System Architecture And Design

Contrasting { Development Space, Architectural Space, Design Space.

CPP-Summit 2020

C++ and System Software Summit

December 4-5 Hyatt Regency Shenzhen Airport

charley bay

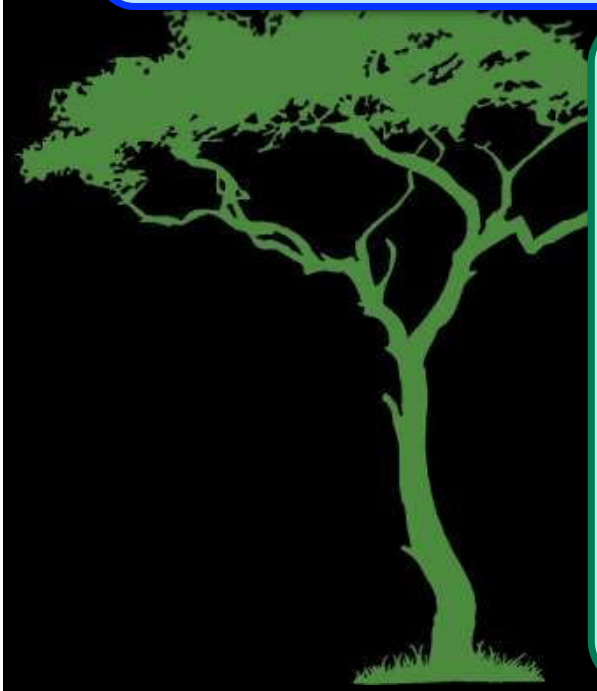
"Space"

Space *(def):* The gradient of concerns that defines possibilities

- *Is often **N-dimensional***
- *Each concern may be **weighted***
- *Each concern may be **ranked***

Space **dictates**:

- What **concepts are represented**
- What **issues and solutions are possible**



Space **implies**:

- Some issues and solutions are **simply discussed** *(the space represents the domain for exploring those issues and solutions)*
- Some issues or solutions are **invisible** *(the space does not represent them):*
  1. Issue is **orthogonal** to that space *(unrelated)*
  2. Issue **cannot be described nor addressed** *(must be managed by a different space)*

Think In Terms Of...

CPP-Summit

As **Developers**, We think in terms of:

- **Programming Language** *(syntactic and semantic rules)*
- **Paradigm** *("How To Think" in breaking down a problem)*
- The **Problem To Be Addressed** *(what technical and ergonomic concerns exist?)*

Think In Terms Of...

CPP-Summit

As **Developers**, We think in terms of:

- **Programming Language** *(syntactic and semantic rules)*
- **Paradigm** *("How To Think" in breaking down a problem)*
- The **Problem To Be Addressed** *(what technical and ergonomic concerns exist?)*

Is all about:
Technical Solutions

Think In Terms Of...

CPP-Summit

As **Developers**, We think in terms of:

Is all about: **Technical Solutions**

- **Programming Language** *(syntactic and semantic rules)*
- **Paradigm** *("How To Think" in breaking down a problem)*
- The **Problem To Be Addressed** *(what technical and ergonomic concerns exist?)*

As **Experienced Developers**, We ***also*** think about:

- The **Development Process** *(Traditional and Iterative domain exploration)*
- **Roles played** *(interaction by diverse interested parties)*
- **Dimensions defining the "spaces"** in which:
  - the **problem is defined**
  - the **solution is expressed**

Think In Terms Of...

CPP-Summit

As **Developers**, We think in terms of:

- **Programming Language** *(syntactic and semantic rules)*
- **Paradigm** *("How To Think" in breaking down a problem)*
- The **Problem To Be Addressed** *(what technical and ergonomic concerns exist?)*

Is all about:
Technical Solutions

As **Experienced Developers**, We ***also*** think about:

- The **Development Process** *(Traditional and Iterative domain exploration)*
- **Roles played** *(interaction by diverse interested parties)*
- **Dimensions defining the "spaces"** in which:
  - the **problem is defined**
  - the **solution is expressed**

Is all about:
Business Coordination, Constraints Management

"Role"

Role *(def):* A function or part performed

Example roles:

Developer, Designer, Architect,
Domain Expert, Product Owner, Program Manager,
Customer Advocate, Executive Owner

Role **dictates**:

- What you **care about**
- What you **are responsible for**

Role **implies**:

- Your **concerns are unique** *(other roles have different concerns)*
- You **must interact with other roles** *(others are worried about different but overlapping concerns)*

Roles Change Over Time

CPP-Summit

New Developers tend to focus on:

- How does this **C++ language feature** work?
- **How to implement** a technical solution?
- What is **Best Practice**?

*Is **plenty** to explore!*
(technical skills must be acquired)

Roles Change Over Time

CPP-Summit

New Developers tend to focus on:

- How does this **C++ language feature** work?
- **How to implement** a technical solution?
- What is **Best Practice**?

*Is **plenty** to explore!*
(technical skills must be acquired)

Designers tend to focus on:

- What **design approach** should this system prefer?
- What **paradigms** *(how to think)* should be used?
- How should our code **change**: What **C++ Language additions** enable **new idioms** to solve past problems?

