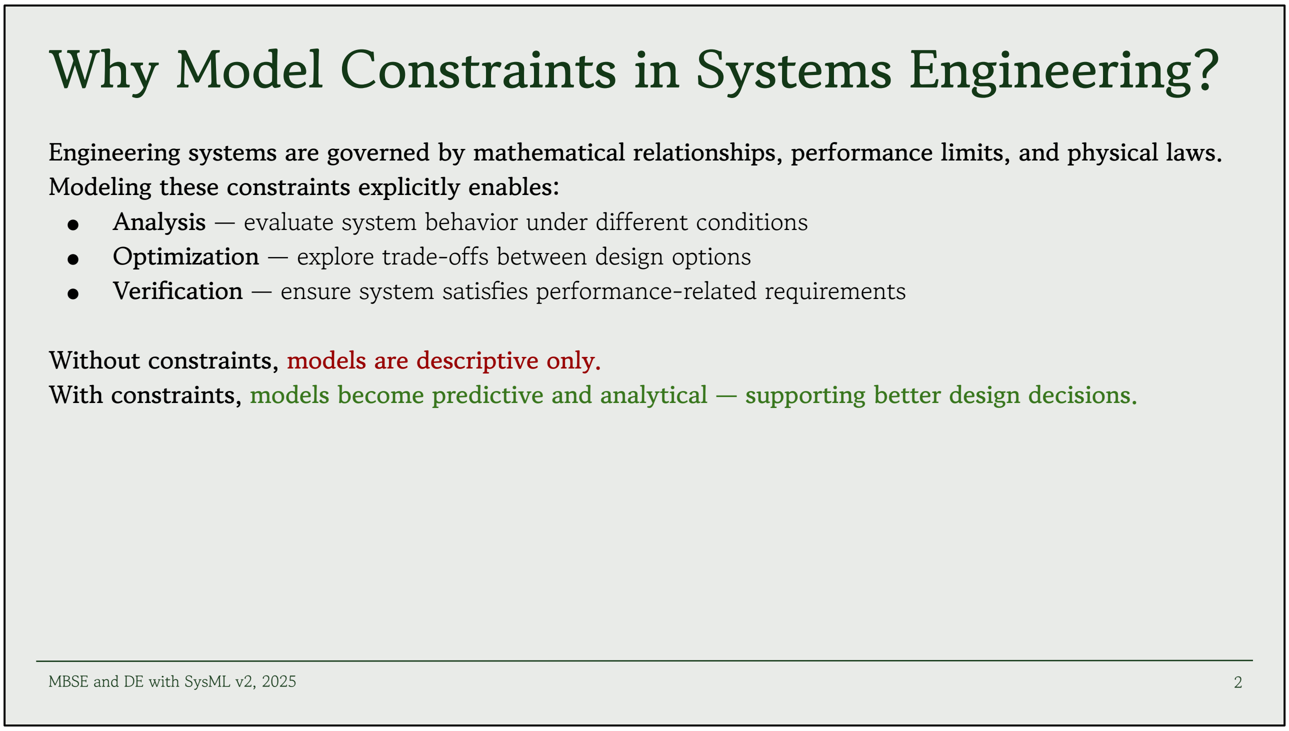
Parametric and Constraint Modeling

Week 10

Define and apply engineering constraints to support analysis, trade studies, and performance modeling in SysML v2.
Welcome to **Week 10** — this week we move into an essential capability of **model-based systems engineering**:
👉 **Parametric and Constraint Modeling**.
So far, we have modeled:
✅ **Structure** — what the system is
✅ **Behavior** — what the system does
✅ **Requirements** — what the system must achieve
This week we add:
👉 **Constraints** — the *mathematical relationships and engineering rules* that govern the system's performance and behavior.
Why is this important?
👉 Without **constraints**, models remain abstract.
👉 With **constraints**, we can **simulate**, **analyze**, and **optimize** our designs.
👉 We will also see how **constraints trace back to requirements** — especially **non-functional requirements** like:
✅ "Endurance ≥ 30 min"
✅ "Payload ≤ 2 kg"
✅ "Max power draw ≤ 300 W"
By the end of this session, you will know how to:
✅ Define constraints using **SysML v2 syntax**
✅ Connect them to your **system structure**
✅ Support **trade studies and performance analysis** in your models.



Why Model Constraints in Systems Engineering?

Engineering systems are governed by mathematical relationships, performance limits, and physical laws. Modeling these constraints explicitly enables:

- **Analysis** — evaluate system behavior under different conditions
- **Optimization** — explore trade-offs between design options
- **Verification** — ensure system satisfies performance-related requirements

Without constraints, models are descriptive only.
With constraints, models become predictive and analytical — supporting better design decisions.

In systems engineering, every system we design is subject to **engineering constraints**:
👉 Physical laws (e.g., energy balance, mass flow)
👉 Performance limits (e.g., max speed, endurance)
👉 Safety margins
👉 Customer performance requirements

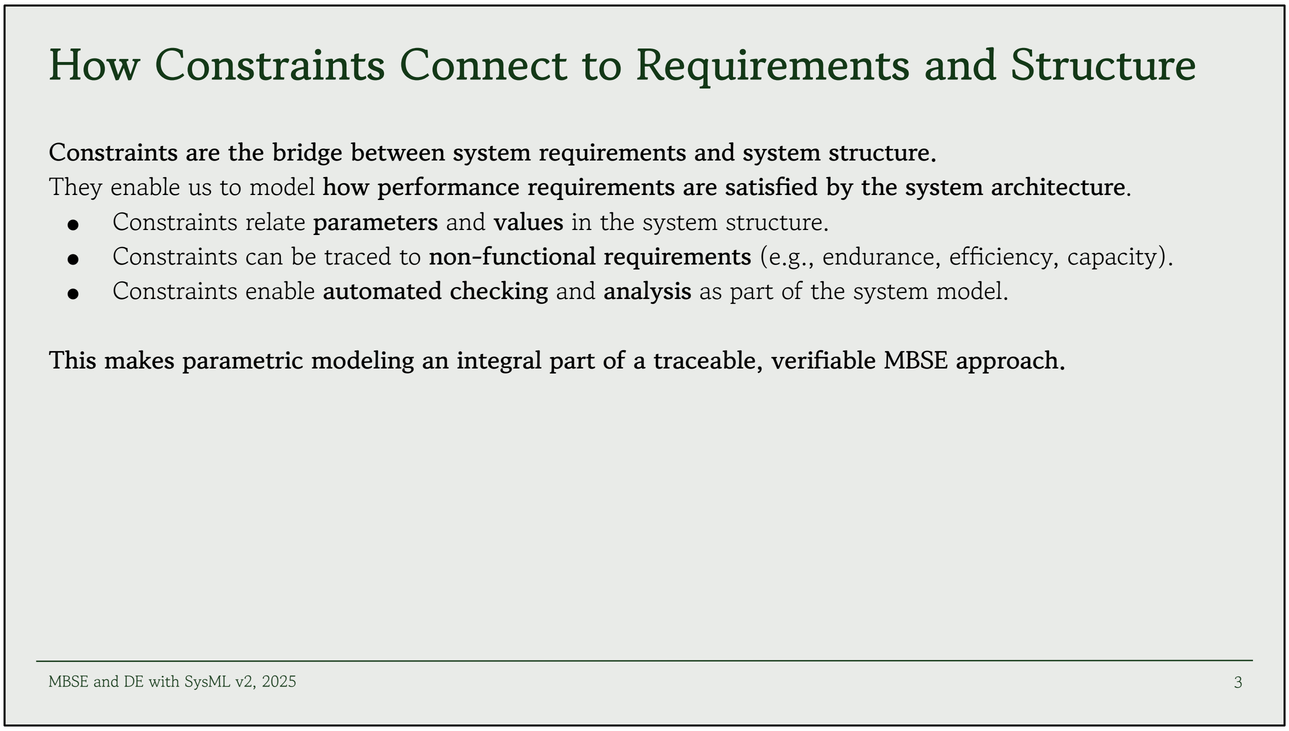
Why do we model these constraints?
First — **Analysis**: We can test how the system behaves when we change conditions:

- ❖ What happens if the battery capacity changes?
- ❖ What if we add payload weight?

Second — **Optimization**: We can run **trade studies** — compare design options based on their impact on system performance.
Third — **Verification**: We can show that the system **satisfies non-functional requirements** — using traceable, testable constraints.

- ❖ Without constraints — the model is just **descriptive** (structure, behavior).
- ❖ With constraints — the model becomes **predictive** — a tool for **decision support** and **validation**.

That is the power of **parametric and constraint modeling** — and why it is a key part of SysML v2.



How Constraints Connect to Requirements and Structure

Constraints are the bridge between system requirements and system structure.
They enable us to model **how performance requirements are satisfied by the system architecture**.

- Constraints relate **parameters** and **values** in the system structure.
- Constraints can be traced to **non-functional requirements** (e.g., endurance, efficiency, capacity).
- Constraints enable **automated checking** and **analysis** as part of the system model.

This makes parametric modeling an integral part of a traceable, verifiable MBSE approach.

Constraints are not an isolated modeling element. They serve an important role in connecting the system's architecture to its requirements.
Every non-functional requirement expresses a performance expectation, and this expectation can usually be described mathematically. For example, a requirement like "The drone shall have a flight endurance of at least 30 minutes" implies a relationship between battery capacity, propulsion efficiency, and power consumption.
By defining this relationship as a constraint in the model, we make the connection between the requirement and the system structure explicit. We can then test and verify that the system satisfies this requirement under different conditions.
In SysML v2, constraints are linked to system parts through parameters and values. This ensures that changes in the design or in input conditions will automatically propagate through the model, keeping the analysis up to date.
This is why constraint modeling is a core part of modern model-based systems engineering: it transforms the model from static documentation into a dynamic tool for verification and decision-making.

What Is a Constraint?

