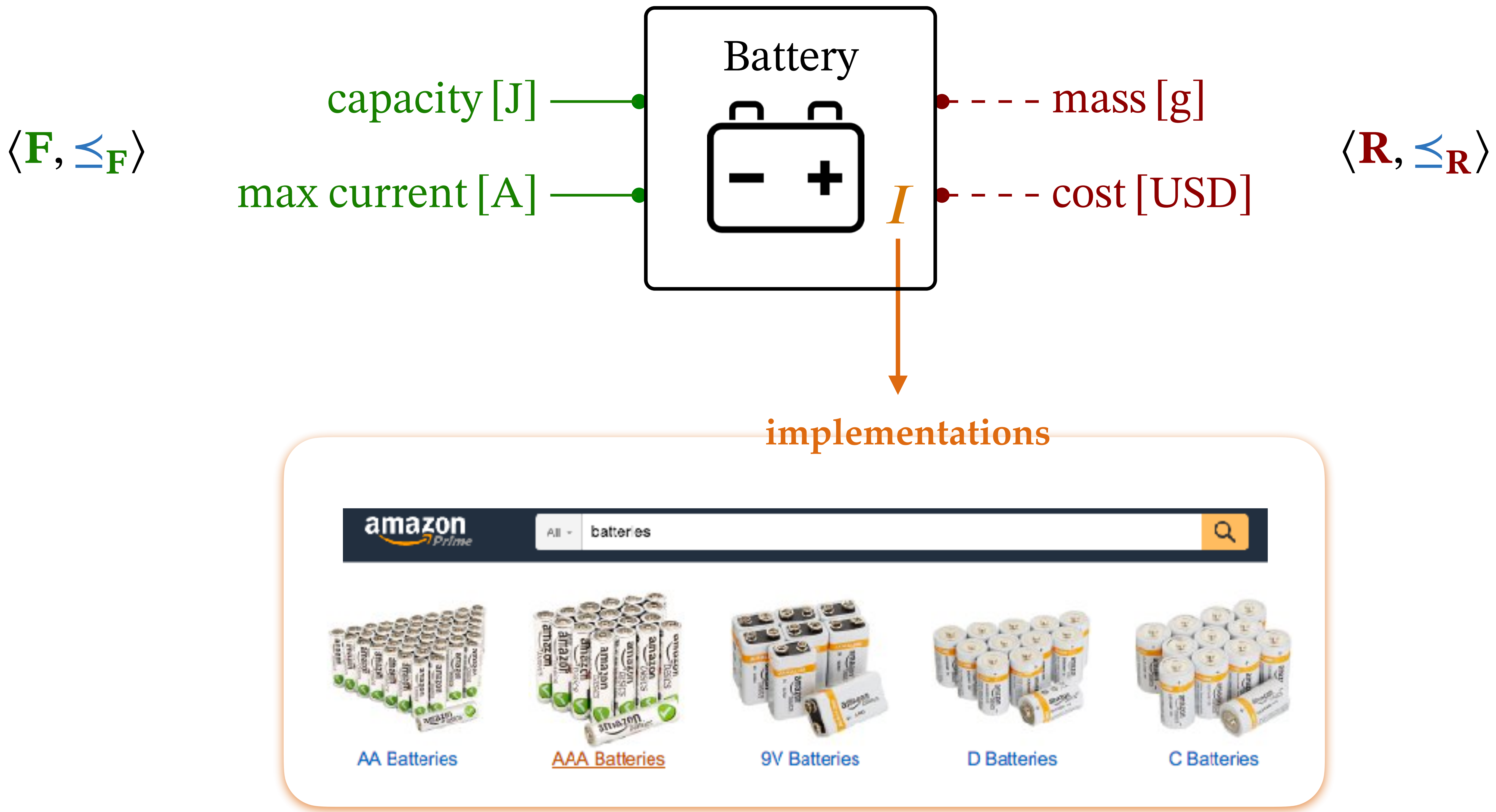
where:

- ▷ $\mathbf{F}$ is a poset, called *functionality space*;
- ▷ $\mathbf{R}$ is a poset, called *requirements space*;
- ▷ $\mathbf{I}$ is a set, called *implementation space*;
- ▷ the map $\mathtt{prov} : \mathbf{I} \to \mathbf{F}$ maps an implementation to the functionality it provides;
- ▷ the map $\mathtt{req} : \mathbf{I} \to \mathbf{R}$ maps an implementation to the resources it requires.



Graphical notation for DPIs

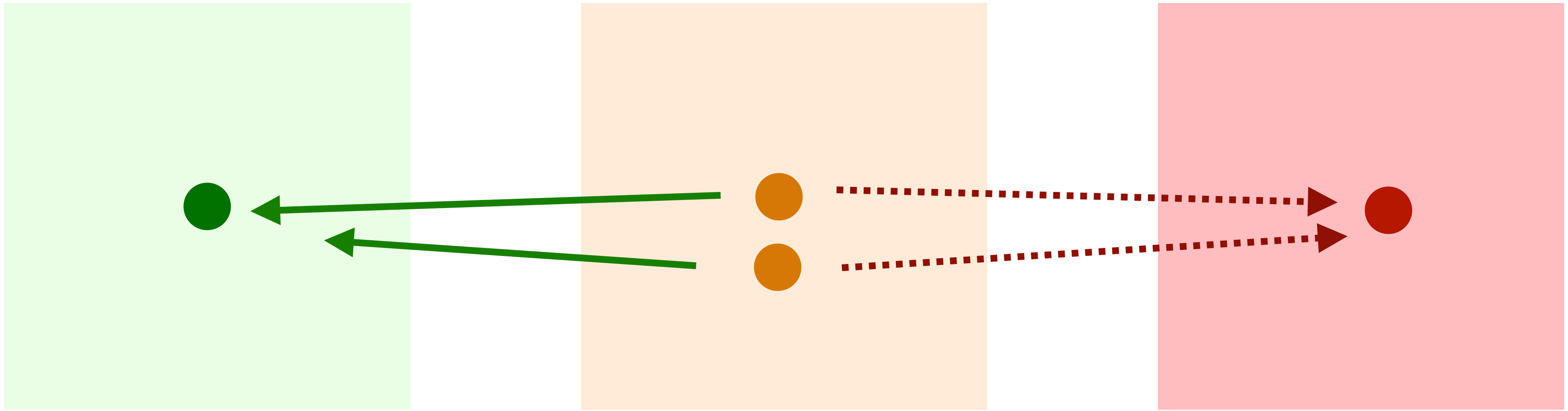
- We use this graphical notation:
  - functionality: **green continuous wires** on the left
  - requirements: **dashed red wires** on the right.



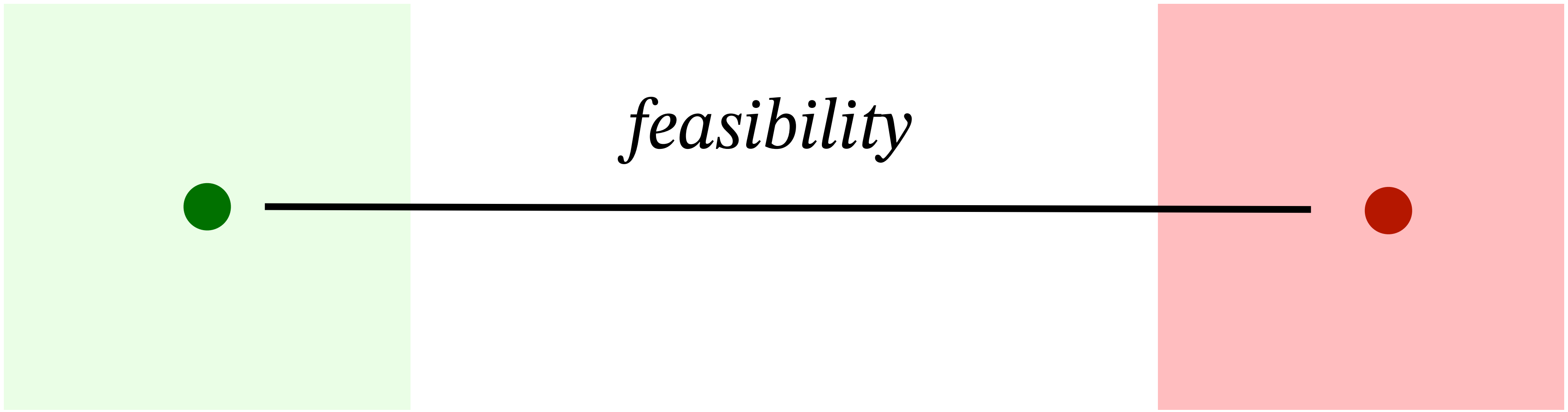
Engineering is constructive

- For the purpose of design, we **need to know how something is done**, not just that it is possible to do something: **engineering is constructive**.

- We need to know what are the implementation(s), if any, that relate functionality and costs.



- For the algorithmic solution of co-design problem, **it is useful to consider a direct feasibility relation** from functionality to costs.