Favor addressing some concerns, over a different design that addresses differently ranked concerns

Roles Change Over Time

Both care about **C++ Language Features**, but for **different reasons**!

New Developers tend to focus on:

- How does this **C++ language feature** work?
- **How to implement** a technical solution?
- What is **Best Practice**?

*Is **plenty** to explore!*
(technical skills must be acquired)

Designers tend to focus on:

- What **design approach** should this system prefer?
- What **paradigms** *(how to think)* should be used?
- How should our code **change**: What **C++ Language additions** enable **new idioms** to solve past problems?

Favor addressing some concerns, over a different design that addresses differently ranked concerns

Roles Change Over Time *(continued)*

CPP-Summit

Architects tend to focus on:

- What should the **system do**?
- What **technologies** should we leverage *(to permit the system to be implemented and function as expected)*?
- What **reuse** should we manage?
- What organizational **technical evolution** should we pursue?

Roles Change Over Time *(continued)*

CPP-Summit

Architects tend to focus on:

- What should the **system do**?
- What **technologies** should we leverage *(to permit the system to be implemented and function as expected)*?
- What **reuse** should we manage?
- What organizational **technical evolution** should we pursue?

Primary considerations:

1. Solution Space
2. Product Space
3. Organizational Health

Roles Change Over Time *(continued)*

CPP-Summit

Architects tend to focus on:

- What should the **system do**?
- What **technologies** should we leverage *(to permit the system to be implemented and function as expected)*?
- What **reuse** should we manage?
- What organizational **technical evolution** should we pursue?

Primary considerations:

1. Solution Space
2. Product Space
3. Organizational Health

Secondary considerations:

1. External technological landscape *(including C++ Language advances)*

Caring About C++ Language Evolution

CPP-Summit

Developers in all roles care about C++ Language Features, but for **different reasons**!

New Developer

1. How do I **do my job**?
2. What is **Best Practice**?

Caring About C++ Language Evolution

CPP-Summit

Developers in all roles care about C++ Language Features, but for **different reasons**!

New Developer

1. How do I **do my job**?
2. What is **Best Practice**?

Experienced Designer

1. What **design idioms** are now available?
2. What problems can I solve now, in a more **robust** or **elegant** manner?

Caring About C++ Language Evolution

CPP-Summit

Developers in all roles care about C++ Language Features, but for **different reasons**!

New Developer

1. How do I **do my job**?
2. What is **Best Practice**?

Experienced Designer

1. What **design idioms** are now available?
2. What problems can I solve now, in a more **robust** or **elegant** manner?

Architect

1. How should our **products change** *(use of hardware and software)*?
2. What **C++ Language features** make it compelling…
   - To prefer C++ over other languages
   - To prefer C++ over alternative *(such as implementing subsystem in hardware)*

Developer's Perspective Of Roles

CPP-Summit

- Q: How is "Experienced Designer" different from "Architect"?

Developer's Perspective Of Roles

CPP-Summit

- Q: How is "Experienced Designer" different from "Architect"?
- A:

Architect

is concerned about the business direction

Developer's Perspective Of Roles

CPP-Summit

- Q: How is "Experienced Designer" different from "Architect"?
- A:

Architect	**Designer**
is concerned about the business direction	is concerned about the successful technical delivery

Developer's Perspective Of Roles

CPP-Summit

- Q: How is "Experienced Designer" different from "Architect"?
- A:

Architect

is concerned about the business direction

Designer

is concerned about the successful technical delivery

New Developer

is concerned about making sense of all the "sparkly" things
(not yet able to rank importance among topics)

How Development Works

Non-Optional: Architecture and Design

Review: Development Overview

Process Artifacts
(things delivered)

- The **Problem Space** defines the **Business Need**

  …leading to

  - **System Analysis** defining *"**What** needs to be addressed"*

    …leading to

    - **Proposed Architectures** defining *"**How** to establish a solution"*

Review: Development Overview

Process Artifacts
(things delivered)

- The **Problem Space** defines the **Business Need**
  …leading to
  - **System Analysis** defining *"**What** needs to be addressed"*
    …leading to
    - **Proposed Architectures** defining *"**How** to establish a solution"*

Roles Played
(responsible actors)

Business Leadership
Prioritize within the problem space

Technical Leadership
(Architects and Designers) manage technical aspects that define the solution space

Roles and responsibilities are unique and not overlapping; but in practice are often confused due to tradeoffs resulting from:

- Constraints
- Preferences
- Capabilities
- Cross-functional ranking of requirements

The "Olden Days"