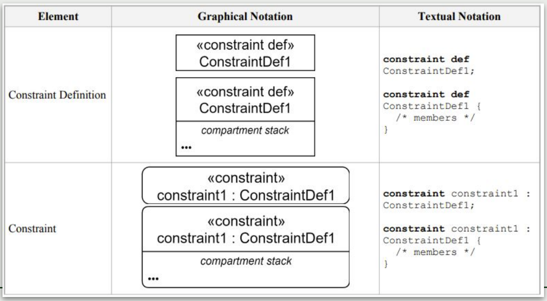
A Constraint defines a rule or relationship that must hold true between parameters or values in the system.

In SysML v2, we model constraints using:

- constraint def — defines the mathematical form of the constraint
- constraint — applies the constraint to specific elements in the model

Constraints express engineering knowledge directly in the model.

They can be used to support analysis, trade studies, and verification against performance requirements.



Element	Graphical Notation	Textual Notation

Constraint Definition	«constraint def» ConstraintDef1
	«constraint def» ConstraintDef1 / *compartment stack* / ...
		`constraint def ConstraintDef1;`
		`constraint def ConstraintDef1 { /* members */ }`

Constraint	«constraint» constraint1 : ConstraintDef1
	«constraint» constraint1 : ConstraintDef1 / *compartment stack* / ...
		`constraint constraint1 : ConstraintDef1;`
		`constraint constraint1 : ConstraintDef1 { /* members */ }`

Example of Constraint in both Graphic Notation and Textual Notation from '2a-OMG_Systems_Modeling_Language.pdf' page 126

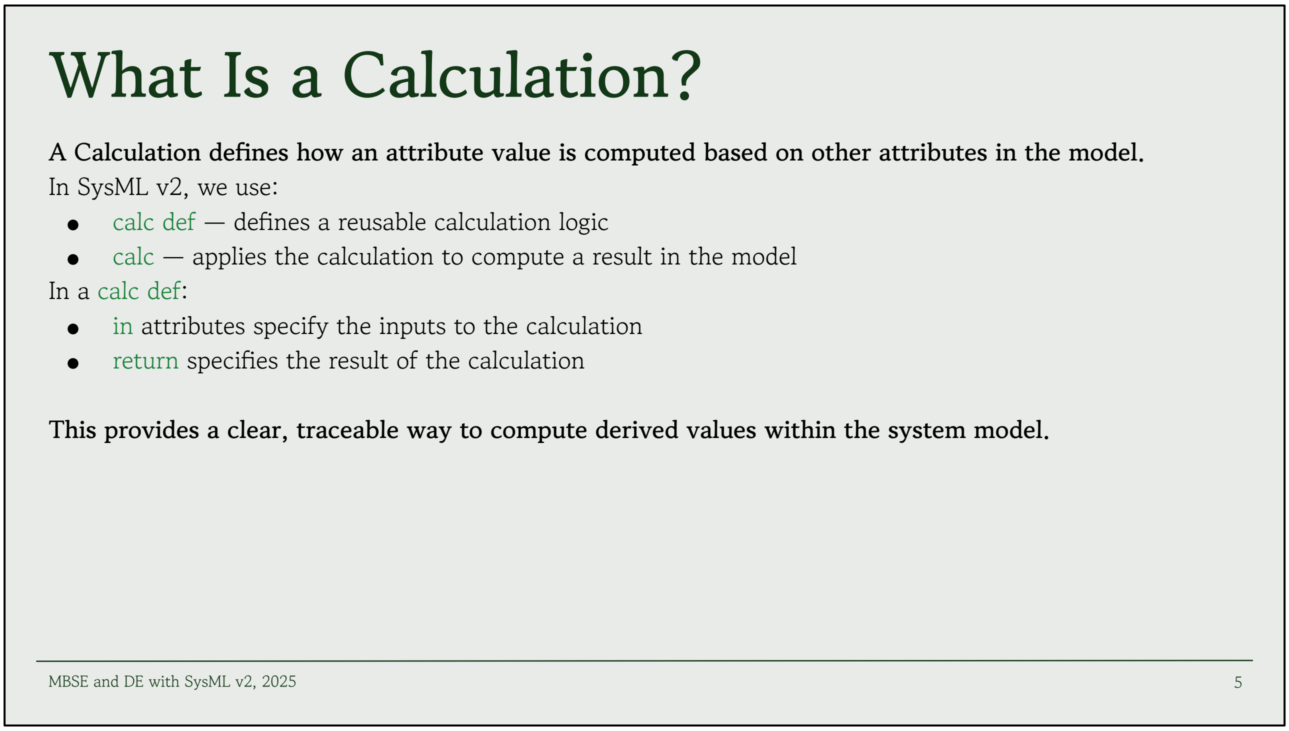
A Constraint captures an essential piece of engineering knowledge: a mathematical rule that defines how certain values in the system are related.

For example, in an electric vehicle, we might model a constraint that relates vehicle mass, aerodynamic drag, and energy consumption to the expected driving range.

In SysML v2, constraints are modeled explicitly. We use constraint def to define the general form of the constraint, similar to defining an equation or formula. Then we use constraint to apply this definition to specific parts of the system, binding the constraint to actual system parameters or values.

By placing the constraint in the model, we make it visible, testable, and reusable. We can automatically check whether the system design satisfies the constraint under different scenarios, and we can trace constraints back to the requirements they help to fulfill.

This makes constraints a powerful tool for ensuring that our system meets its performance goals.



What Is a Calculation?

A Calculation defines how an attribute value is computed based on other attributes in the model.
In SysML v2, we use:

- calc def — defines a reusable calculation logic
- calc — applies the calculation to compute a result in the model

In a calc def:

- in attributes specify the inputs to the calculation
- return specifies the result of the calculation

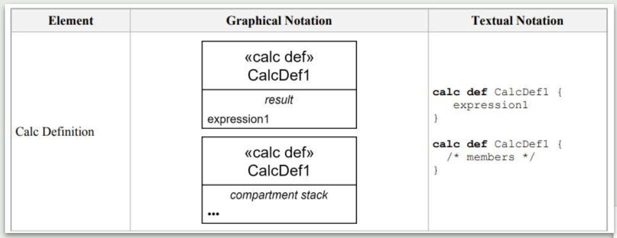
This provides a clear, traceable way to compute derived values within the system model.

In SysML v2, a Calculation defines how to compute an attribute based on other attributes in the system.
Unlike older modeling styles that had separate "parameters," SysML v2 simply operates on **attributes**.
A calc def defines the calculation logic. It specifies:

- in attributes — the inputs to the calculation
- return — the computed result

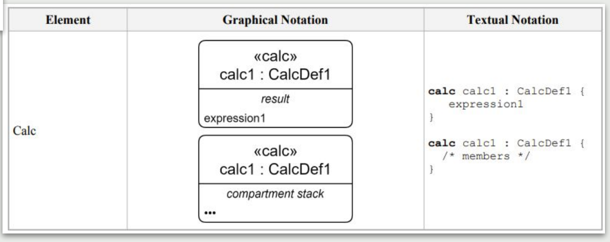
For example, we might define a calc def to compute **vehicle range** based on **battery capacity** and **energy consumption rate**.
When we use this calc def in the model, we create a calc usage and bind its inputs to specific attributes of parts in the system.
This ensures that the calculation is fully integrated with the system structure. Changes in attribute values will automatically propagate through the calculation, keeping derived values up to date.
This approach supports model-driven analysis, simulation, and trade studies — all within SysML v2.

What Is a Calculation?



Element	Graphical Notation	Textual Notation

Calc Definition	«calc def» CalcDef1 / *result* / expression1
	
	«calc def» CalcDef1 / *compartment stack* / •••
		`calc def CalcDef1 {`
		`   expression1`
		`}`
		
		`calc def CalcDef1 {`
		`  /* members */`
		`}`

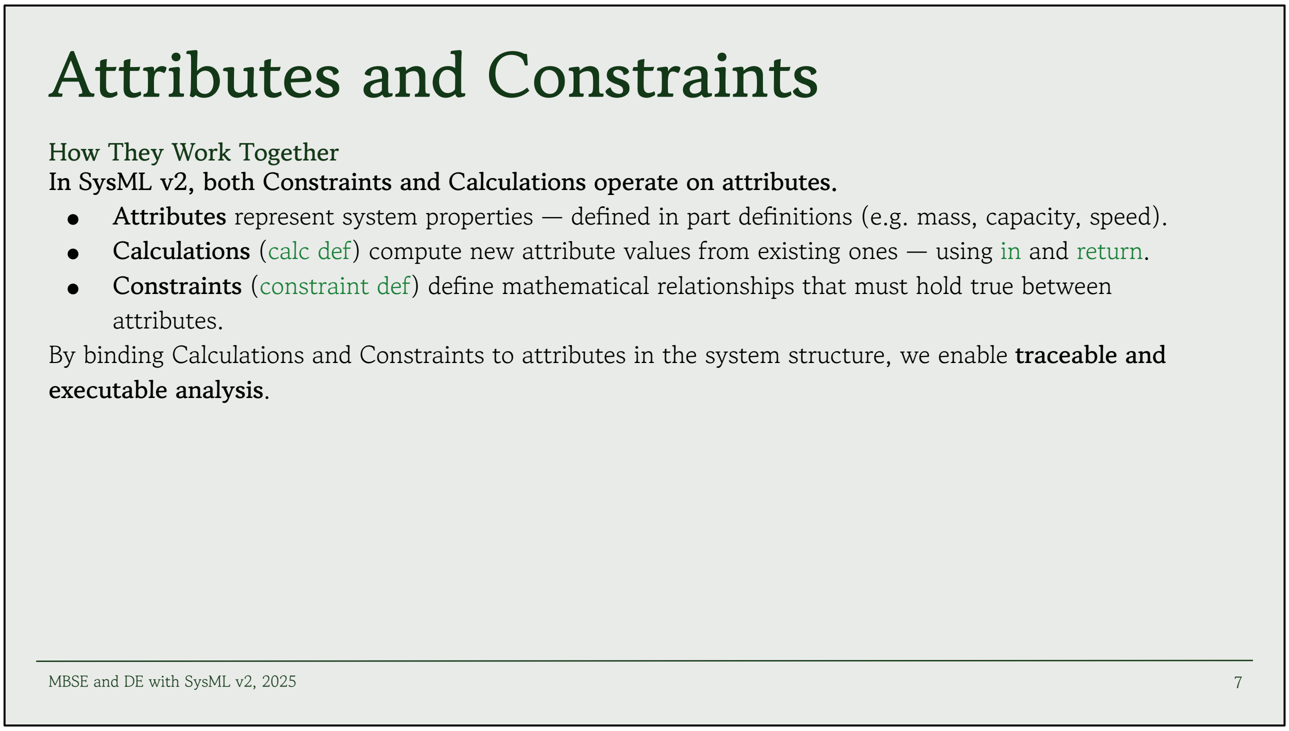


Element	Graphical Notation	Textual Notation

Calc	«calc» calc1 : CalcDef1 / *result* / expression1
	
	«calc» calc1 : CalcDef1 / *compartment stack* / •••
		`calc calc1 : CalcDef1 {`
		`   expression1`
		`}`
		
		`calc calc1 : CalcDef1 {`
		`  /* members */`
		`}`

Example of Calculation in both Graphic Notation and Textual Notation from ‘2a-OMG_Systems_Modeling_Language.pdf’ page 123

Example of Calculation in both Graphic Notation and Textual Notation from ‘2a-OMG_Systems_Modeling_Language.pdf’ page 123



Attributes and Constraints

How They Work Together

In SysML v2, both Constraints and Calculations operate on attributes.