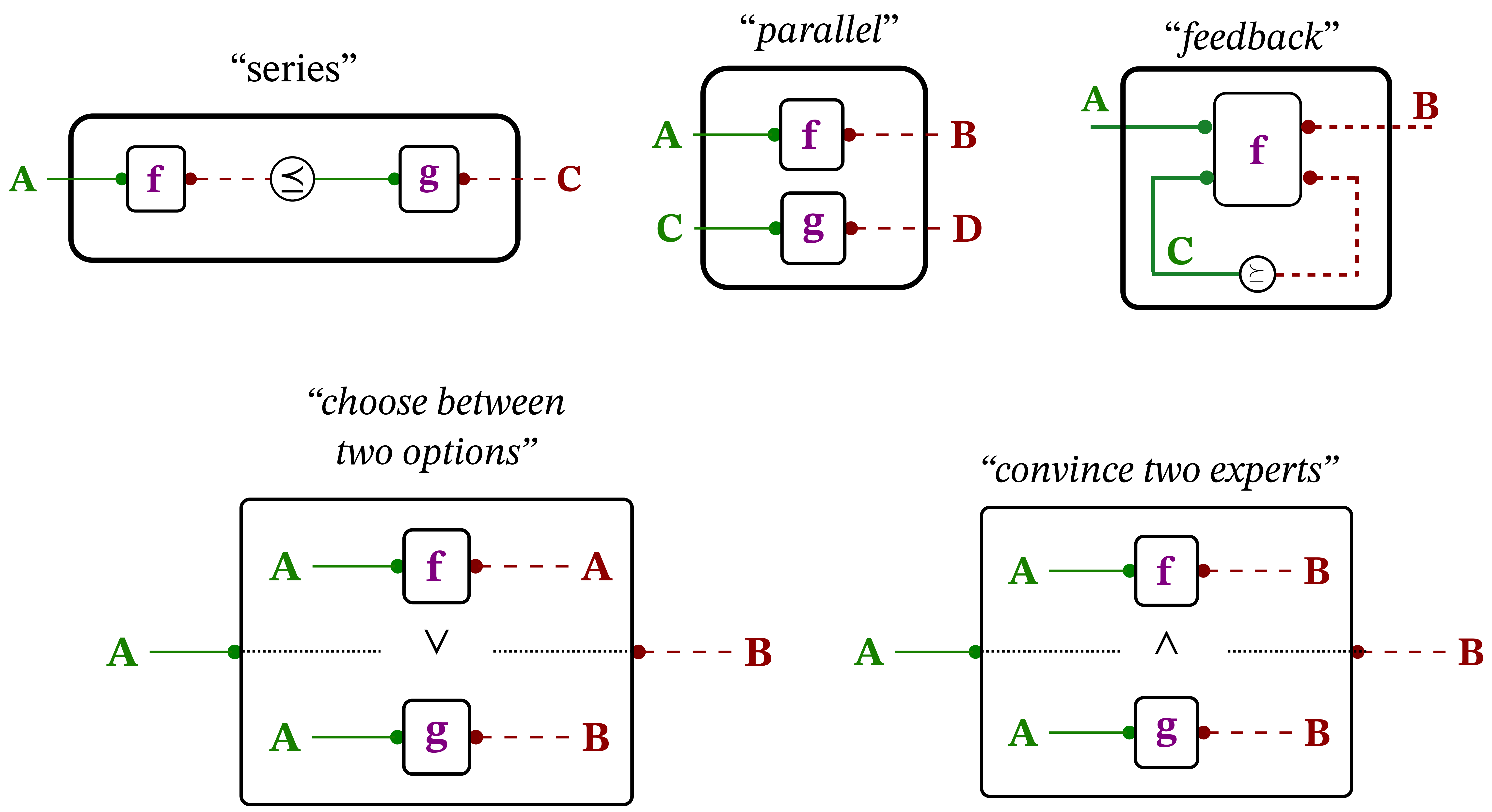


$$\mathbf{d} : \mathbf{F}^{\mathrm{op}} \times \mathbf{R} \to_{\mathbf{Pos}} \mathbf{Bool}$$

$$\langle f^*, r \rangle \mapsto \exists i \in \mathbf{I} : (f \preceq_{\mathbf{F}} \mathsf{prov}(i)) \wedge (\mathsf{req}(i) \preceq_{\mathbf{R}} r)$$

- **Monotone** map: **Lower functionalities** does **not** require **more resources**, **higher resources** do not provide **less functionalities**

Composition operators

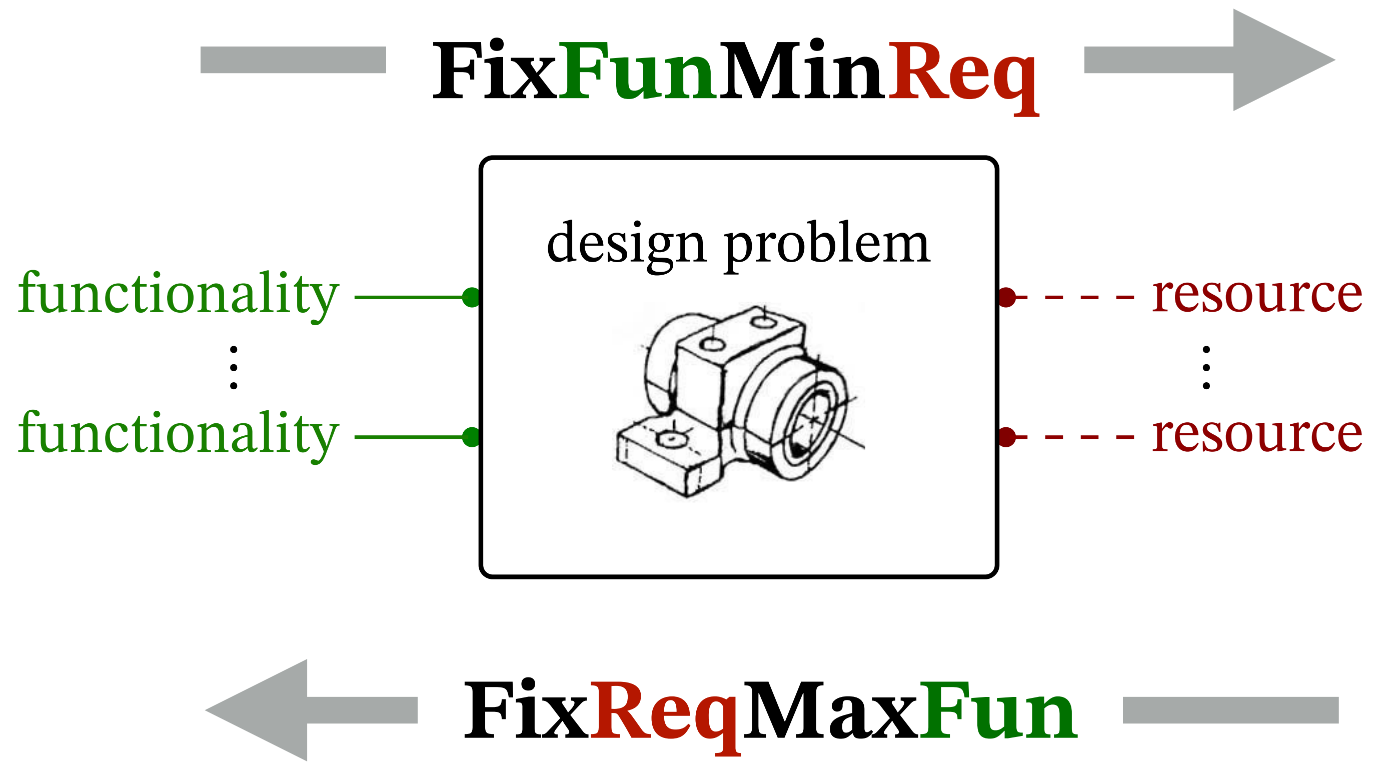


- The **composition** of any two **DPs** returns a **DP** (closure)
- Very practical tool to **decompose** large **problems** into **subproblems**

Design queries

- Two basic design queries are:
  - **FixFunMinReq**: Fixed a lower bound on functionality, minimize the resources.
  - **FixReqMaxFun**: Fixed an upper bound on the resource, maximize the functionality

Given the functionality to be provided,
what are the **minimal resources** required?



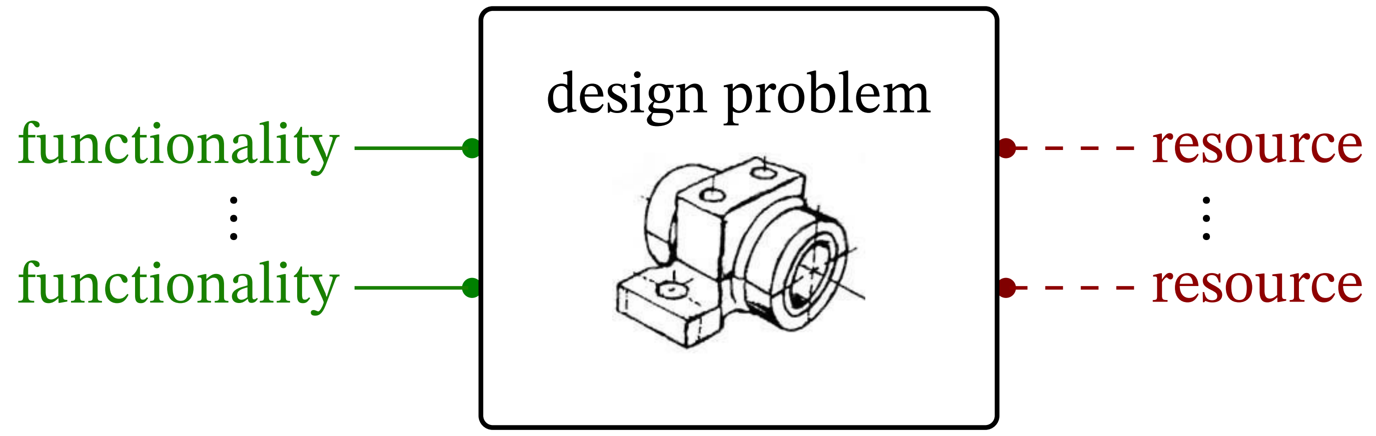
Given the resources that are available, what is
the **maximal functionality** that can be provided?

Design queries

- Two basic design queries are:
  - **FixFunMinReq**: Fixed a lower bound on functionality, minimize the resources.
  - **FixReqMaxFun**: Fixed an upper bound on the resource, maximize the functionality

Given the functionality to be provided, what are the **minimal resources** required?