CPP-Summit

1 **Requirements Analysis**
- Specification for what the system must do

- Who does this? ➔ Business Managers

2 **Architecture & Design**
- Proposed system to deliver that specified in 1

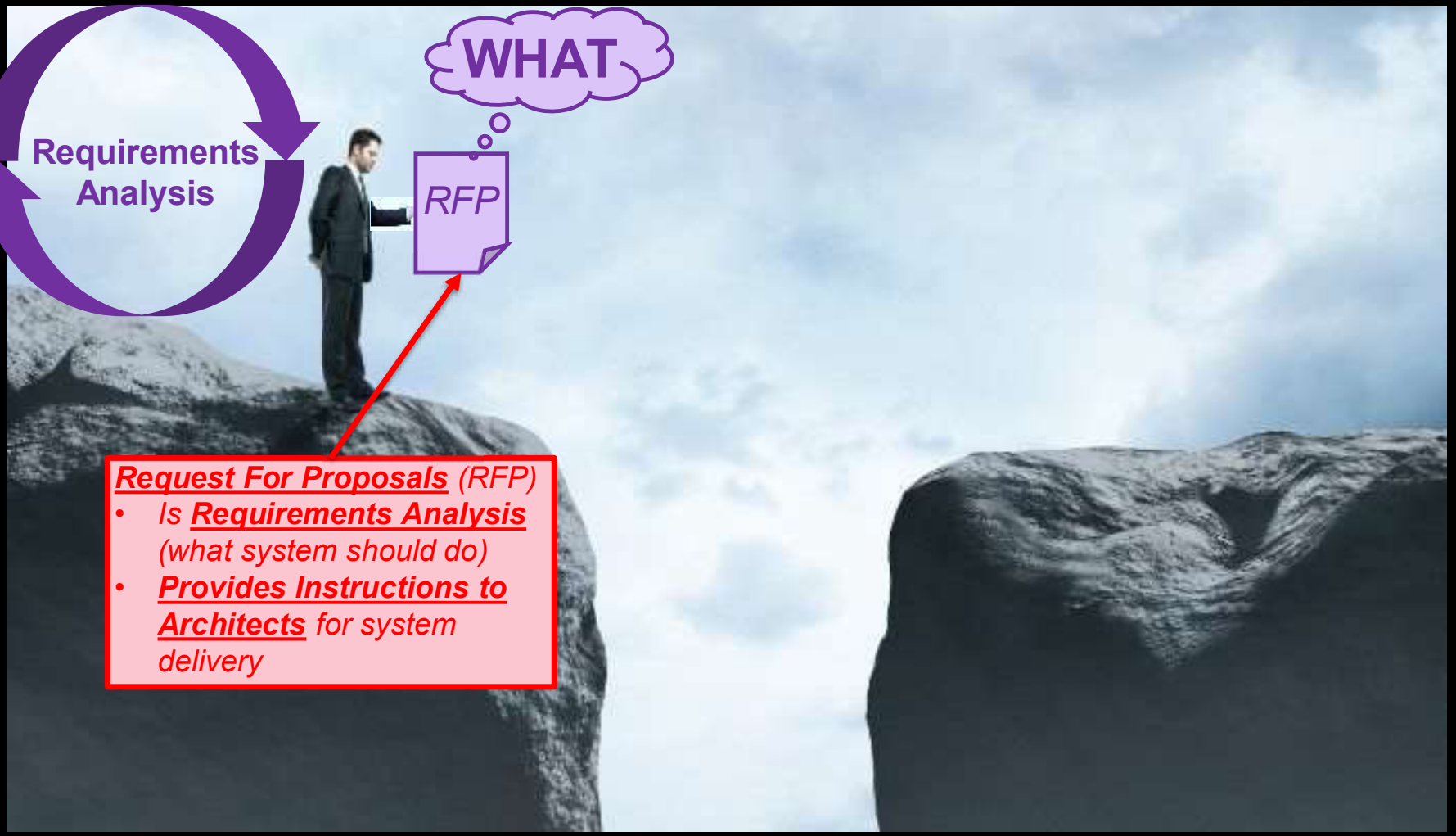
- Who does this? ➔ Architects & Designers