- **Attributes** represent system properties — defined in part definitions (e.g. mass, capacity, speed).
- **Calculations** (calc def) compute new attribute values from existing ones — using in and return.
- **Constraints** (constraint def) define mathematical relationships that must hold true between attributes.

By binding Calculations and Constraints to attributes in the system structure, we enable **traceable and executable analysis**.

In SysML v2, both Calculations and Constraints operate directly on **attributes** of system parts. There is no need to define separate "parameters" or "values." Attributes provide all the needed modeling elements.

Attributes are typically defined in part definitions. For example, an ElectricVehicle part might have:

- batteryCapacity :> ISQ::ElectricCharge
- mass :> ISQ::Mass
- energyConsumptionRate :> ISQ::Power

A calc def defines how to compute an attribute based on others. It uses:

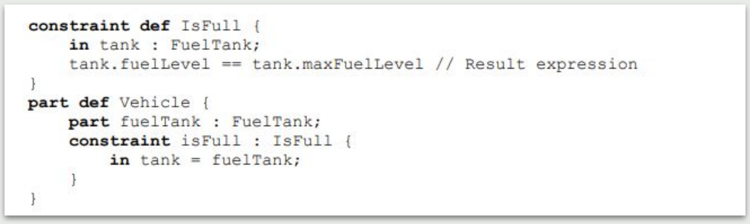
- in attributes — the inputs
- return — the result

A constraint def defines a mathematical relationship between attributes.

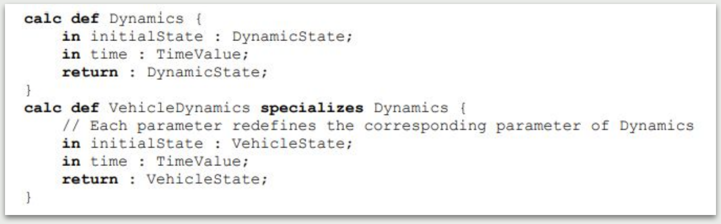
When we apply a Calculation or Constraint in the model, we bind its inputs to specific attributes of parts in the system. This connects the model’s structure to its engineering rules, enabling dynamic and traceable analysis.

This is how we will model relationships like **range vs. battery capacity vs. mass** in our Vehicle example — and later in your Drone systems.

Example Calculation and Constraint — Textual Notation



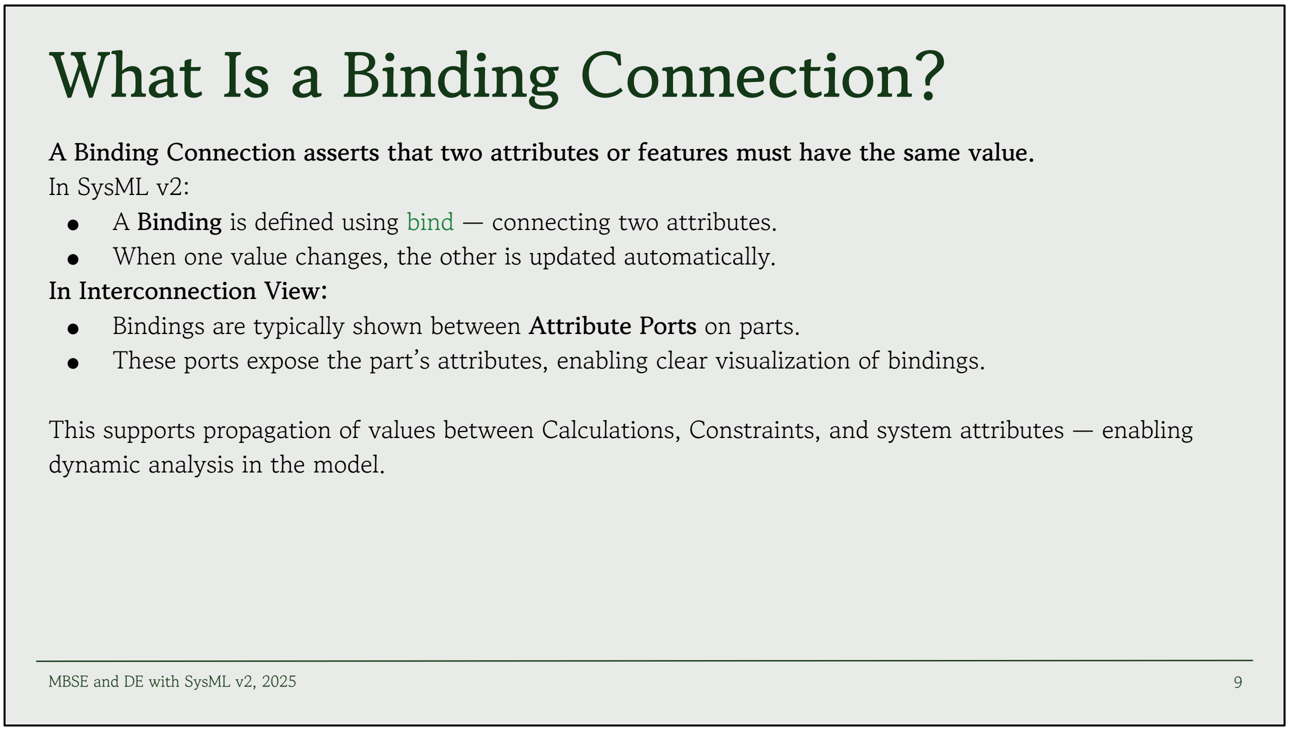
```
constraint def IsFull {
    in tank : FuelTank;
    tank.fuelLevel == tank.maxFuelLevel // Result expression
}
part def Vehicle {
    part fuelTank : FuelTank;
    constraint isFull : IsFull {
        in tank = fuelTank;
    }
}
```



```
calc def Dynamics {
    in initialState : DynamicState;
    in time : TimeValue;
    return : DynamicState;
}
calc def VehicleDynamics specializes Dynamics {
    // Each parameter redefines the corresponding parameter of Dynamics
    in initialState : VehicleState;
    in time : TimeValue;
    return : VehicleState;
}
```

Example from ‘2a-OMG_Systems_Modeling_Language.pdf’ page 124 and page 128

Example from ‘2a-OMG_Systems_Modeling_Language.pdf’ page 124 and page 128



What Is a Binding Connection?

A Binding Connection asserts that two attributes or features must have the same value.
In SysML v2:

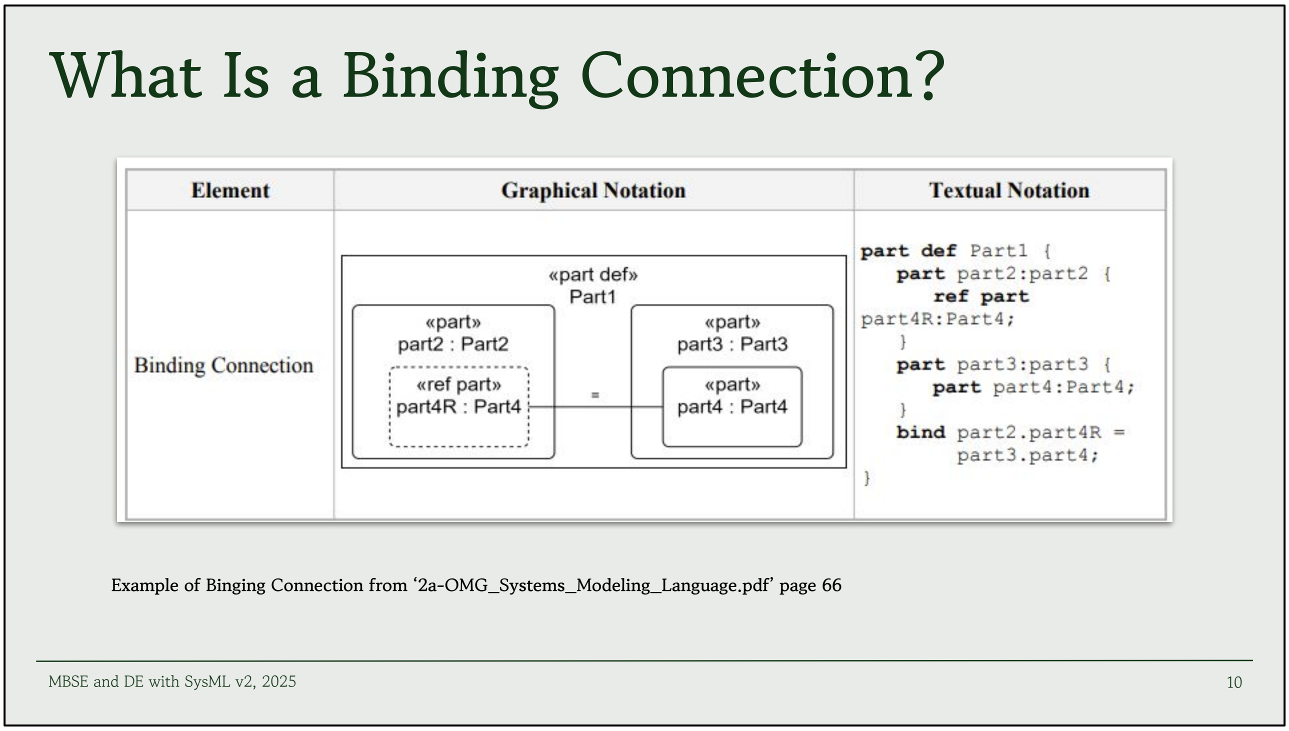
- A **Binding** is defined using bind — connecting two attributes.
- When one value changes, the other is updated automatically.