FixFunMinReq



FixReqMaxFun

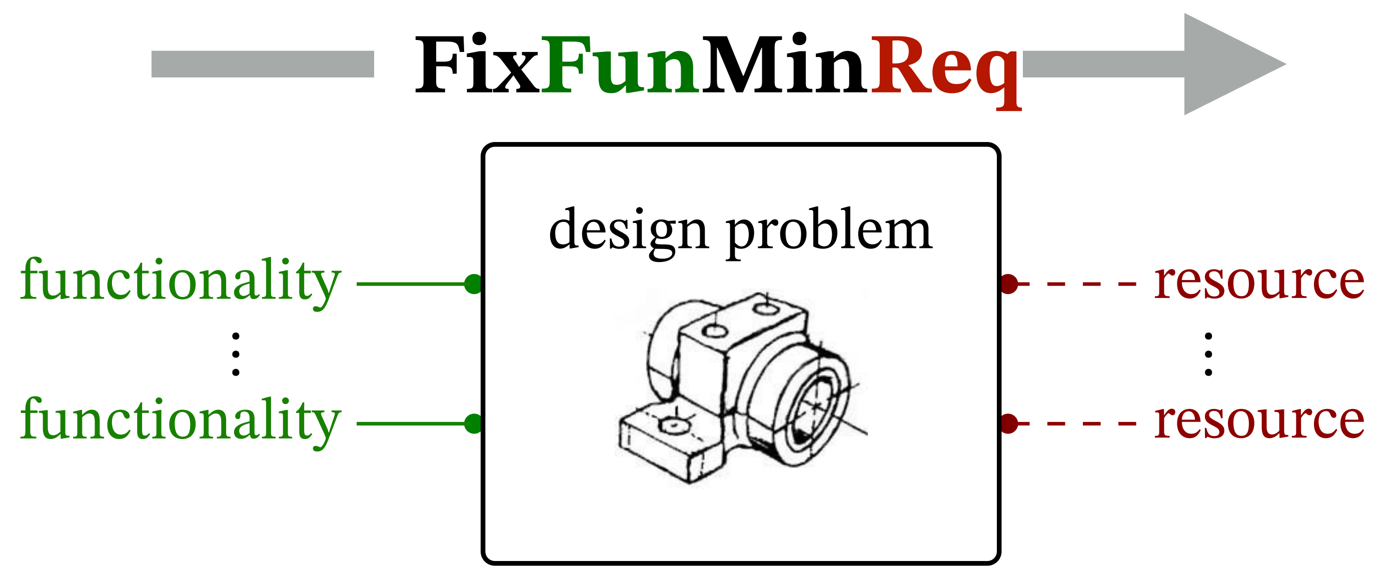
Given the resources that are available, what is the **maximal functionality** that can be provided?

- The two problems are **dual**
- From the solutions, one can retrieve the **implementations** (design choices)

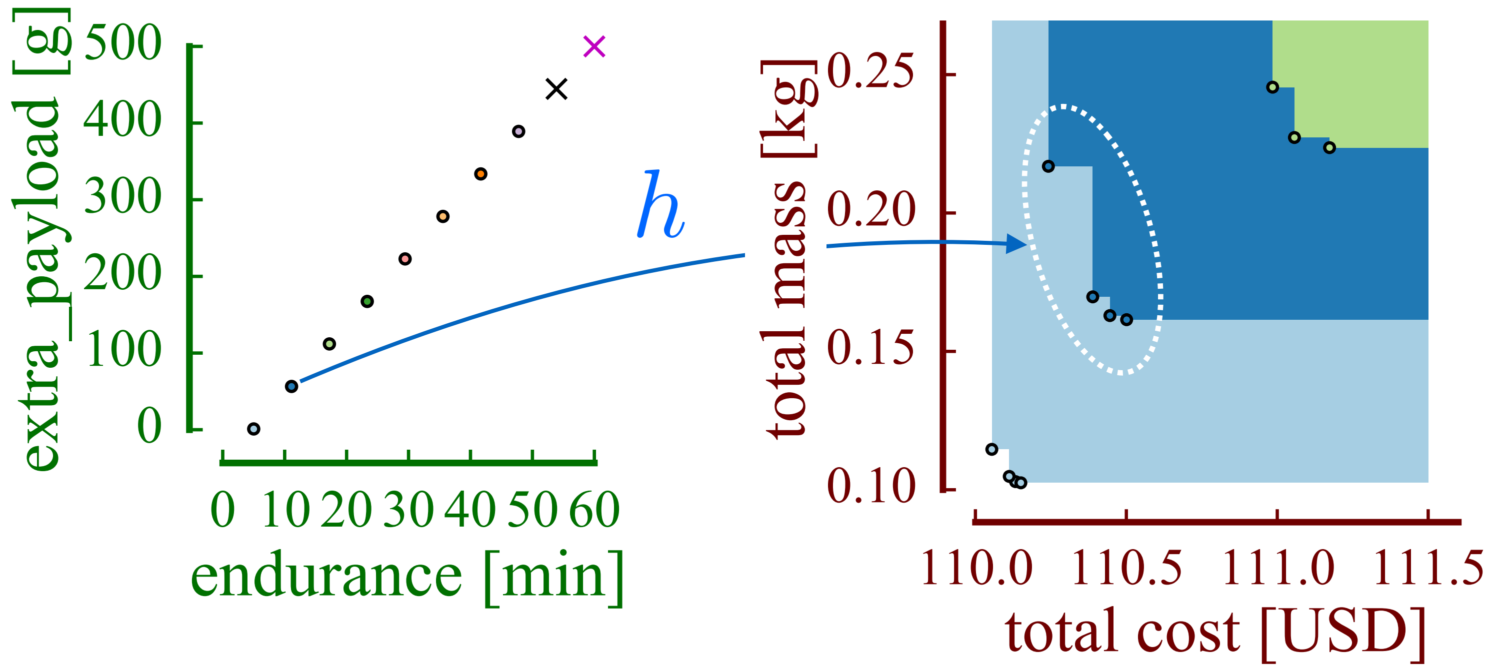
Design queries

- Two basic design queries are:
  - **FixFunMinReq**: Fixed a lower bound on functionality, minimize the resources.
  - **FixReqMaxFun**: Fixed an upper bound on the resource, maximize the functionality

Given the functionality to be provided,
what are the **minimal resources** required?



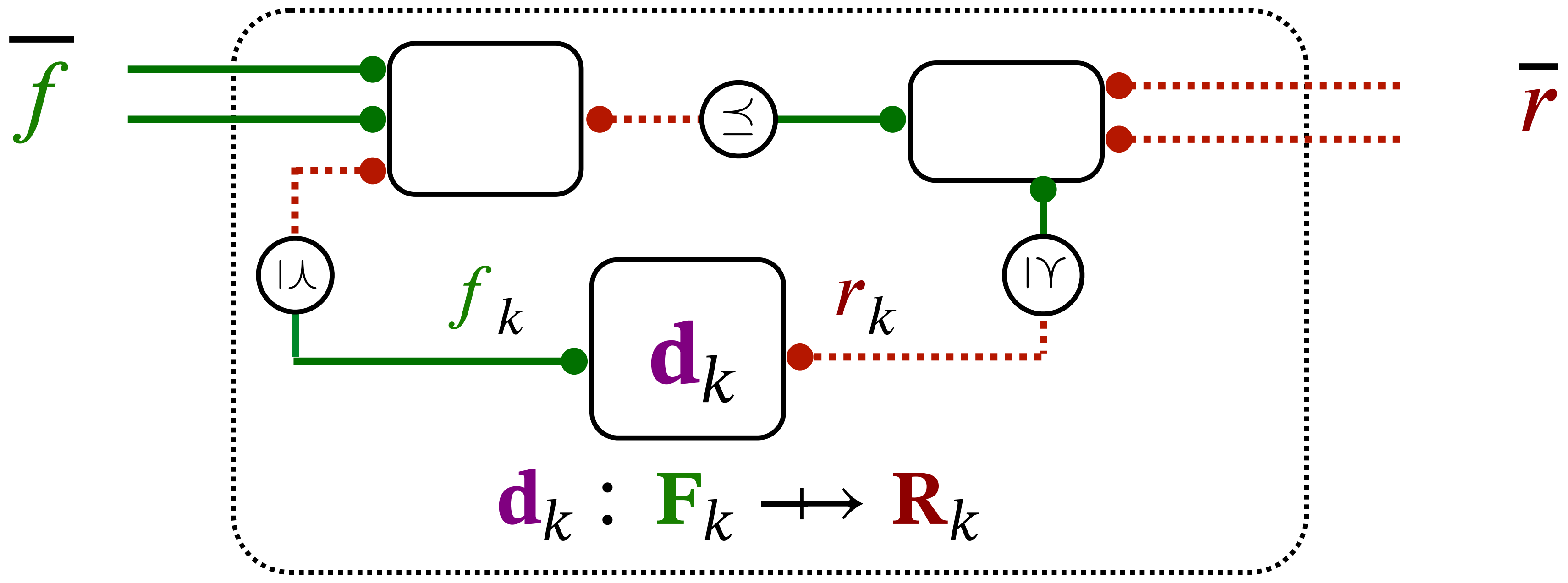
- We are looking for:
  - A map from functionality to upper sets of feasible resources: $h: \mathbf{F} \to \mathcal{U}\mathbf{R}$
  - A map from functionality to antichains of minimal resources: $h: \mathbf{F} \to \mathcal{A}\mathbf{R}$



Optimization semantics

- This is the semantics of **FixFunMinReq** as a **family of optimization problems.**