3 **Implementation**
- Construct system specified in 2

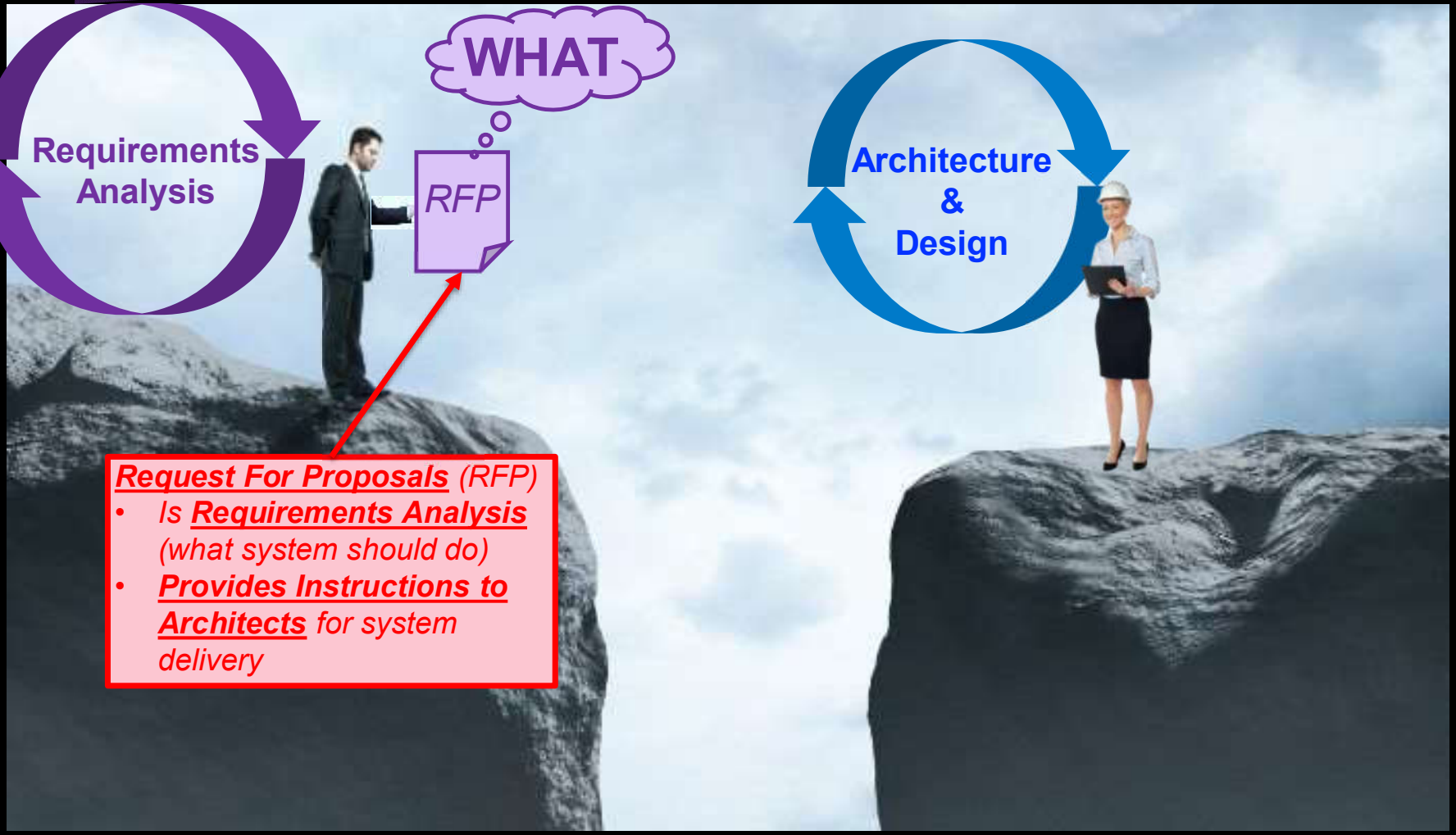
- Who does this? ➔ Architects & Designers & Developers