In Interconnection View:

- Bindings are typically shown between **Attribute Ports** on parts.
- These ports expose the part's attributes, enabling clear visualization of bindings.

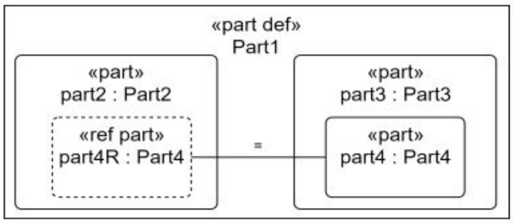
This supports propagation of values between Calculations, Constraints, and system attributes — enabling dynamic analysis in the model.

A **Binding Connection** is a core concept in SysML v2. It ensures that two attributes or features of the system have the same value — they are equal.
In the textual model, we define a binding using bind.
For example: *bind vehicle.range = computeRange.range;*
This says that the result of the computeRange calculation is bound to the range attribute of the vehicle part.
In **Interconnection View**, Bindings are typically shown between **Attribute Ports** of parts. These ports expose the underlying attributes of the part, allowing us to visualize how values are connected and propagated.
This is fully aligned with the SysML v2 specification. The Binding operates on attributes — the ports simply provide a convenient interface for visualization and for structuring the connections clearly in the diagram.
When we model **Calculations and Constraints**, the resulting and participating attributes can be **bound** using these connections — ensuring that changes propagate correctly through the system.
This enables dynamic, traceable analysis — and supports practices such as trade studies and sensitivity analysis.



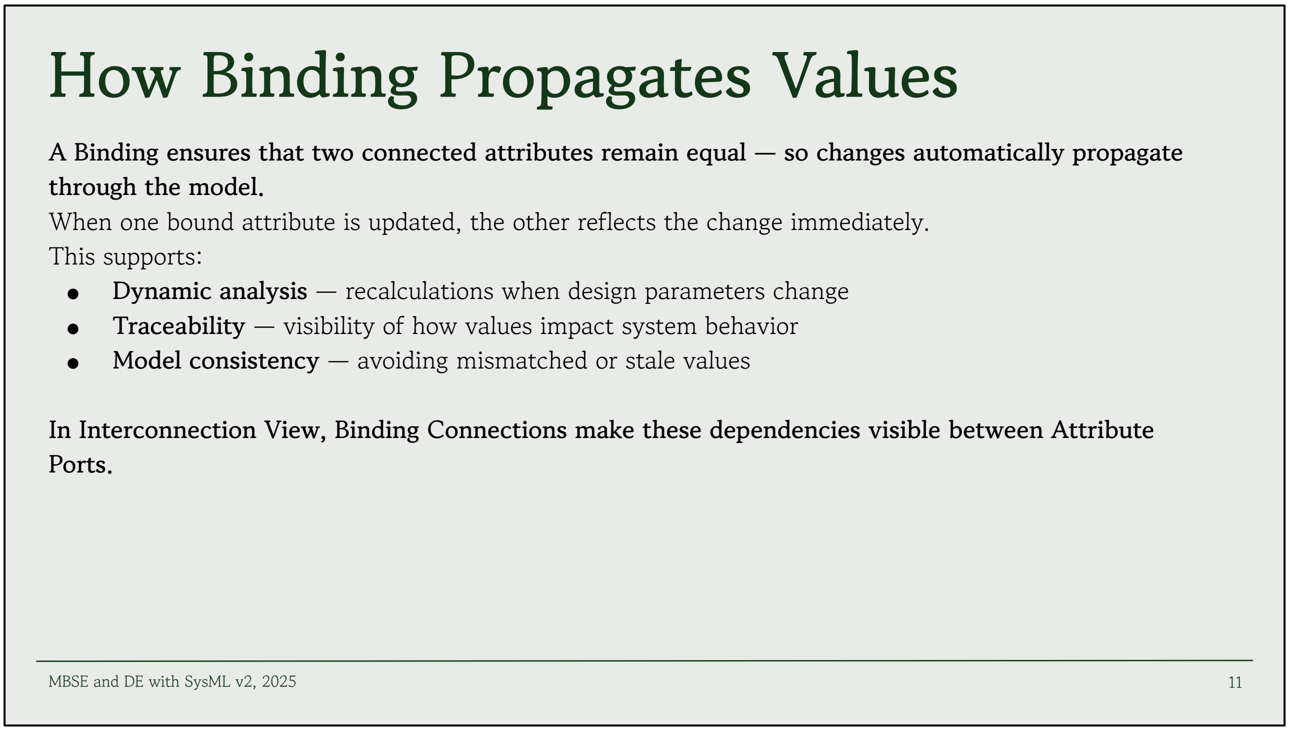
What Is a Binding Connection?

Element	Graphical Notation	Textual Notation

Binding Connection	 «part def»
	Part1
	«part»
	part2 : Part2
	«ref part»
	part4R : Part4
	=
	«part»
	part3 : Part3
	«part»
	part4 : Part4
		`part def Part1 {`
		`   part part2:part2 {`
		`      ref part`
		`part4R:Part4;`
		`   }`
		`   part part3:part3 {`
		`      part part4:Part4;`
		`   }`
		`   bind part2.part4R =`
		`        part3.part4;`
		`}`

Example of Binging Connection from ‘2a-OMG_Systems_Modeling_Language.pdf’ page 66

Example of Binging Connection from ‘2a-OMG_Systems_Modeling_Language.pdf’ page 66



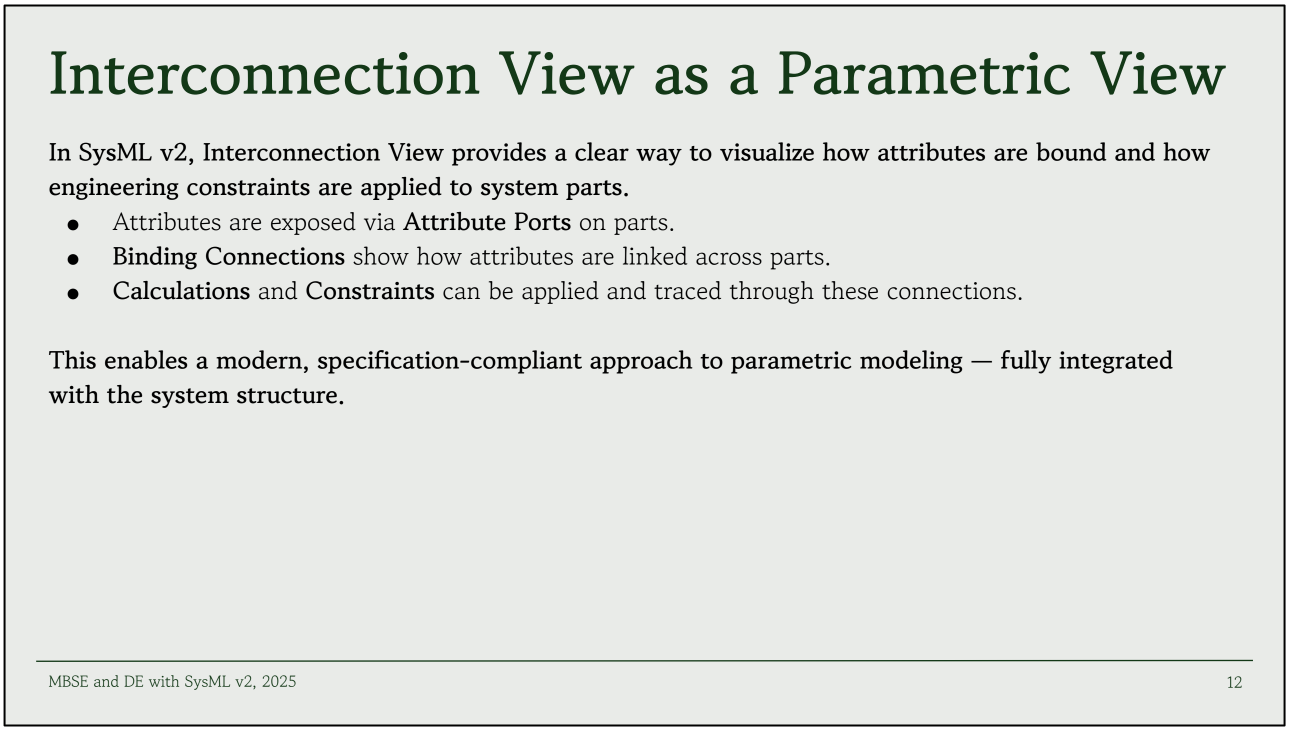
How Binding Propagates Values

A Binding ensures that two connected attributes remain equal — so changes automatically propagate through the model.
When one bound attribute is updated, the other reflects the change immediately.
This supports:

- **Dynamic analysis** — recalculations when design parameters change
- **Traceability** — visibility of how values impact system behavior
- **Model consistency** — avoiding mismatched or stale values

In Interconnection View, Binding Connections make these dependencies visible between Attribute Ports.