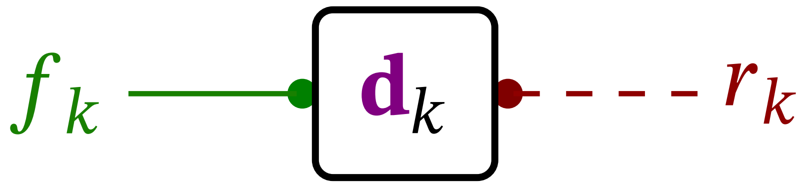
chosen by user

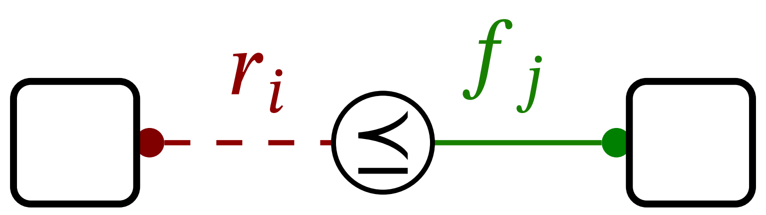


to minimize

variables $r_k \in \langle \mathbf{R}_k, \preceq_{\mathbf{R}_k} \rangle$ $\quad f_k \in \langle \mathbf{F}_k, \preceq_{\mathbf{F}_k} \rangle$

constraints *for **each node**:* *for **each edge**:*





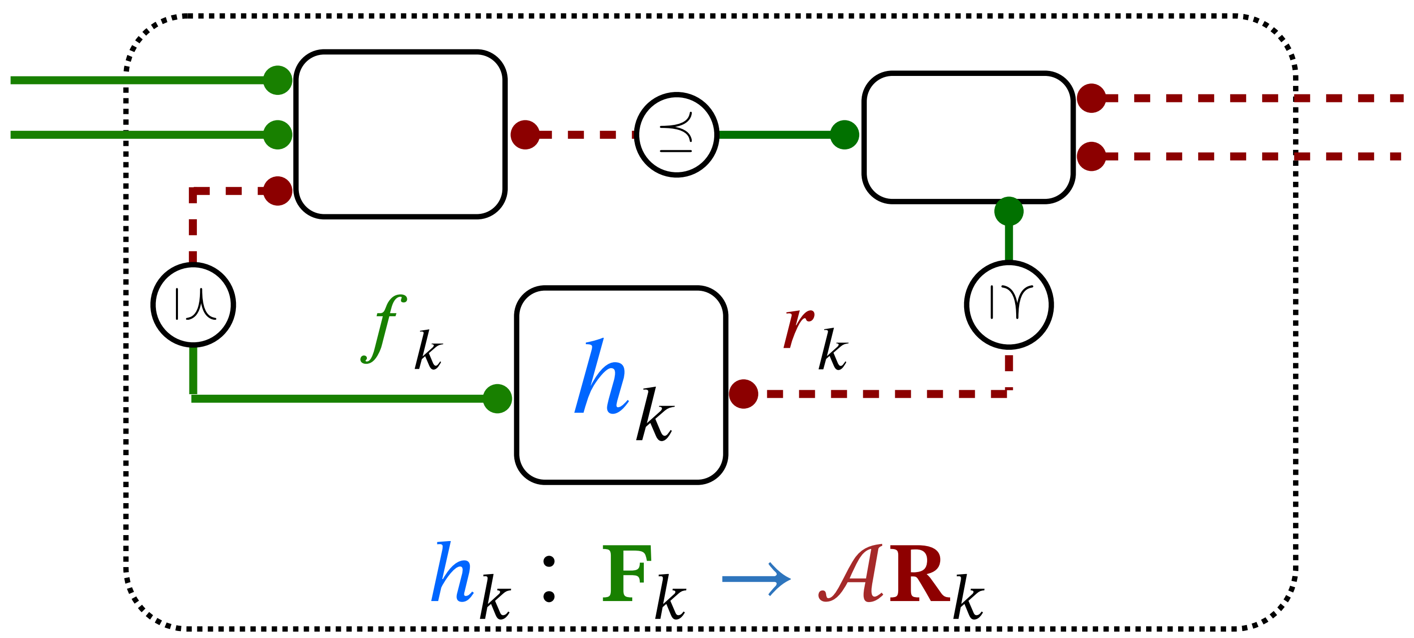
$\mathbf{d}_k(f_k^*, r_k) = \top$ $\quad r_i \preceq f_j$

objective $\operatorname{Min}_{\preceq} \bar{r}$

- ! not convex
- ! not differentiable
- ! not continuous
- ! not even defined on continuous spaces

Solving DP queries

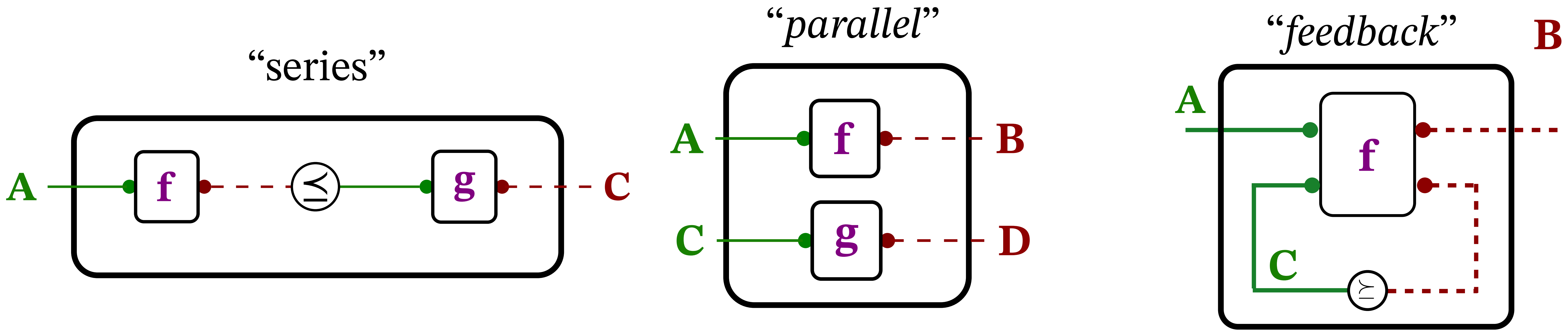
- Suppose we are given the function $h_k : \mathbf{F}_k \to \mathcal{A}\mathbf{R}_k$ for all nodes in the co-design graph.



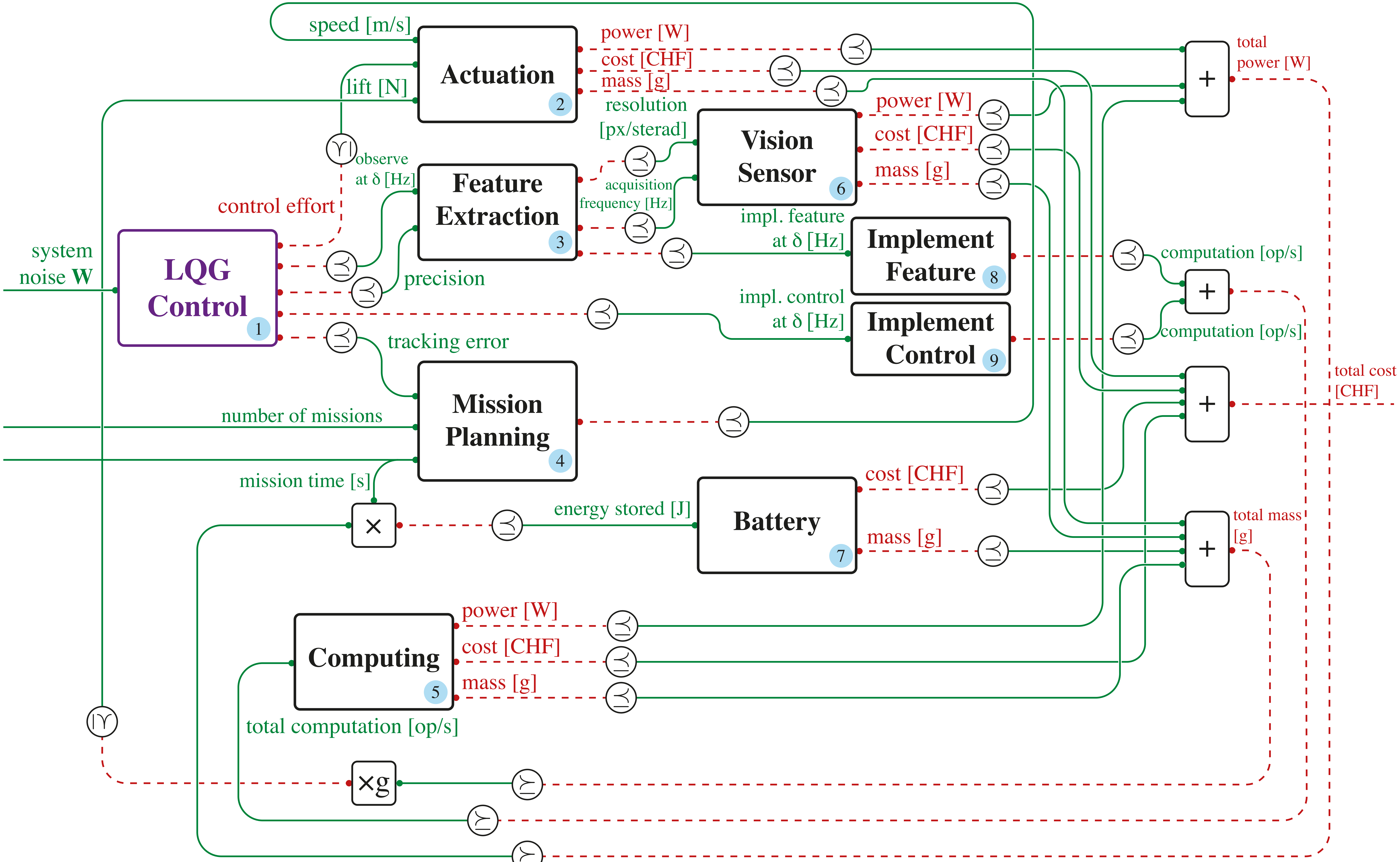
- Can we find the map $h : \mathbf{F} \to \mathcal{A}\mathbf{R}$ for the entire diagram?

- **Recursive approach:** We just need to work out the the composition formulas for all operations we have defined

- The set of **minimal** feasible resources can be obtained as the **least fixed point** of a monotone function in the space of anti-chains.