The "Olden Days" *(same thing, different view)*

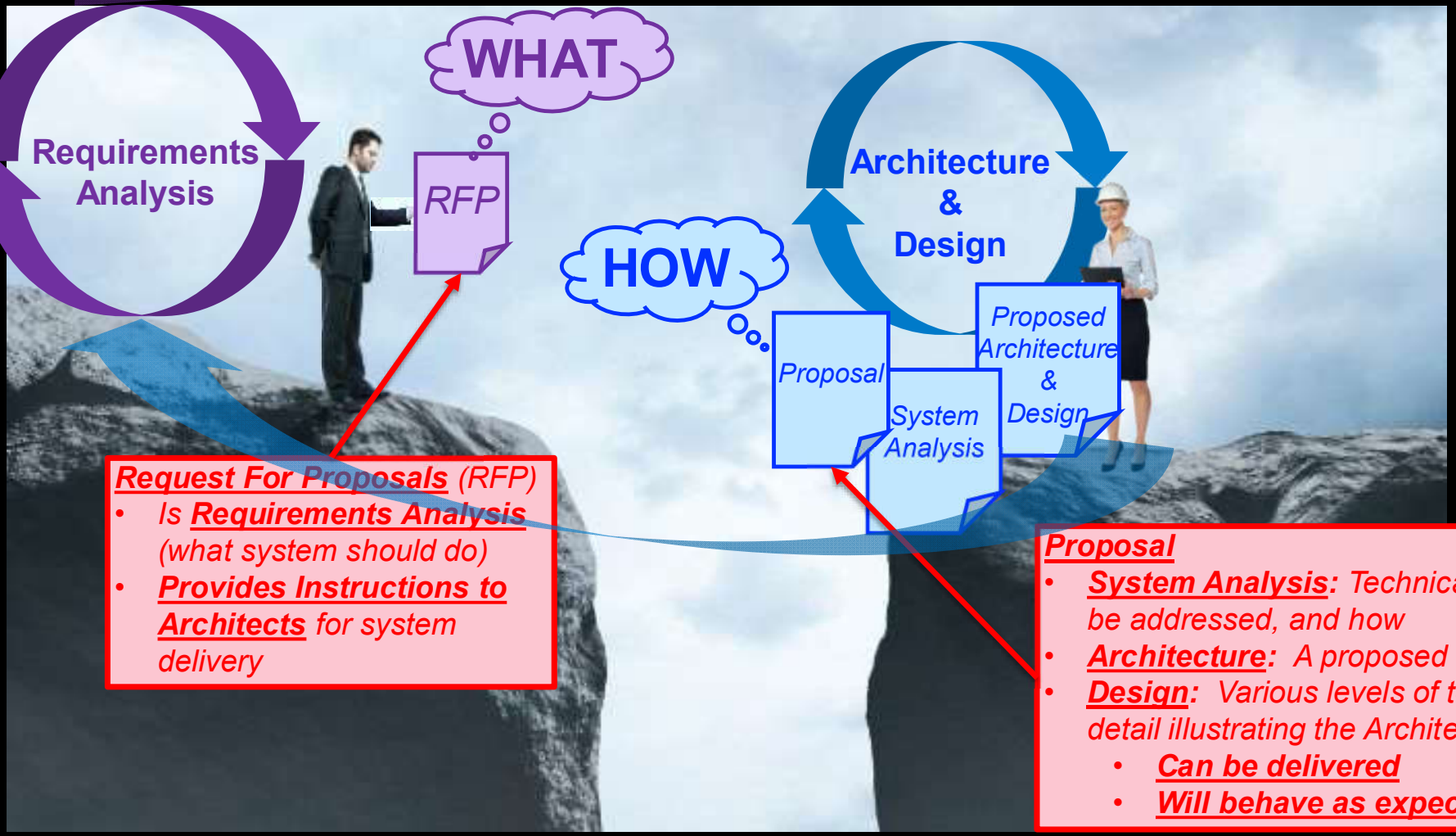
CPP-Summit



The "Olden Days" *(same thing, different view)*



The "Olden Days" *(same thing, different view)*



Request For Proposals *(RFP)*

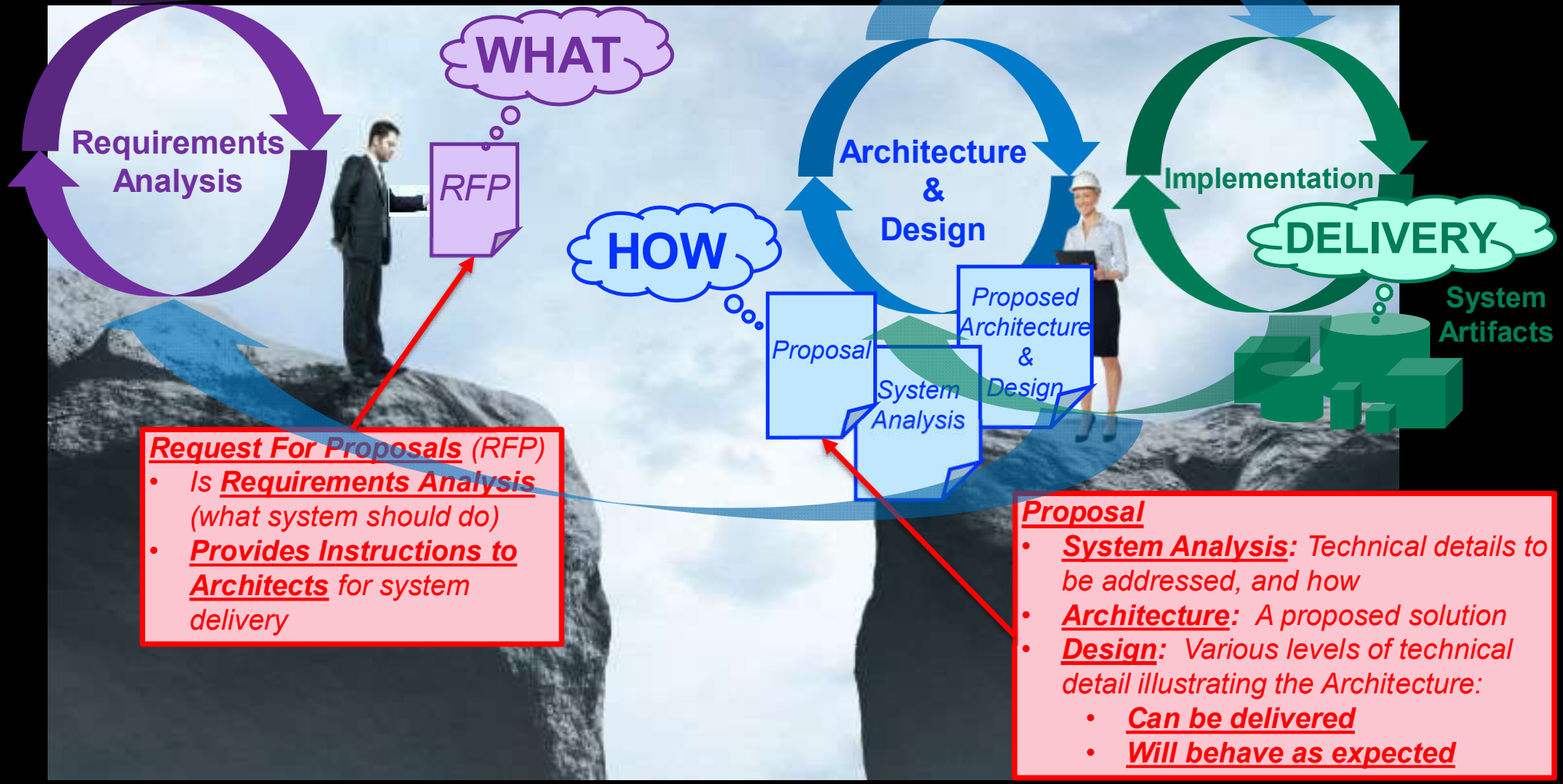
- *Is* ***Requirements Analysis*** *(what system should do)*
- ***Provides Instructions to Architects*** *for system delivery*