The key power of a **Binding Connection** is that it causes value propagation through the model. When two attributes are bound, any change in one will automatically update the other. This is true whether the change comes from a Calculation, a Constraint result, or a manual update to the model.
This enables dynamic analysis. For example, if we change the battery capacity of a vehicle part, and the range is computed via a Calculation, then a Binding will propagate the new value to the vehicle's range attribute — and to any Constraint that checks the range against a requirement.
Binding also provides traceability. In Interconnection View, we can see how attributes are connected and which values depend on each other. This helps us understand the system's behavior and reasoning — supporting model-based verification and trade studies.
Finally, Binding ensures consistency across the model. Bound attributes cannot drift apart — the model always maintains correct relationships as specified by the engineer.
This is one of the key enablers for making SysML v2 a dynamic and analytical modeling language — not just descriptive.



Interconnection View as a Parametric View

In SysML v2, Interconnection View provides a clear way to visualize how attributes are bound and how engineering constraints are applied to system parts.

- Attributes are exposed via **Attribute Ports** on parts.
- **Binding Connections** show how attributes are linked across parts.
- **Calculations** and **Constraints** can be applied and traced through these connections.

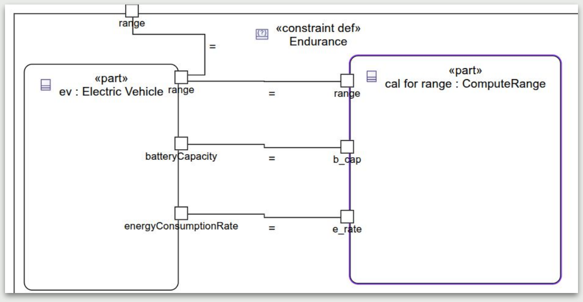
This enables a modern, specification-compliant approach to parametric modeling — fully integrated with the system structure.

In older modeling styles, parametric diagrams were often a separate view with dedicated stereotypes.
In SysML v2, we achieve the same goals — but in a much more integrated and modern way.
The **Interconnection View** serves as a parametric view because it shows how **attributes** of parts are connected via **Binding Connections**.
Each part exposes attributes through **Attribute Ports**. We can then create Binding Connections between these ports to represent the equalities and relationships that must hold between different parts of the system.
This is also how we connect Calculations and Constraints to the system structure. The result of a Calculation can be bound to an attribute of a part. Attributes participating in a Constraint can be bound to relevant system attributes as well.
This approach is fully compliant with the SysML v2 specification. Binding operates on attributes and features — and the Interconnection View provides a clear and traceable visualization of how these relationships propagate through the model.
In this way, we can model and analyze complex engineering relationships — such as how performance depends on configuration — using modern SysML v2 practices.

Example: Binding Vehicle Attributes in Interconnection View

Scenario: Modeling how a Vehicle's **computed range** is linked to system attributes and constraints.
Elements involved:

- vehicle.batteryCapacity and vehicle.energyConsumptionRate — input attributes
- calc computeRange — calculates vehicle.range
- constraint enduranceConstraint — checks if vehicle.range meets required endurance
- **Binding Connections** link all these attributes



In Interconnection View:

- Attribute Ports expose these attributes on the Vehicle part and related elements
- Binding Connections show how values propagate across the model

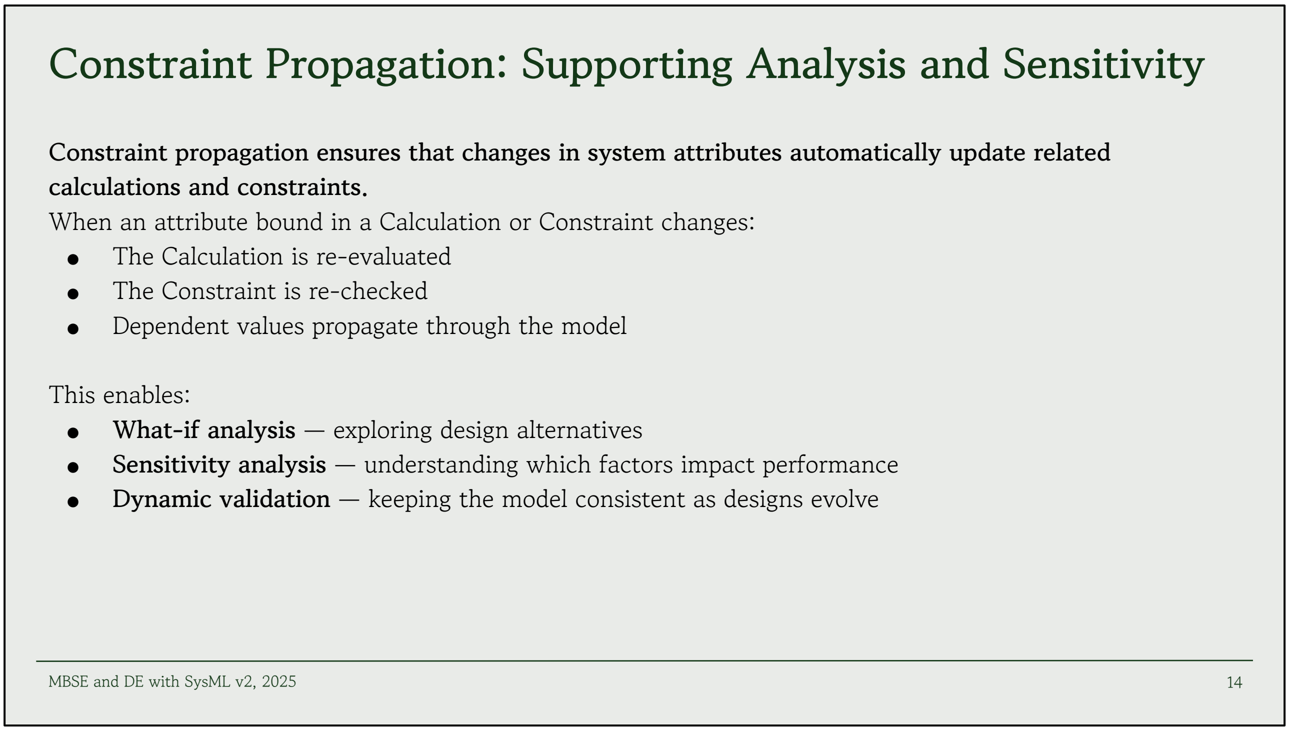
Let's look at a concrete example of how this works in an Interconnection View.
We want to model how an **Electric Vehicle range** depends on its **battery capacity** and **energy consumption rate** — and how this supports verification against an endurance requirement.
The relevant attributes are:

- vehicle.batteryCapacity
- vehicle.energyConsumptionRate
- vehicle.range

We have a calc computeRange, which computes the range based on the two input attributes.
We also have a constraint enduranceConstraint, which checks that the computed range meets or exceeds the required endurance.
In the Interconnection View, we expose these attributes via **Attribute Ports**.
We create **Binding Connections**:

- Between the input attributes and the inputs to the Calculation
- Between the Calculation result and the vehicle.range attribute
- Between vehicle.range and the attribute checked by the Constraint

This makes it immediately clear how values propagate through the model — and which attributes affect system performance.
This is a modern and specification-compliant way to model **parametric relationships** in SysML v2 — fully integrated with the system structure.



Constraint Propagation: Supporting Analysis and Sensitivity

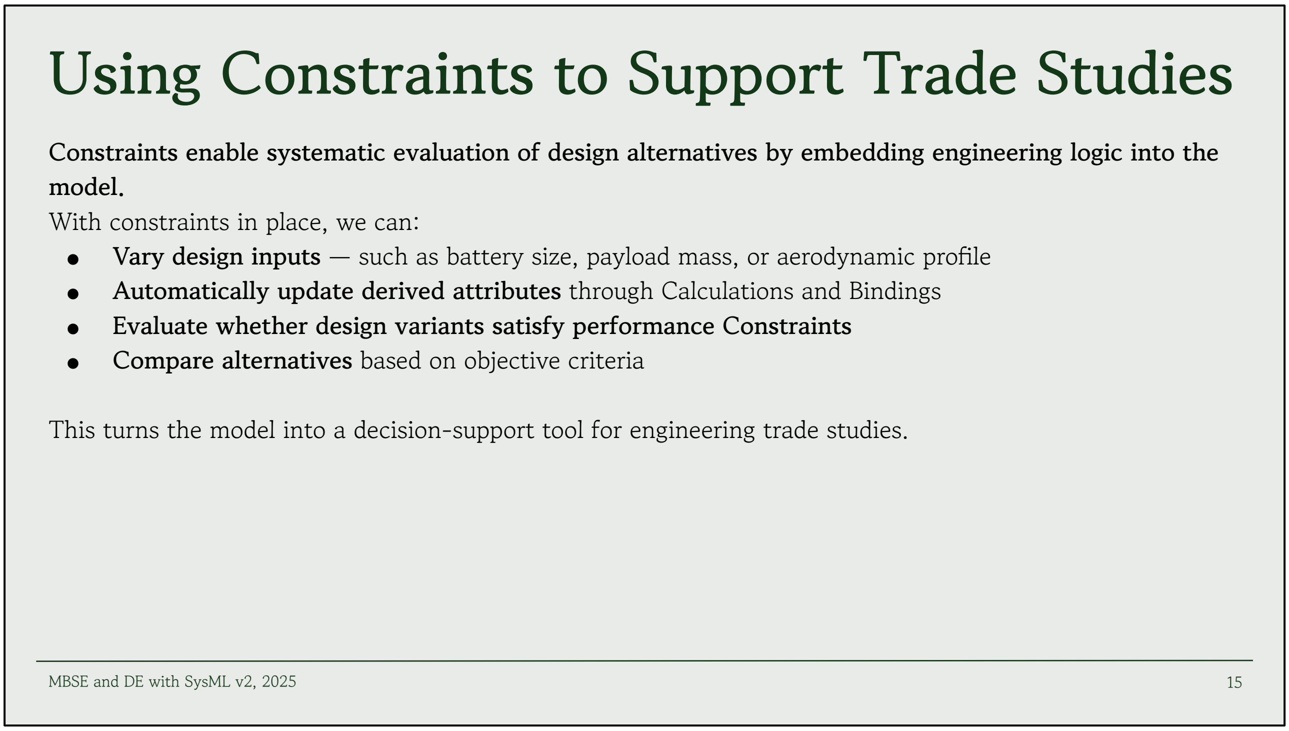
Constraint propagation ensures that changes in system attributes automatically update related calculations and constraints.
When an attribute bound in a Calculation or Constraint changes:

- The Calculation is re-evaluated
- The Constraint is re-checked
- Dependent values propagate through the model

This enables:

- **What-if analysis** — exploring design alternatives
- **Sensitivity analysis** — understanding which factors impact performance
- **Dynamic validation** — keeping the model consistent as designs evolve

One of the key advantages of modeling Constraints and Calculations in SysML v2 is that they support **automatic propagation of changes**.
When we create a model where attributes are connected through Binding Connections, Calculations, and Constraints, we create a network of relationships that reflect the engineering reality of the system.
When one of the connected attributes changes — such as battery capacity — the related Calculations are automatically re-evaluated, and the related Constraints are automatically re-checked.
This is extremely powerful because it enables **What-if analysis**. We can quickly test what happens if we increase battery capacity or reduce vehicle mass. The model will immediately show how these changes affect derived attributes such as range — and whether the system still meets its requirements.
It also enables **Sensitivity analysis**. We can determine which attributes have the greatest impact on system performance, guiding our engineering trade-offs.
Finally, this propagation ensures **dynamic validation**. As the design evolves, the model continuously reflects whether the system remains compliant with its performance goals.
This is one of the core reasons we invest in model-based systems engineering — to enable this level of traceable and dynamic analysis directly in the model.