Use case: Co-design of an autonomous drone



Infinite-horizon LQG control in one slide

- Let's consider the **continuous time**, **stochastic** dynamics

$$\begin{aligned} \mathrm{d}\mathbf{x}_t &= \mathbf{A}\mathbf{x}_t \mathrm{d}t + \mathbf{B}\mathbf{u}_t \mathrm{d}t + \mathbf{E}\mathrm{d}\mathbf{w}_t \\ \mathrm{d}\mathbf{y}_t &= \mathbf{C}\mathbf{x}_t \mathrm{d}t + \mathbf{G}\mathrm{d}\mathbf{v}_t, \end{aligned}$$

where **A, B, C**, **D**, **E** , **G** are of adequate dimensions, $\mathbf{v}_t$ and $\mathbf{w}_t$ Brownian processes, and $\mathbf{W} = \mathbf{E}\mathbf{E}^*$, $\mathbf{V} = \mathbf{G}\mathbf{G}^*$ noise covariances

- We consider the classic **infinite-horizon LQG problem**, finding a control law minimizing the cost

$$J = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left\{\int_0^T \left(\left(\mathbf{x}_t^\intercal \mathbf{Q}\mathbf{x}_t\right) + \left(\mathbf{u}_t^\intercal \mathbf{R}\mathbf{u}_t\right)\right) \mathrm{d}t\right\}$$

where **Q** is a positive semi-definite matrix and **R** is a positive definite matrix

- **Well-known lemma:** the optimal control law for the problem is

$$\mathbf{u}_t^\star = -\mathbf{K}\hat{\mathbf{x}}_t = -\mathbf{R}^{-1}\mathbf{B}^*\bar{\mathbf{S}}\hat{\mathbf{x}}_t$$

where $\hat{\mathbf{x}}_t$ is the unbiased minimum-variance estimate of $\mathbf{x}_t$, and $\bar{\mathbf{S}}$ solves the Riccati equation $\mathbf{S}\mathbf{A} + \mathbf{A}^*\mathbf{S} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^*\mathbf{S} + \mathbf{Q} = \mathbf{0}$.

- We can obtain the **optimal cost**

$$\begin{aligned} J^\star &= \mathrm{Tr}(\bar{\mathbf{S}}\bar{\boldsymbol{\Sigma}}\mathbf{C}^*\mathbf{V}^{-1}\mathbf{C}\bar{\boldsymbol{\Sigma}} + \bar{\boldsymbol{\Sigma}}\mathbf{Q}) \\ &= \mathrm{Tr}(\bar{\boldsymbol{\Sigma}}\bar{\mathbf{S}}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^*\bar{\mathbf{S}} + \bar{\mathbf{S}}\mathbf{W}), \end{aligned}$$

where $\bar{\boldsymbol{\Sigma}}$ solves the Riccati equation $\mathbf{A}\boldsymbol{\Sigma} + \boldsymbol{\Sigma}\mathbf{A}^* - \boldsymbol{\Sigma}\mathbf{C}^*\mathbf{V}^{-1}\mathbf{C}\boldsymbol{\Sigma} + \mathbf{W} = \mathbf{0}$.

LQG control as a co-design problem

- Let's consider the **performance** metrics

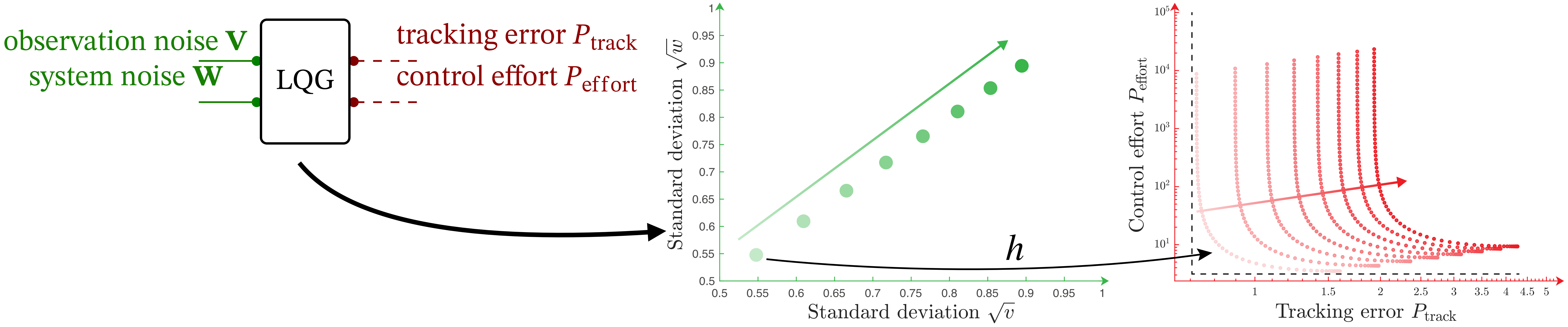
$$P_{\text{track}} = \lim_{t\to\infty} \mathbb{E}\{\mathbf{x}_t^\intercal \mathbf{Q} \mathbf{x}_t\} \quad P_{\text{effort}} = \lim_{t\to\infty} \mathbb{E}\{\mathbf{u}_t^\intercal \mathbf{R} \mathbf{u}_t\}$$

LQG control as a co-design problem

- Let's consider the **performance** metrics

$$P_{\text{track}} = \lim_{t\to\infty} \mathbb{E}\{\mathbf{x}_t^\intercal \mathbf{Q} \mathbf{x}_t\} \quad P_{\text{effort}} = \lim_{t\to\infty} \mathbb{E}\{\mathbf{u}_t^\intercal \mathbf{R} \mathbf{u}_t\}$$

- **Theorem**: We can write the **LQG** problem as a design problem of the form:

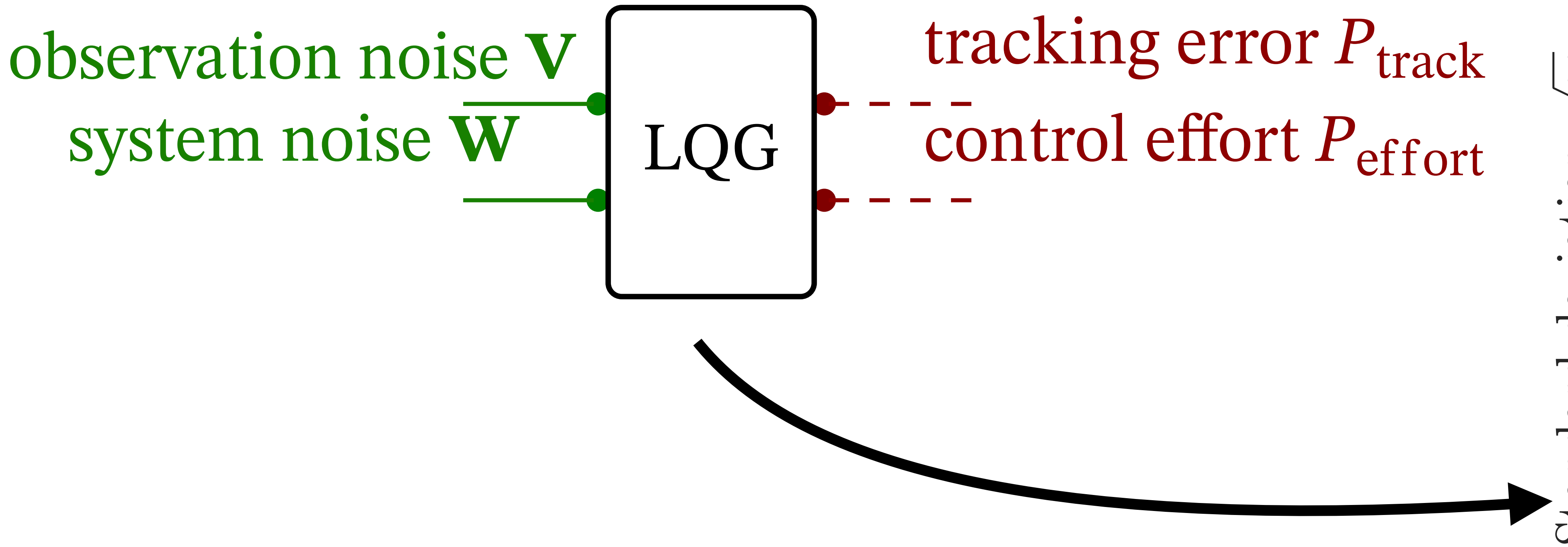


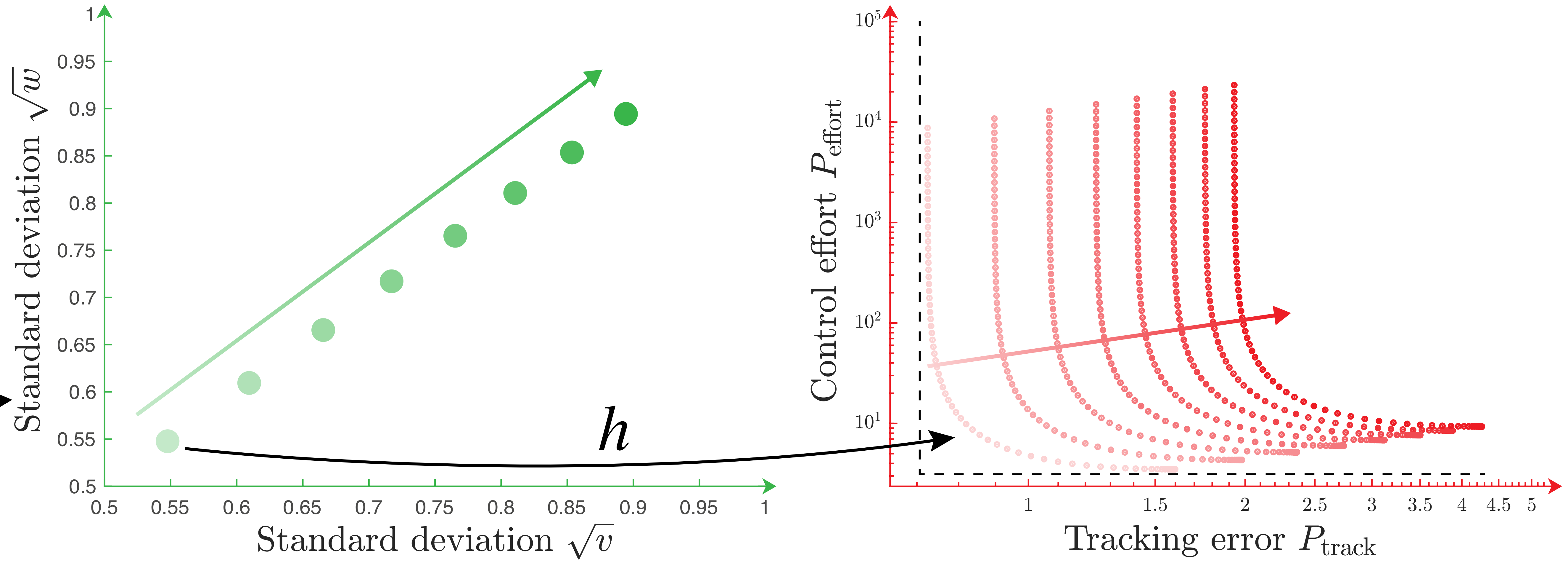
LQG control as a co-design problem

- Let's consider the **performance** metrics

$$P_{\text{track}} = \lim_{t\to\infty} \mathbb{E}\{\mathbf{x}_t^\intercal \mathbf{Q} \mathbf{x}_t\} \quad P_{\text{effort}} = \lim_{t\to\infty} \mathbb{E}\{\mathbf{u}_t^\intercal \mathbf{R} \mathbf{u}_t\}$$