Proposal

- ***System Analysis:*** *Technical details to be addressed, and how*
- ***Architecture:*** *A proposed solution*
- ***Design:*** *Various levels of technical detail illustrating the Architecture:*
  - ***Can be delivered***
  - ***Will behave as expected***

The "Olden Days" *(same thing, different view)*

CPP-Summit



Request For Proposals *(RFP)*

- *Is* ***Requirements Analysis*** *(what system should do)*
- ***Provides Instructions to Architects*** *for system delivery*

Proposal

- ***System Analysis:*** *Technical details to be addressed, and how*
- ***Architecture:*** *A proposed solution*
- ***Design:*** *Various levels of technical detail illustrating the Architecture:*
  - ***Can be delivered***
  - ***Will behave as expected***

Should Not Mix

- Some things you ***should* not** mix:
  - Do not mix **beer** with **wine**
  - Do not mix **business** with **pleasure**
  - Do not mix **ammonia** with **bleach**
  - Do not mix **ketchup** with **baking soda**

***Bad Things Happen*™**
When you mix what should not be mixed

Ketchup and Baking Soda Prank (GONE WRONG) At Home
1.9M views 2.6K 206 SHARE SAVE

https://www.youtube.com/watch?v=fov9cj5Vsj4

Never Mix

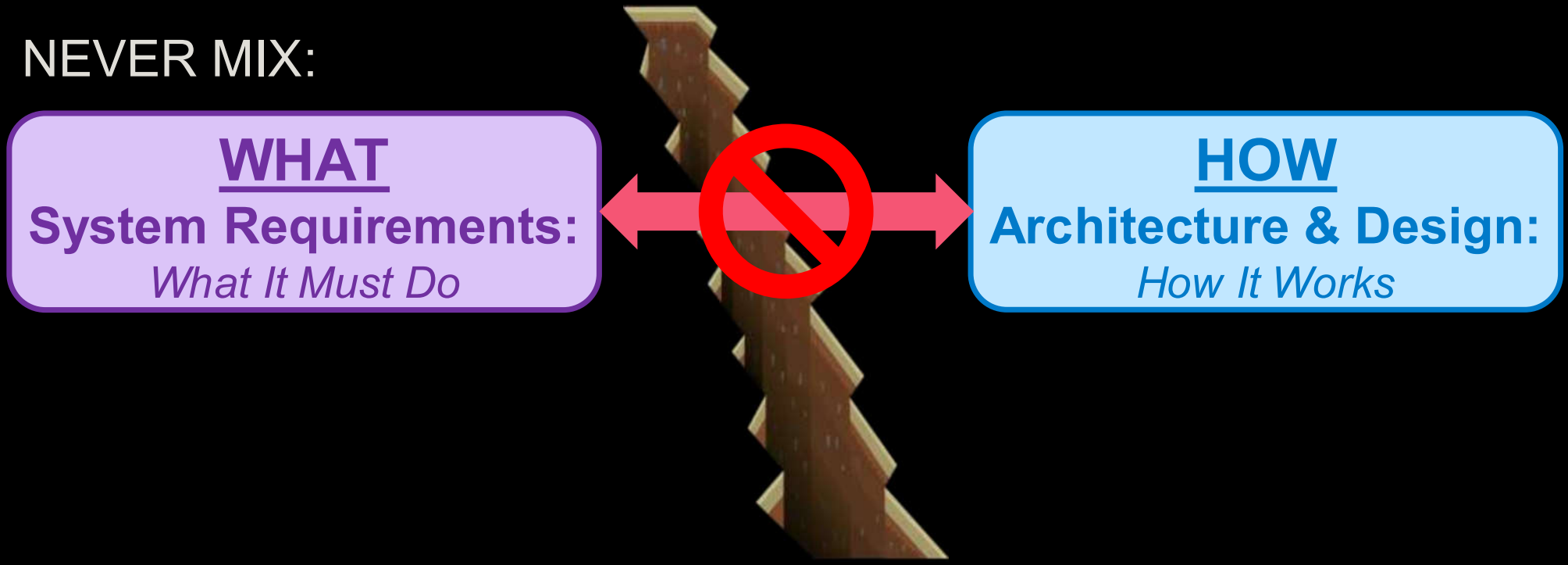
CPP-Summit

- Some things you can **NEVER MIX**
  - You must **ALWAYS know** when you are doing “one” or the “other”
  - If you discover you are **unsure which** you are doing, **STOP** *(and figure it out)*