Using Constraints to Support Trade Studies

Constraints enable systematic evaluation of design alternatives by embedding engineering logic into the model.

With constraints in place, we can:

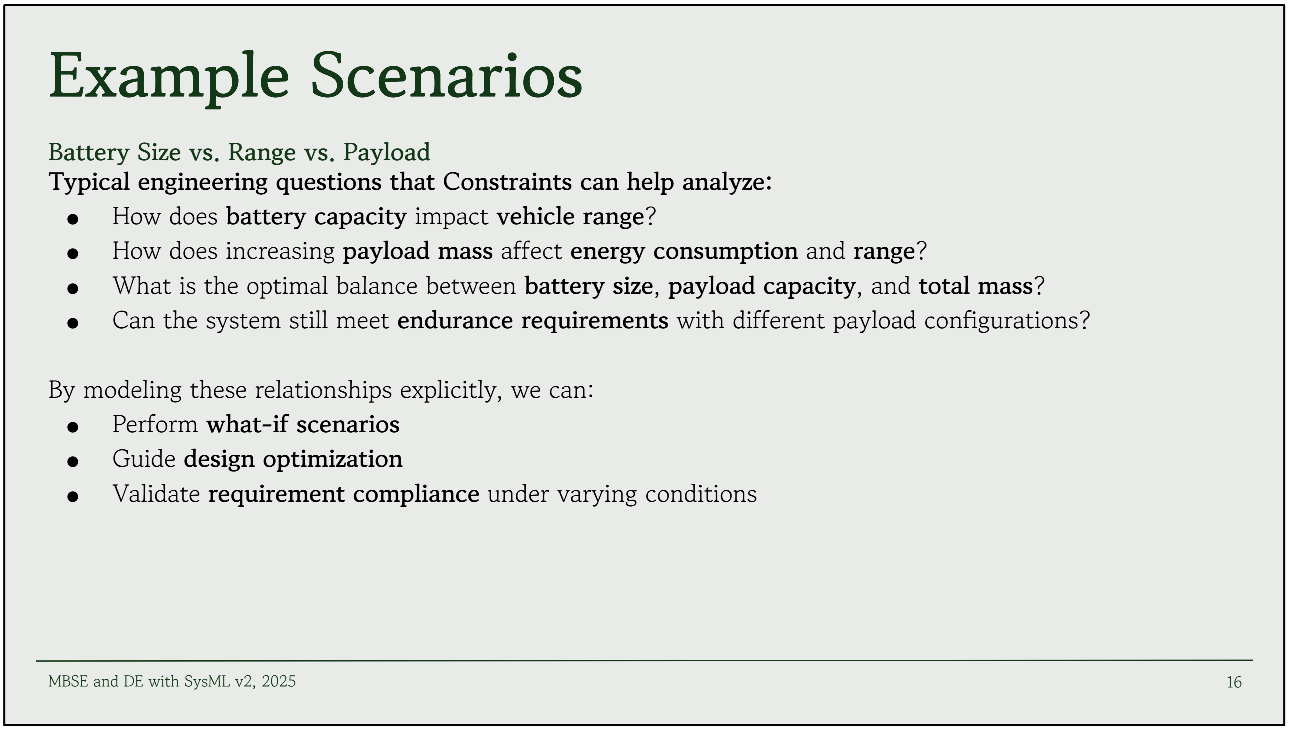
- **Vary design inputs** — such as battery size, payload mass, or aerodynamic profile
- **Automatically update derived attributes** through Calculations and Bindings
- **Evaluate whether design variants satisfy performance Constraints**
- **Compare alternatives** based on objective criteria

This turns the model into a decision-support tool for engineering trade studies.

One of the most valuable uses of parametric and constraint modeling is to support **engineering trade studies**.
A trade study is essentially an exploration of how different design choices affect system performance and compliance with requirements.
When we model Constraints and Calculations correctly — and bind them to system attributes — the model itself becomes an analytical tool.
We can systematically vary design inputs — such as battery capacity, vehicle mass, or propulsion efficiency — and observe how these changes propagate through the model.
Derived attributes such as range or endurance are automatically updated through Calculations.
Constraints are automatically checked to ensure compliance.
This allows us to compare different design alternatives objectively:

- Which battery size provides the best trade-off between weight and range?
- How does increased payload affect endurance?
- Can we meet endurance requirements with a smaller, lighter battery?

By capturing the engineering logic directly in the model, SysML v2 turns our models into powerful **decision-support tools** — enabling faster and more informed design choices.



Example Scenarios

Battery Size vs. Range vs. Payload

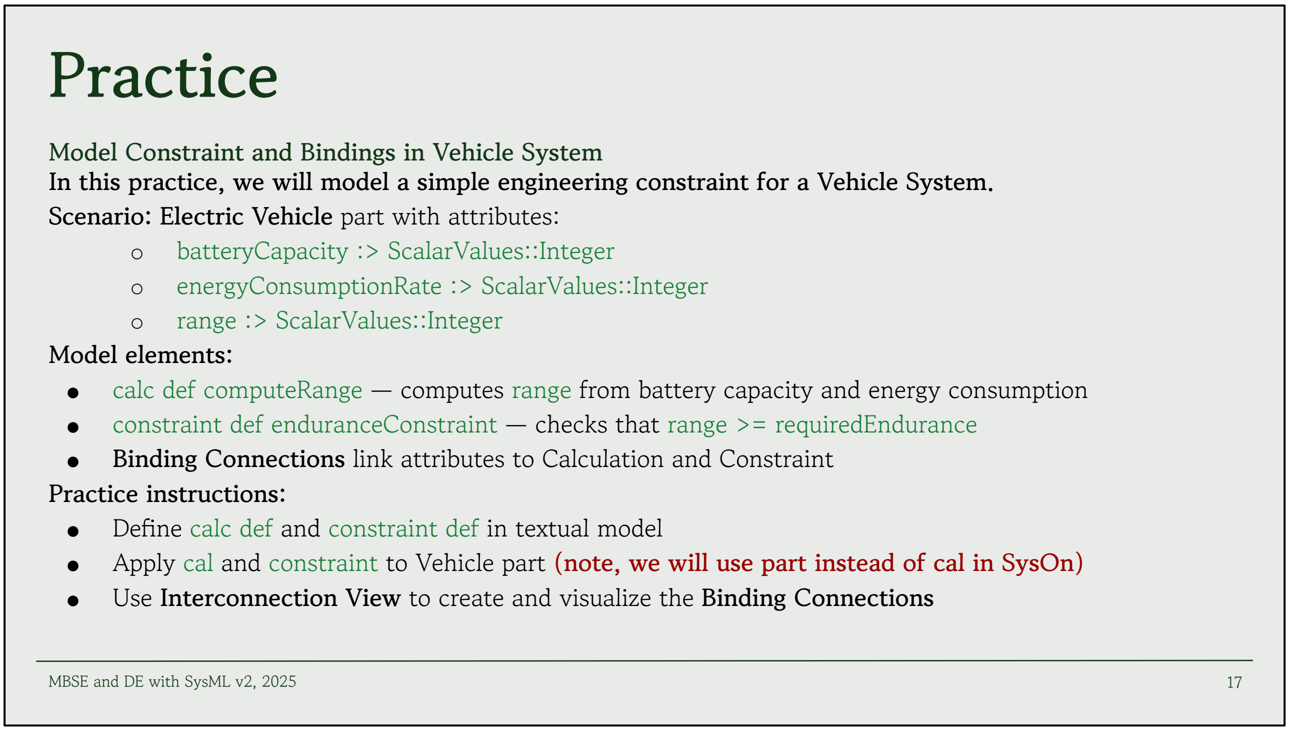
Typical engineering questions that Constraints can help analyze:

- How does **battery capacity** impact **vehicle range**?
- How does increasing **payload mass** affect **energy consumption** and **range**?
- What is the optimal balance between **battery size**, **payload capacity**, and **total mass**?
- Can the system still meet **endurance requirements** with different payload configurations?

By modeling these relationships explicitly, we can:

- Perform **what-if scenarios**
- Guide **design optimization**
- Validate **requirement compliance** under varying conditions

Here are typical engineering questions that we can answer through constraint modeling in a vehicle system:
First, how does **battery capacity** affect the range of the vehicle? We can model this relationship through a Calculation, binding battery capacity and energy consumption rate to compute range.
Second, how does adding **payload mass** impact energy consumption and thus reduce range? We can model this by including payload mass in the calculation of total vehicle mass, which influences power consumption.
Third, what is the **optimal trade-off** between battery size, payload capacity, and total system mass? Larger batteries add mass, which can reduce efficiency, but increase stored energy. Constraints help us explore this balance.
Finally, we can check whether the vehicle meets **endurance requirements** under different configurations. For example, can it still achieve 30 minutes of flight time with a heavier payload?
By embedding these relationships and constraints in the model, we can run systematic **what-if analyses**. We can also guide **design optimization**, helping engineers choose the best combination of parameters.
And crucially, we can continuously validate **requirement compliance** — even as the design evolves or operating conditions change.