- **Theorem**: We can write the **LQG** problem as a design problem of the form:





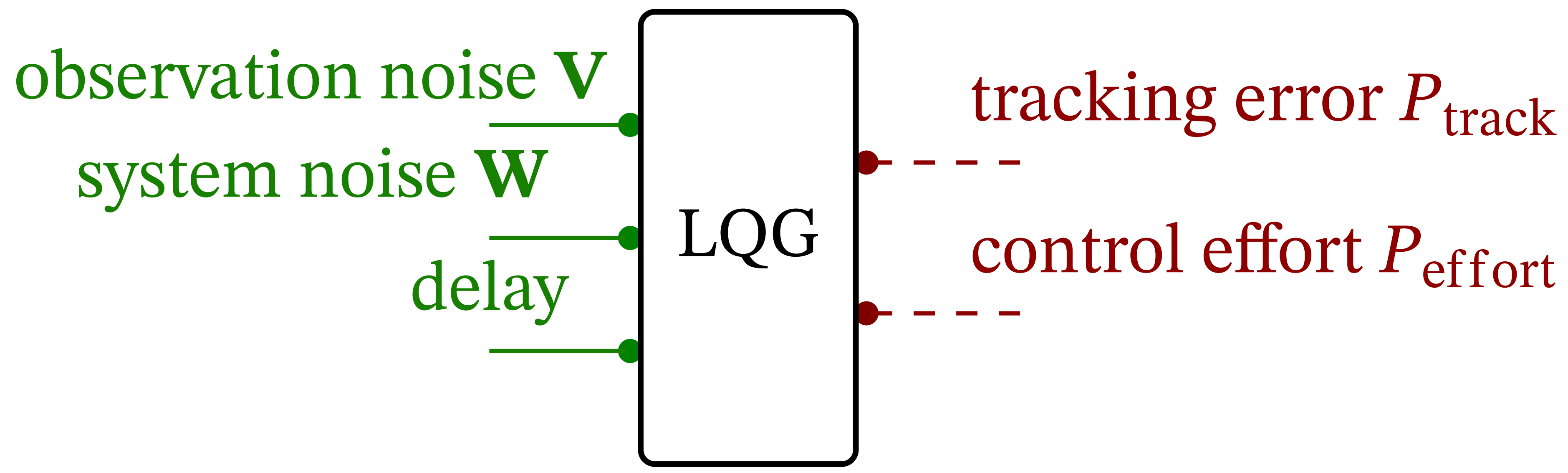
- **Proof procedure** in four steps:
  - Show that one can *rewrite* the performance metrics as

$$\lim_{t\to\infty} \mathbb{E}\{\mathbf{x}_t^\intercal \mathbf{Q}_0 \mathbf{x}_t\} = \mathrm{Tr}(\mathbf{Q}_0 (\mathbf{\Sigma} + \mathbf{F})) \quad \lim_{t\to\infty} \mathbb{E}\{\mathbf{u}_t^\intercal \mathbf{R}_0 \mathbf{u}_t\} = \mathrm{Tr}(\mathbf{S}\mathbf{B}^* \mathbf{R}^{-1} \mathbf{R}_0 \mathbf{R}^{-1} \mathbf{B}\mathbf{S}\mathbf{F}),$$

    where $\mathbf{F}$ solves the Lyapunov equation $(\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{F} + \mathbf{F}(\mathbf{A} - \mathbf{B}\mathbf{K})^* + \mathbf{L}\mathbf{V}\mathbf{L}^* = \mathbf{0}$ and $\mathbf{L} = \mathbf{\Sigma}\mathbf{C}^*\mathbf{V}^{-1}$
  - Show **monotonicity** of **tracking error** and **control effort** performances with respect to $\mathbf{Q}$ and $\mathbf{R}$
  - Show $\langle \mathbf{V}, \mathbf{W} \rangle \preceq \langle \mathbf{V}', \mathbf{W}' \rangle \Rightarrow \mathbf{\Sigma}(\mathbf{V}, \mathbf{W}) \preceq \mathbf{\Sigma}(\mathbf{V}', \mathbf{W}')$
  - Show **monotonicity** of **tracking** and **effort** with respect to $\mathbf{V}$ and $\mathbf{W}$

LQG control with delays and the discrete version

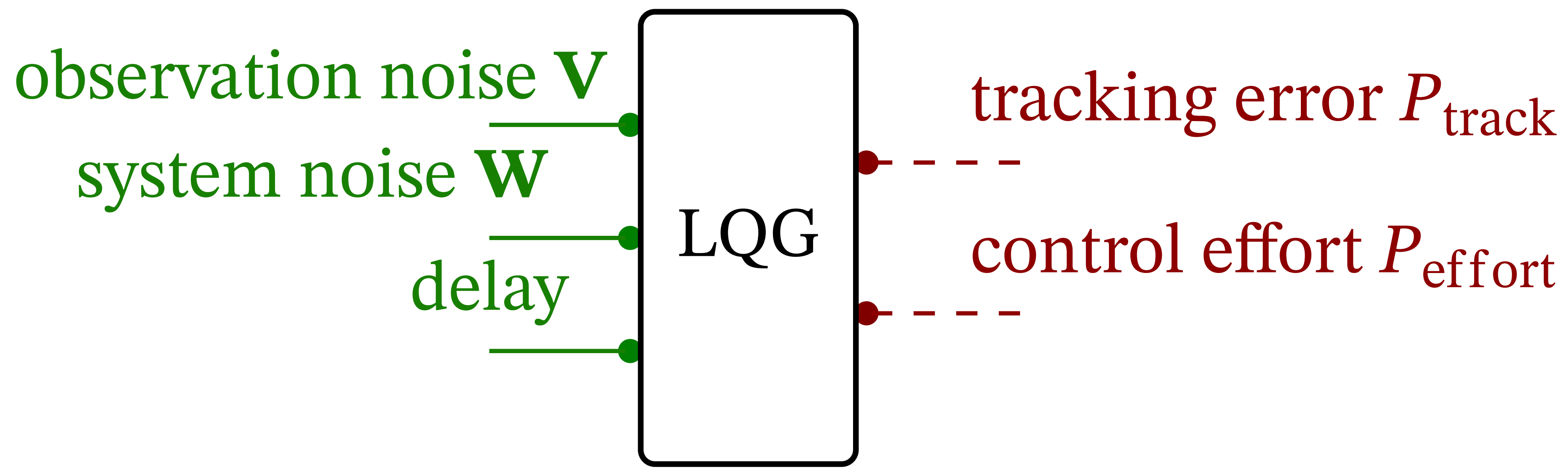
- **Theorem:** For the LQG problem with **observation** and **computation delays** we can write the design problem:



- **Proof sketch:**
  - *Substitution principle:* ***If in the case a certain nuisance is "lower", the controller could simulate a "higher" nuisance***

LQG control with delays and the discrete version

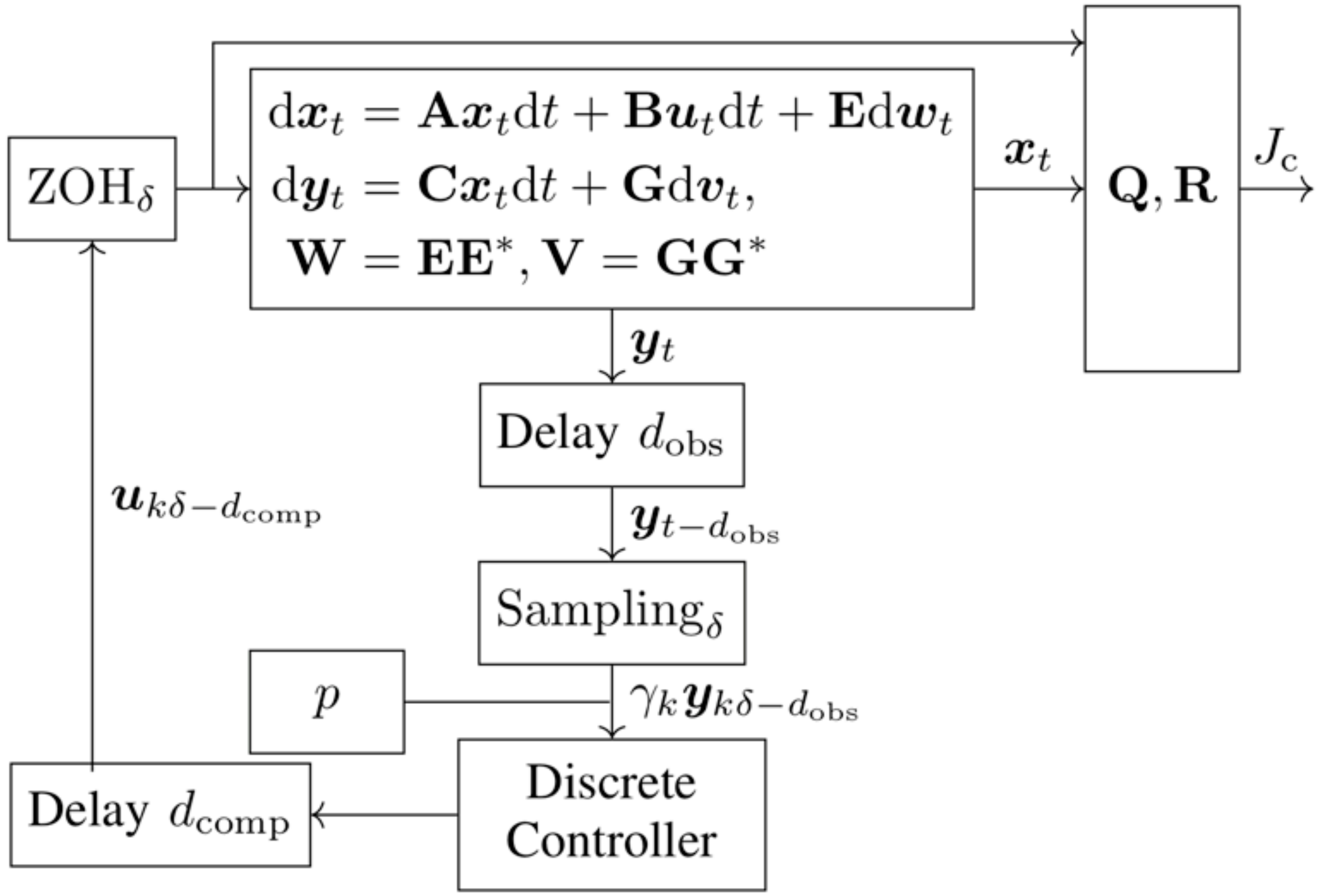
- **Theorem:** For the LQG problem with **observation** and **computation delays** we can write the design problem:



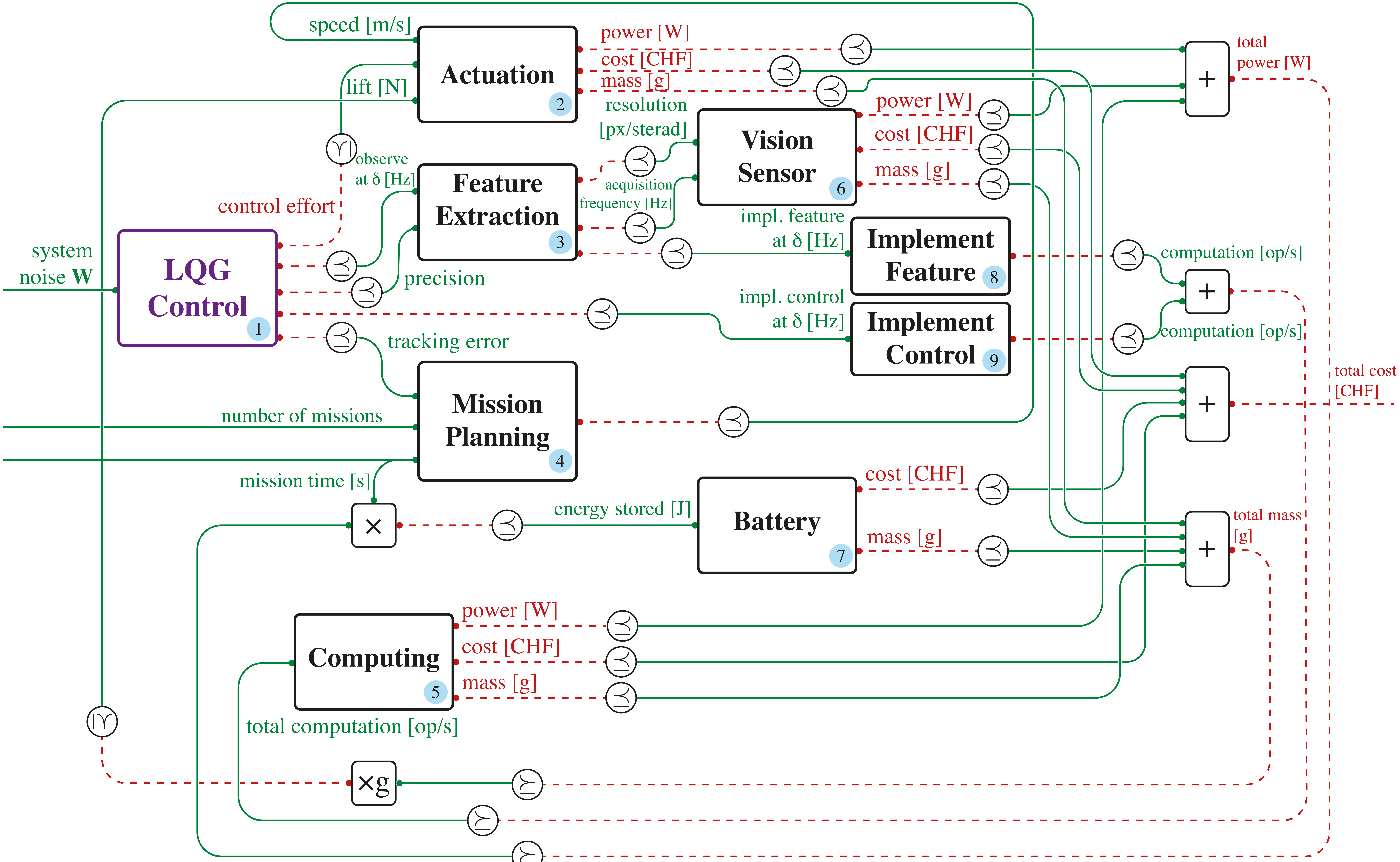
- **Proof sketch:**
  - *Substitution principle:* ***If in the case a certain nuisance is "lower", the controller could simulate a "higher" nuisance***

- Analogous statements can be proven for the **discrete-time** case

- **Theorem:** One can write a design problem of the form:



Use case: Co-design of an autonomous drone



Use case: Co-design of an autonomous drone

Co-design is very intuitive!

- The theory comes with a **formal language** and a **solver (MCDP)**

- Very intuitive to use:

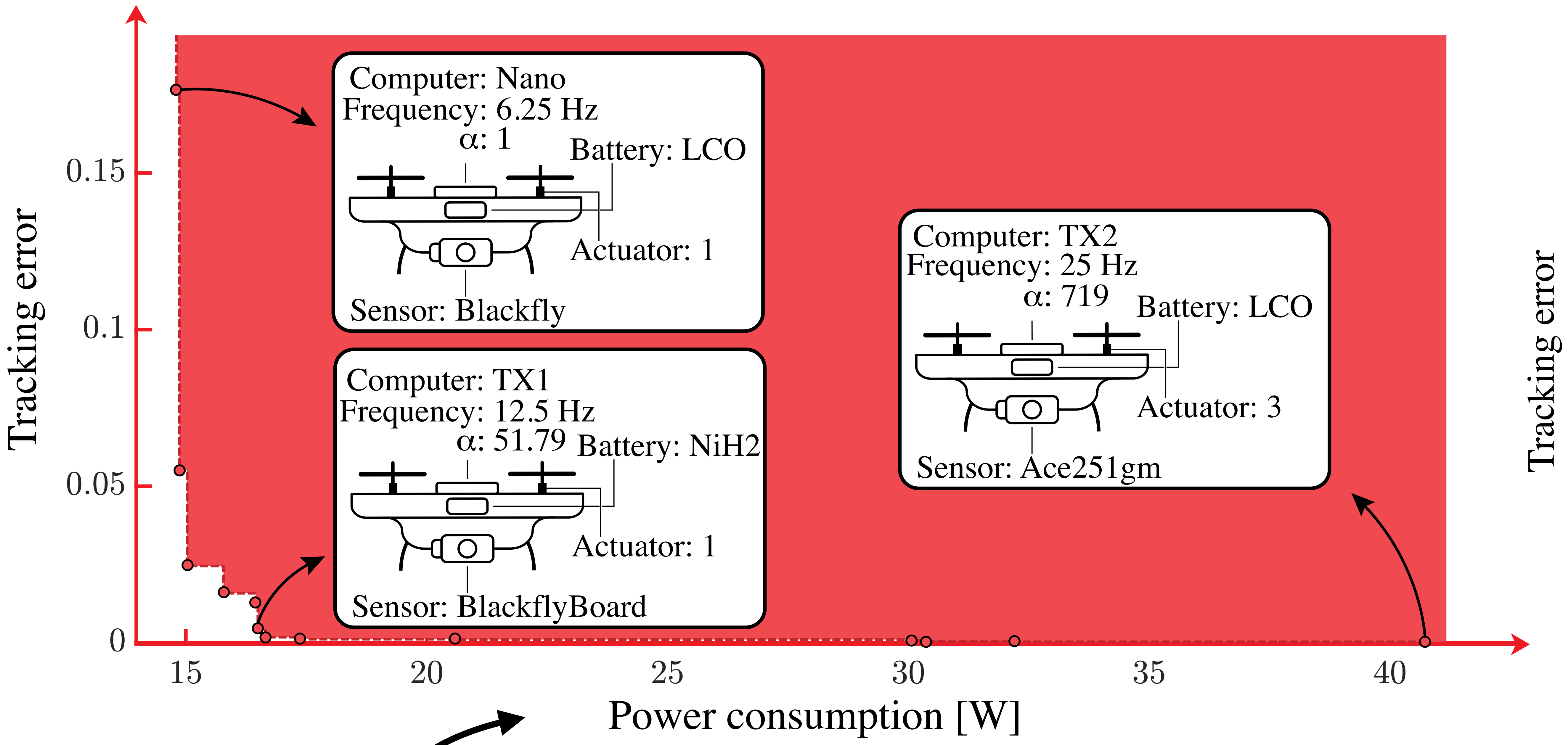
```
mcdp {
    provides computation [op/s]
    requires cost [CHF]
    requires mass [g]
    requires power [W]

}
```