Never Mix

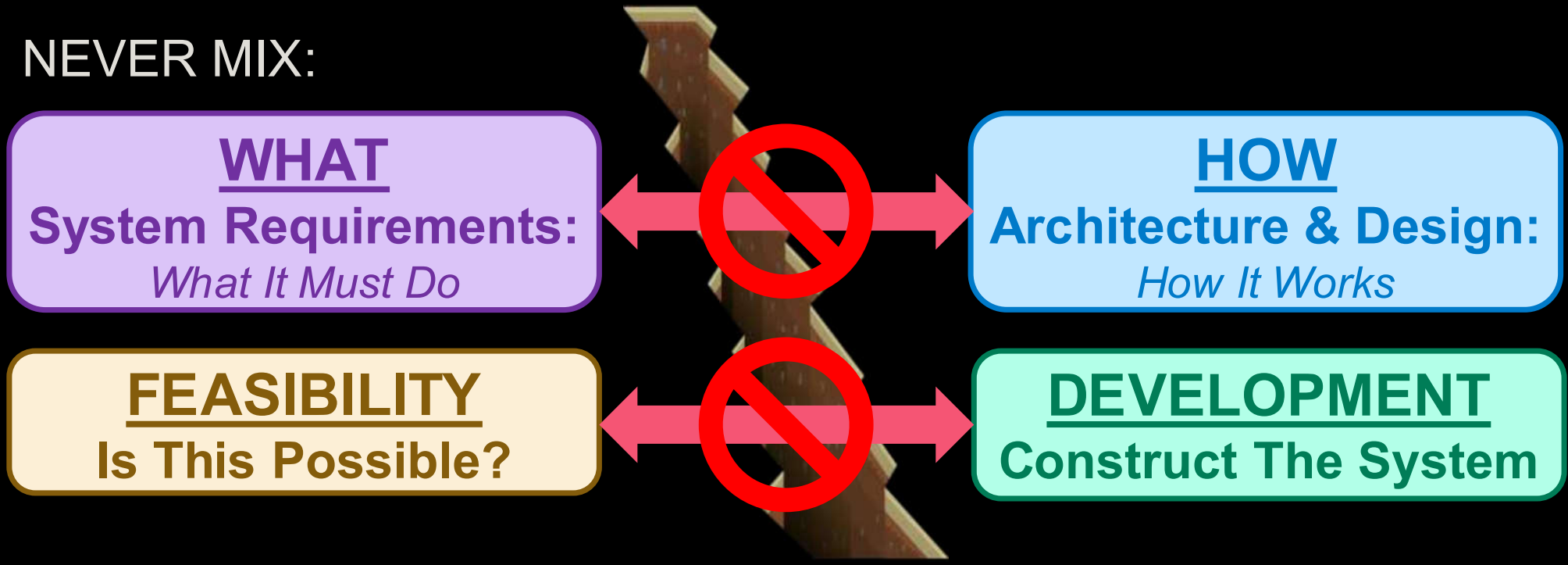
CPP-Summit

- Some things you can **NEVER MIX**
  - You must **ALWAYS know** when you are doing "one" or the "other"
  - If you discover you are **unsure which** you are doing, **STOP** *(and figure it out)*
- NEVER MIX:



Never Mix

- Some things you can **NEVER MIX**
  - You must **ALWAYS know** when you are doing "one" or the "other"
  - If you discover you are **unsure which** you are doing, **STOP** *(and figure it out)*
- NEVER MIX:



System Analysis

CPP-Summit

Given: System Requirements
Perform: Technical breakdown of **essential system behavior**

- **Latency**: How fast must the system respond?
  - What things must be low-latency *(quantify it!)*
  - What acceptable latency tolerances *(by event-category and processing-category)*?
- **Throughput**: What aggregate throughput is required?
  - "Big-pipes": Data acquisition, stream processing, client-transfer
  - "Small-pipes": Status-and-control, health monitoring, telemetry
- **Sensitivity**: What "lost events" are tolerable?
  - Can we lose some status-and-control, if we log the failure?
  - Can we miss some acquisition events *(without detecting the event?)*

Common Examples *(must use criteria for your system)*

Using System Analysis

Business Managers

Requirements

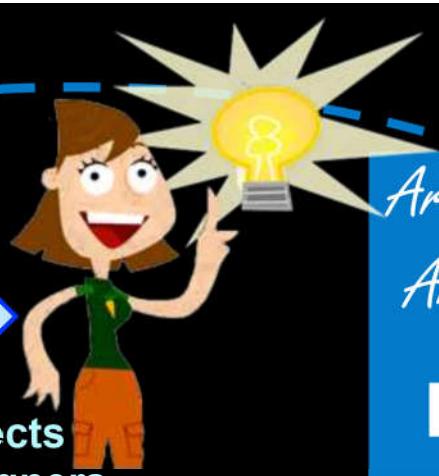
BUSINESS WHAT

Informs

System Analysis

TECHNICAL WHAT

Informs



Architects and Designers

Architecture And Design

HOW

Problem Space

Solution Space

After Requirements, and System Analysis:

- **Conceptualize** scenarios and frameworks that **achieve what is required** from system analysis

Enables delivery of system with required behavior

  - Each conception is an **architectural option** *(defines the total system Theory of Operation)*
  - Each conception for a subsystem is a **design option** *(defines subsystem Theory of Operation, which must align within the whole system conception)*

Be Wary: Overspecification

Overspecification *(def):*
An erroneous constraint

Why Spiral / Agile?
Because **overspecification hurts** your system *(and you!)*

*Erroneous constraints **always** provide **negative value***

We are often tempted to "over-specify":

1. **Specify** *(prescribe)* detail which is **not necessary** detail
2. **Assume** necessary behavior *(which is not necessary behavior)*

Spiral / Agile tends to "skip" *(by "jumping over")* this part of the process, to not be caught by these mistakes