Practice

Model Constraint and Bindings in Vehicle System

In this practice, we will model a simple engineering constraint for a Vehicle System.

Scenario: Electric Vehicle part with attributes:

- batteryCapacity :> ScalarValues::Integer
- energyConsumptionRate :> ScalarValues::Integer
- range :> ScalarValues::Integer

Model elements:

- calc def computeRange — computes range from battery capacity and energy consumption
- constraint def enduranceConstraint — checks that range >= requiredEndurance
- **Binding Connections** link attributes to Calculation and Constraint

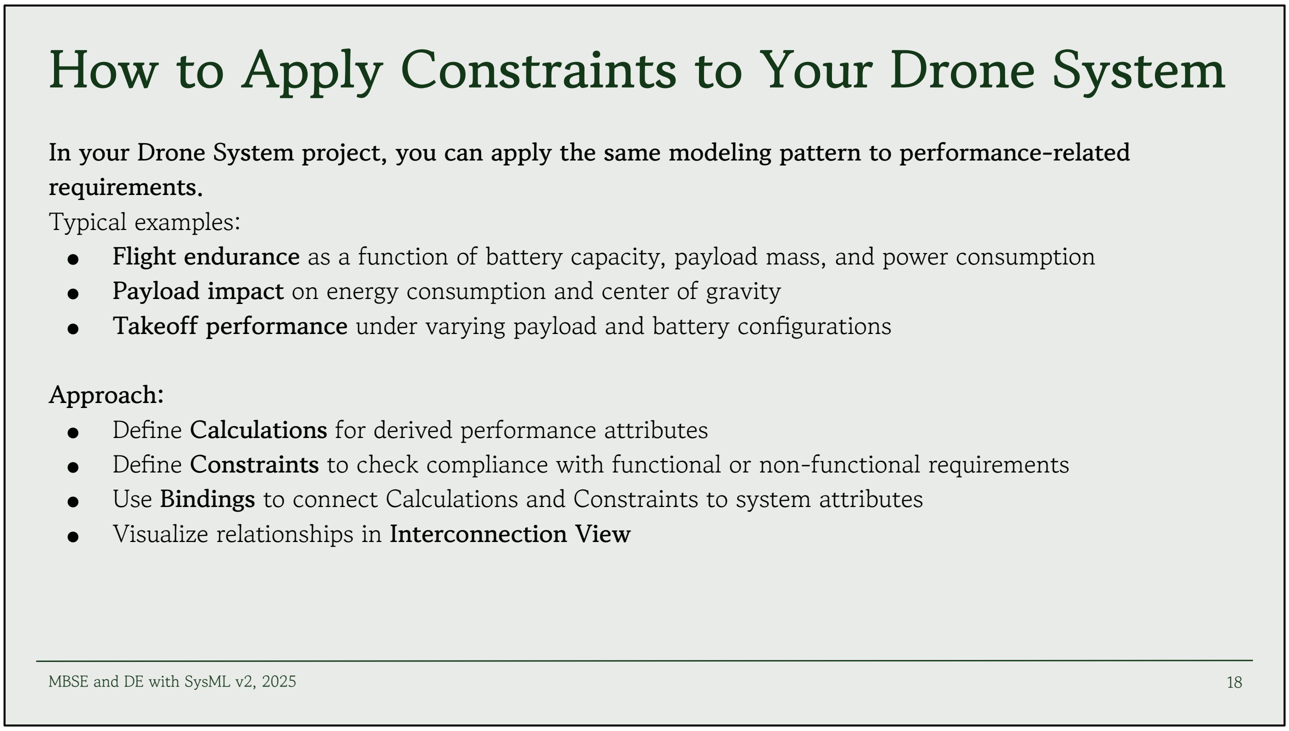
Practice instructions:

- Define calc def and constraint def in textual model
- Apply cal and constraint to Vehicle part **(note, we will use part instead of cal in SysOn)**
- Use **Interconnection View** to create and visualize the **Binding Connections**

Now let us apply what we have learned.
In this practice, we will model a simple but realistic engineering constraint for a Vehicle System — one that could apply equally well to many other domains, including the Drone systems you are building.
Our Electric Vehicle part has three key attributes:

- batteryCapacity — how much energy is stored
- energyConsumptionRate — how quickly energy is used
- range — the resulting driving or flight endurance

First, you will define a calc def computeRange. It will take battery capacity and energy consumption rate as inputs, and return the computed range.
Then you will define a constraint def enduranceConstraint. It will assert that the computed range must meet or exceed a required endurance value.
You will apply both the Calculation and the Constraint to the Vehicle part.
Finally, in the **Interconnection View**, you will create **Binding Connections** between the relevant Attribute Ports — making the propagation of values clear and traceable.
This will give you a complete example of how to model Constraints and Bindings in SysML v2 — fully integrated with system structure.
The same pattern can later be applied to your Drone system for modeling endurance, payload impact, and similar performance factors.



How to Apply Constraints to Your Drone System

In your Drone System project, you can apply the same modeling pattern to performance-related requirements.

Typical examples:

- **Flight endurance** as a function of battery capacity, payload mass, and power consumption
- **Payload impact** on energy consumption and center of gravity
- **Takeoff performance** under varying payload and battery configurations

Approach:

- Define **Calculations** for derived performance attributes
- Define **Constraints** to check compliance with functional or non-functional requirements
- Use **Bindings** to connect Calculations and Constraints to system attributes
- Visualize relationships in **Interconnection View**

Now that you have practiced this approach with a simple Vehicle System, you are ready to apply it to your own Drone System project.

The same pattern applies. In your Drone, you will have requirements related to performance — especially **flight endurance** and **payload capacity**.

For example, you can model how endurance depends on battery capacity, payload mass, and power consumption. You can also model how payload mass affects center of gravity, or how different battery configurations impact takeoff performance.

The steps are exactly the same as in today's practice:

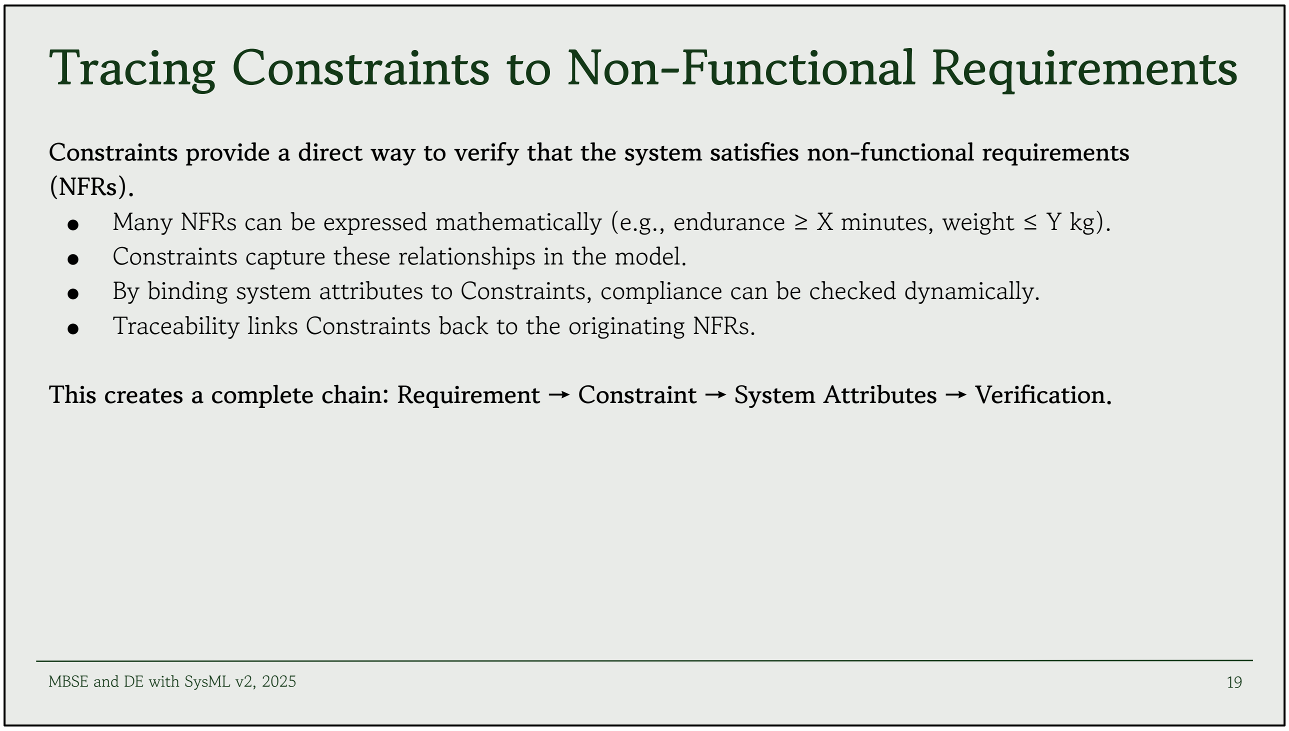
First, define **Calculations** to compute the derived attributes you care about — such as endurance.

Next, define **Constraints** that check whether these computed attributes satisfy your project's functional or non-functional requirements.

Use **Bindings** to connect your Calculations and Constraints to the actual attributes of your Drone system parts.

Finally, use **Interconnection View** to visualize and trace these relationships — making your analysis visible and explainable.

This is how we bring real engineering analysis into the model — turning it into a decision-support tool for design.



Tracing Constraints to Non-Functional Requirements

Constraints provide a direct way to verify that the system satisfies non-functional requirements (NFRs).

- Many NFRs can be expressed mathematically (e.g., endurance ≥ X minutes, weight ≤ Y kg).
- Constraints capture these relationships in the model.
- By binding system attributes to Constraints, compliance can be checked dynamically.
- Traceability links Constraints back to the originating NFRs.