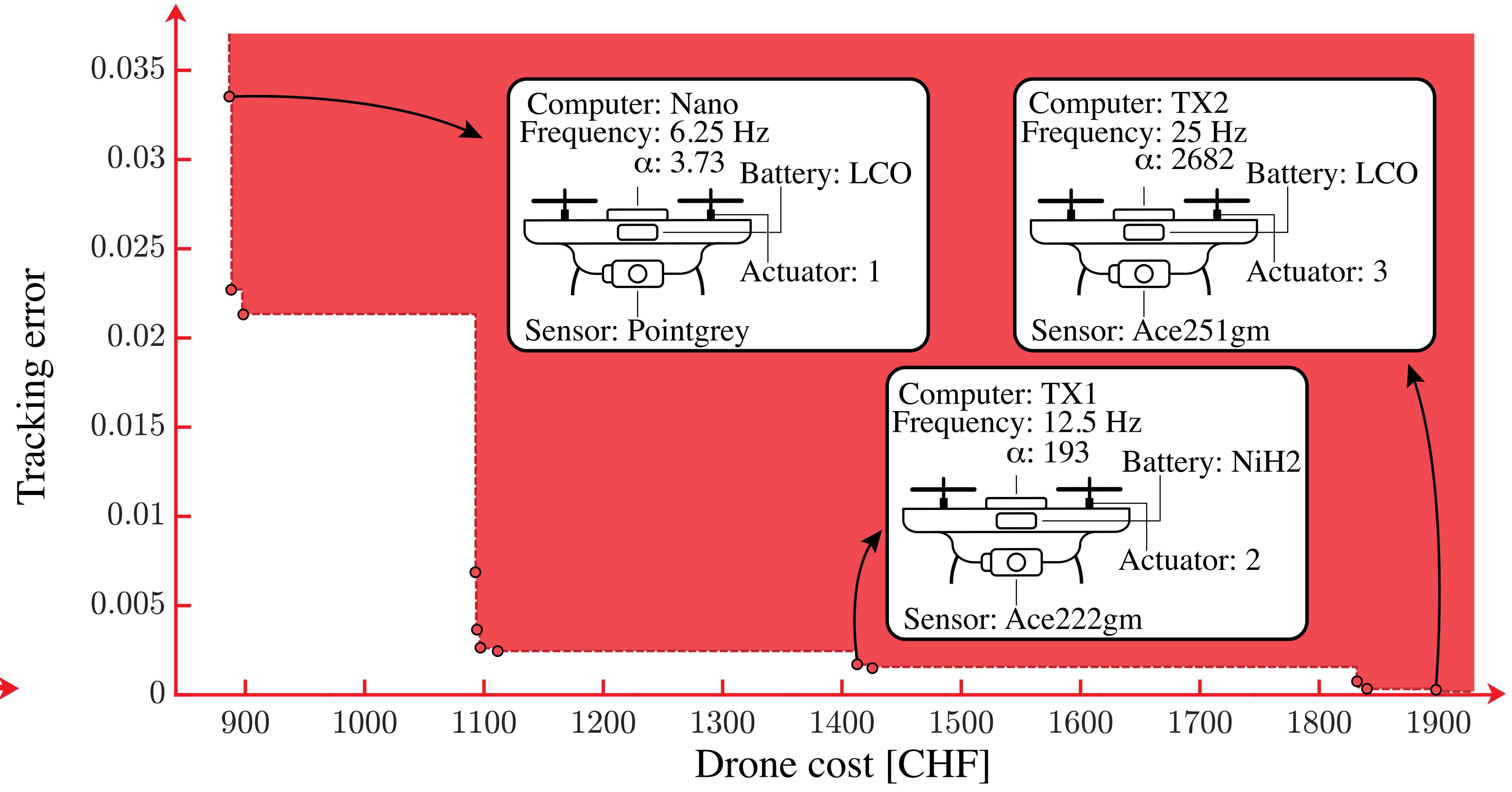
```
choose(
        SedanS: (load Car_SedanS),
        SedanM: (load Car_SedanM),
        SedanL: (load Car_SedanL),
        SUVS: (load Car_SuvS),
        SUVM: (load Car_SuvM),
        Minivan: (load Car_Minivan),
        Shuttle: (load Car_Shuttle),
        Hybrid: (load Car_Hybrid),
        BEV: (load Car_BEV)
)
```

Choose query type:

- Fixed the functionality, minimize the resources.
- Fixed the resources, maximize the functionality.
- Given an implementation, evaluate functionality/resources. [UI not implemented]
- Given min functionality and max resources, determine if there is a feasible implementation. [UI not implemented]
- Given min functionality and max resources, find a feasible implementation. [UI not implemented]
- "Solve for X": find the minimal component that makes the co-design problem feasible. [UI not implemented]

Solution of DPs

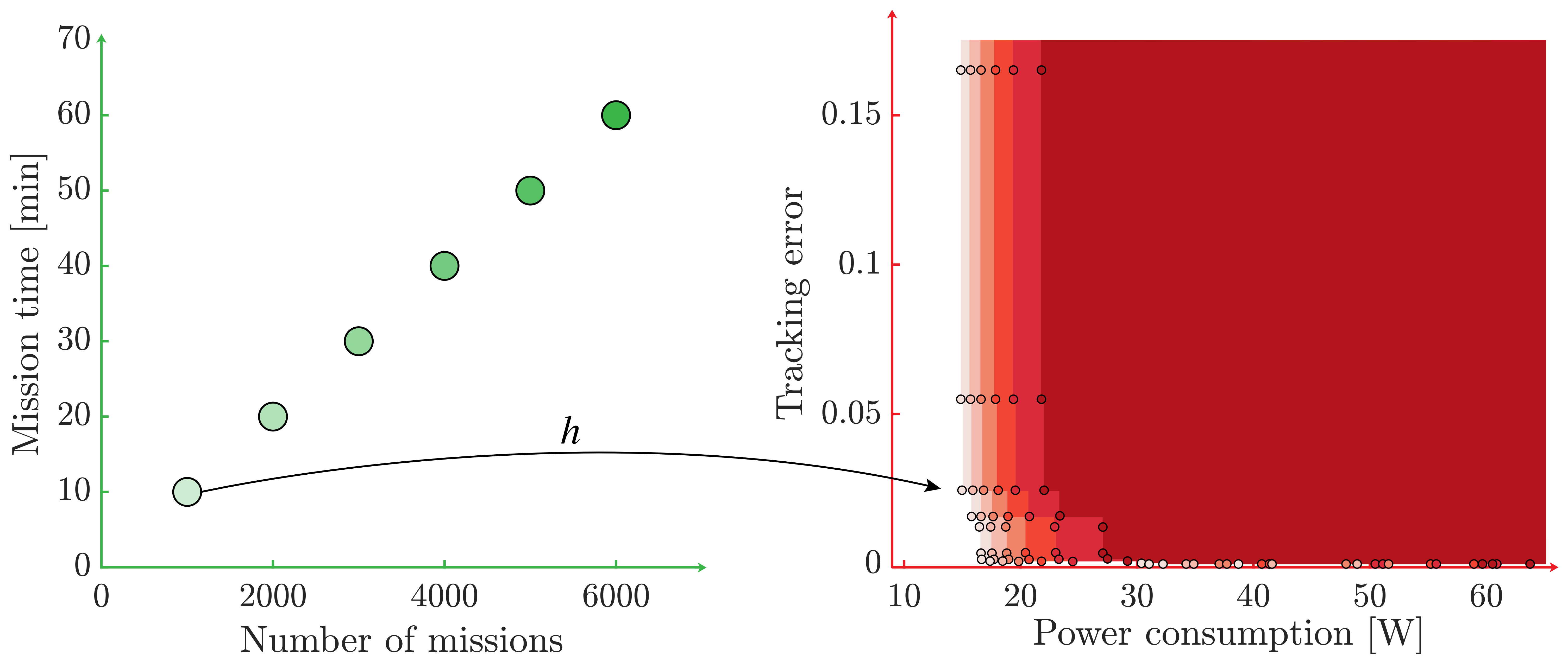




***Fix** functionalities,*
***Minimize** resources*

Details of autonomy,
*both **hardware** and **software***

Monotonicity



Takeaways

- Using co-design, it is easy to **embed** the synthesis of **controllers** into the co-design problem of the whole **autonomous robot**
- *We have shown how to **embed** (variations of) **LQG control** problems into the co-design problem of an **autonomous robot***

- Very **intuitive** modeling approach (no acrobatics like common in optimization theory)
  *The **interpreter** allows one to easily model problems of interest*

- **Rich modeling capabilities:**
  ***Simulation**: Algorithms' performances*
  ***Catalogues**: Sensors, vehicles, computers, algorithms, ...*
  ***Analytical**: LQG closed-form solutions, discomfort models, ...*

- **Compositionality** and **modularity** allow **interdisciplinarity**
  *We did all of it, but technically this could have been possible with different **teams***

- Co-design comes with a **formal language** and an **optimizer**
  *After easily modeling the problem, you can directly solve **queries** of your choice*

- Co-design produces **actionable information** for designers to **reason** about their problems
  *We have shown actionable information for **municipalities**, as well as for **AV developers***

Outlook and references

- Showcase **compositionality** by including the co-design of the **robot** in the co-design of **fleets of robots** (fleet control)

- Generalize this modeling approach to other **control structures** (nonlinear, receding horizon, ...)

- Exploit the framework to synthesize **energy** and **computation-aware** control strategies

- **References:**
  - This paper: *Co-Design of Autonomous Systems: From Hardware Selection to Control Synthesis* (**https://bit.ly/3ixXa5g**)
  - Related work:

    Co-Design of Embodied Intelligence: A Structured Approach (**https://bit.ly/3zq4dTN**)

    Co-Design to Enable User-Friendly tools to Assess the Impact of Future Mobility Solutions (**https://bit.ly/35a5Wyx**)

  - This is a **new** topic, we are making an effort in **evangelization:**

    We are writing a **book,** teaching **classes**, both at ETH and internationally, and organizing **workshops**

https://applied-compositional-thinking.engineering *https://idsc.ethz.ch/research-frazzoli/workshops/compositional-robotics*

http://gioele.science