This creates a complete chain: Requirement → Constraint → System Attributes → Verification.

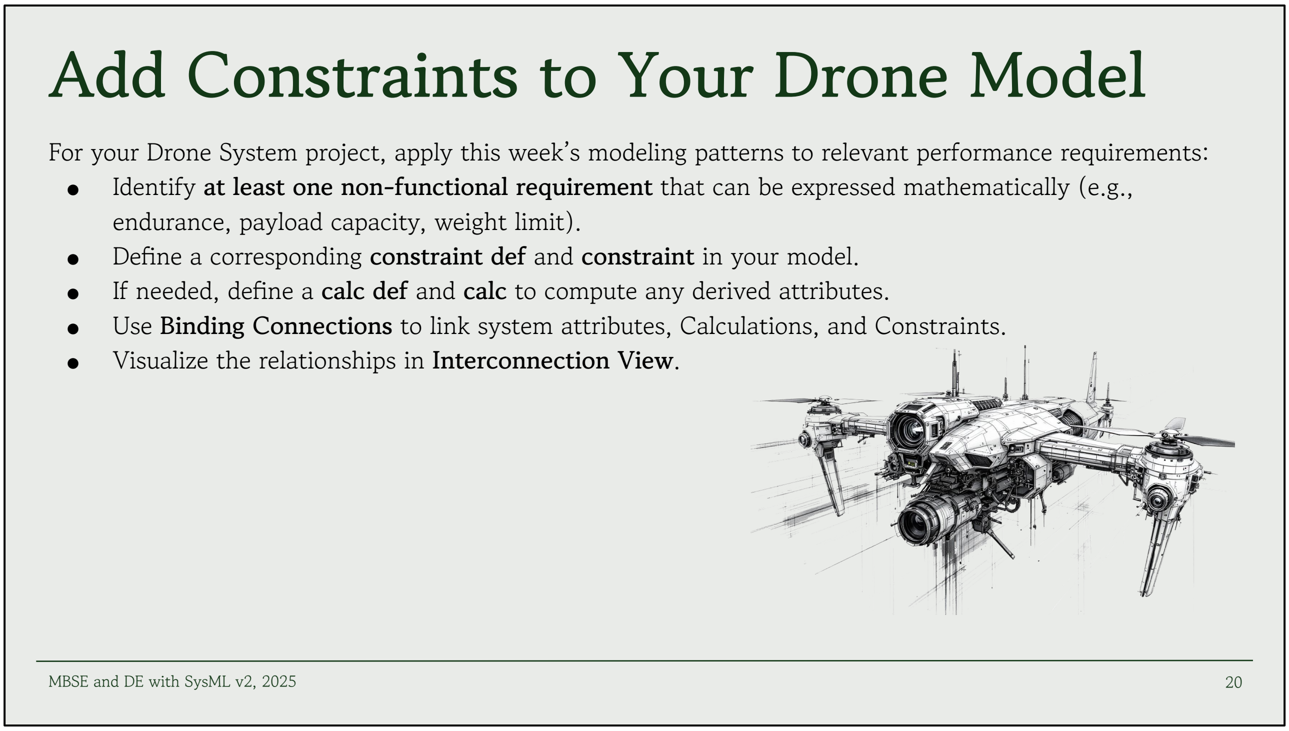
Constraints are particularly valuable for ensuring that our system satisfies **non-functional requirements** — the kinds of requirements that specify performance, capacity, limits, and margins.
Many NFRs can be expressed mathematically. For example:

- The Drone shall have a flight endurance of at least 30 minutes.
- The Drone shall carry a payload of up to 2 kg.
- The total system mass shall not exceed 5 kg.

Constraints allow us to model these requirements directly.
We define a Constraint that expresses the required relationship.
We bind that Constraint to the relevant system attributes — which may be derived via Calculations.
This creates a **traceable verification path**:

- The original NFR is modeled in the Requirements model.
- The Constraint is linked to that Requirement.
- The system attributes are linked to the Constraint.
- As system values change, the model automatically checks whether the Constraint is still satisfied.

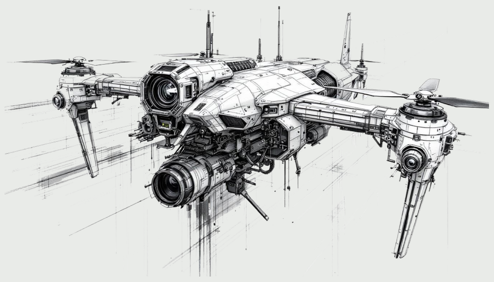
This is a core principle of model-based systems engineering: turning the model into a tool for **continuous verification**, not just documentation.



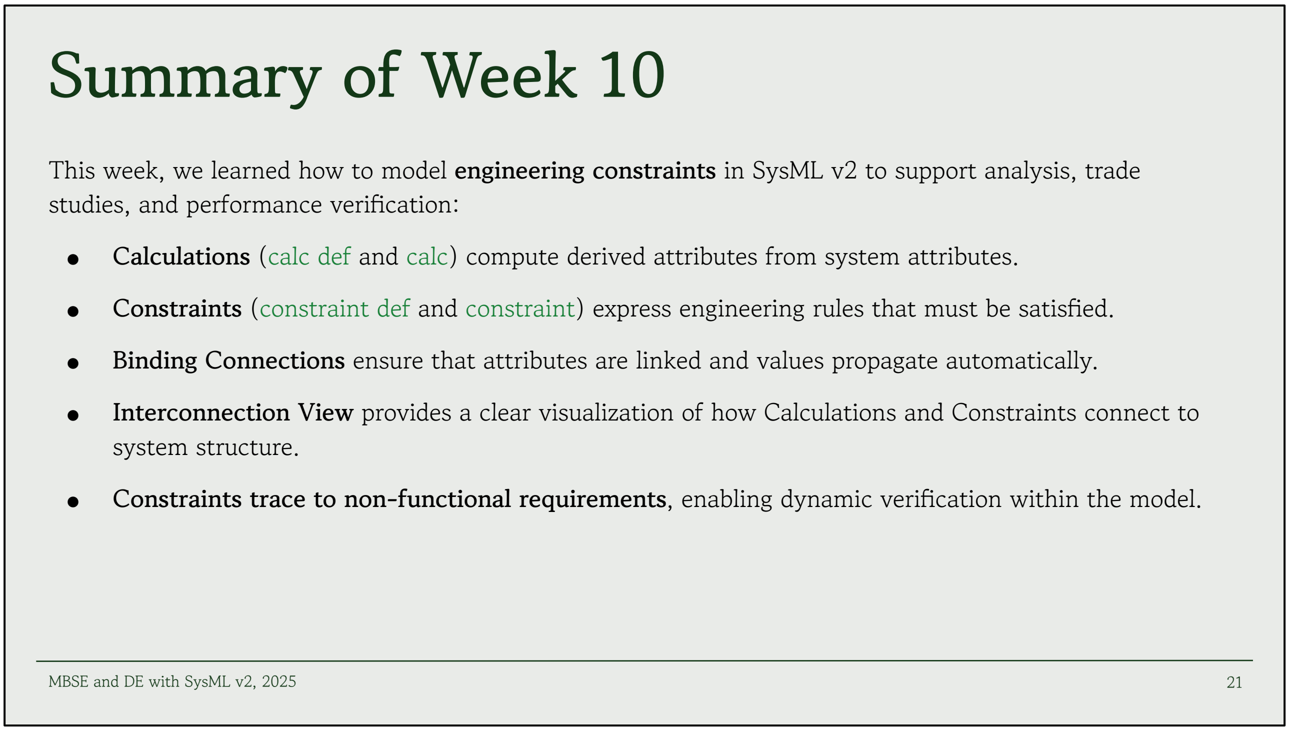
Add Constraints to Your Drone Model

For your Drone System project, apply this week’s modeling patterns to relevant performance requirements:

- Identify **at least one non-functional requirement** that can be expressed mathematically (e.g., endurance, payload capacity, weight limit).
- Define a corresponding **constraint def** and **constraint** in your model.
- If needed, define a **calc def** and **calc** to compute any derived attributes.
- Use **Binding Connections** to link system attributes, Calculations, and Constraints.
- Visualize the relationships in **Interconnection View**.



For this week’s homework, you will extend your Drone System model by adding Constraints that trace to your performance-related requirements.
Begin by identifying **one non-functional requirement** that can be modeled mathematically. Good candidates include endurance, payload capacity, or total system mass.
Then, define a **constraint def** and apply it to your model.
If the required attribute is derived — such as endurance — you may also define a **calc def** and use it to compute the value.
Bind the relevant attributes using **Binding Connections** — and use **Interconnection View** to visualize how the system architecture, Calculations, and Constraints are linked.
This will demonstrate that your Drone model is becoming not just a structural model — but an analytical model that supports verification and trade studies.



Summary of Week 10

This week, we learned how to model **engineering constraints** in SysML v2 to support analysis, trade studies, and performance verification:

- **Calculations** (calc def and calc) compute derived attributes from system attributes.
- **Constraints** (constraint def and constraint) express engineering rules that must be satisfied.
- **Binding Connections** ensure that attributes are linked and values propagate automatically.
- **Interconnection View** provides a clear visualization of how Calculations and Constraints connect to system structure.
- **Constraints trace to non-functional requirements**, enabling dynamic verification within the model.

Let us review what we have achieved this week.
We explored how to model **engineering constraints** in SysML v2 — an essential capability for moving from descriptive to analytical models.
We learned that **Calculations** allow us to compute derived attributes — using calc def and calc.
We saw how **Constraints** express engineering rules — using constraint def and constraint — and how they allow us to test whether our system meets its performance goals.
We understood that **Binding Connections** propagate values through the model — keeping the system consistent and supporting dynamic analysis.
We used **Interconnection View** as a modern parametric view — visualizing how Calculations and Constraints connect to system structure.
Finally, we saw how **Constraints trace back to non-functional requirements**, providing a powerful mechanism for verification and decision support.
This is how we turn our models into tools for real engineering — supporting analysis, trade studies, and continuous verification.

QUESTION!