Overspecifications are **identified** by *(continually)* re-visiting **"First Principles"** *(i.e., the system's **Purpose**)*

Corollary: Any assumption unrelated to purpose is not a "real" specification/constraint

*One might surmise that the entire Agile movement is motivated solely through a **fear of overspecification***

A Real Problem™ That Agile does Really Solve™

Overspecification Hurts

Overspecification *(def):*
An erroneous constraint

Erroneous constraints **always** provide **negative value**

Over-Specifications

- **Limit options**
  - For **no benefit**
  - For **negative benefit**
- **Confuse/Hide** the actual **constraints**!
  - **Harder to discover** elegant patterns
  - **Harder to reason** about *(understand)* behavior
- **Slow** our systems
  - by adding **unnecessary constraints**
- **Break** our systems
  - when unnecessary constraints are *(eventually)* violated

Over-specifications
are **identified** by *(continually)*
re-visiting **"First Principles"**
*(i.e., the system's **Purpose**)*

Corollary: Any assumption unrelated to purpose is not a "real" specification/constraint

Review: Terms To Limit Your System

CPP-Summit

- These terms relate to “limits” in delivery of Your System:

Requirement *(def):* A mandated specification

Constraint *(def):* A limit of possibility

Preference *(def):* An expressed bias to rank alternatives

Capability *(def):* An ability to deliver

Understand Your *(System-)*Limits

CPP-Summit

Overspecification is a painful limit, but only one limit.

As architect, you **balance many limits**:

Requirement You are limited by **mandate** *(specification)* for **what system must do**

Understand Your *(System-)*Limits

CPP-Summit

Overspecification is a painful limit, but only one limit.
As architect, you **balance many limits**:

Requirement You are limited by **mandate** *(specification)* for **what system must do**

Constraint You are limited by **what is possible** *(limits of technology or physics)*

Understand Your *(System-)*Limits

CPP-Summit

Overspecification is a painful limit, but only one limit.
As architect, you **balance many limits**:

Requirement You are limited by **mandate** *(specification)* for **what system must do**

Constraint You are limited by **what is possible** *(limits of technology or physics)*

Preference You are limited by **bias** expressing preferred tradeoffs

Understand Your *(System-)*Limits

CPP-Summit

Overspecification is a painful limit, but only one limit.
As architect, you **balance many limits**:

Requirement — You are limited by **mandate** *(specification)* for **what system must do**

Constraint — You are limited by **what is possible** *(limits of technology or physics)*

Preference — You are limited by **bias** expressing preferred tradeoffs

Capability — You are limited by the **implementation team** *(what they understand, and can deliver; their availability and capabilities)*

Understand Your *(System-)*Limits

CPP-Summit

Overspecification is a painful limit, but only one limit.
As architect, you **balance many limits**:

Requirement — You are limited by **mandate** *(specification)* for **what system must do**

How to relax?

Constraint — You are limited by **what is possible** *(limits of technology or physics)*

How to relax?

Preference — You are limited by bias expressing preferred tradeoffs

How to relax?

Capability — You are limited by the **implementation team** *(what they understand, and can deliver; their availability and capabilities)*

How to relax?

Understand Your *(System-)*Limits

CPP-Summit

Overspecification is a painful limit, but only one limit.
As architect, you **balance many limits**:

Requirement — You are limited by **mandate** *(specification)* for **what system must do**

How to relax?

- *Can:* **Negotiate the mandate**, or change "Acceptance Criteria"

Constraint — You are limited by **what is possible** *(limits of technology or physics)*

How to relax?

Preference — You are limited by bias expressing preferred tradeoffs

How to relax?

Capability — You are limited by the **implementation team** *(what they understand, and can deliver; their availability and capabilities)*

How to relax?

Understand Your *(System-)*Limits

CPP-Summit

Overspecification is a painful limit, but only one limit.
As architect, you **balance many limits**:

Requirement — You are limited by **mandate** *(specification)* for **what system must do**

How to relax?
- *Can:* **Negotiate the mandate**, or change "Acceptance Criteria"

Constraint — You are limited by **what is possible** *(limits of technology or physics)*

How to relax?
- *Can:* **Explore alternative technology**

Preference — You are limited by bias expressing preferred tradeoffs

How to relax?

Capability — You are limited by the **implementation team** *(what they understand, and can deliver; their availability and capabilities)*

How to relax?

Understand Your *(System-)*Limits

CPP-Summit

Overspecification is a painful limit, but only one limit.
As architect, you **balance many limits**:

Requirement — You are limited by **mandate** *(specification)* for **what system must do**

How to relax?
- *Can:* **Negotiate the mandate**, or change "Acceptance Criteria"

Constraint — You are limited by **what is possible** *(limits of technology or physics)*

How to relax?
- *Can:* **Explore alternative technology**

Preference — You are limited by bias expressing preferred tradeoffs

How to relax?
- *Can:* **Negotiate different tradeoff ranking**

Capability — You are limited by the **implementation team** *(what they understand, and can deliver; their availability and capabilities)*

How to relax?

Understand Your *(System-)*Limits

CPP-Summit

Overspecification is a painful limit, but only one limit.
As architect, you **balance many limits**:

Requirement — You are limited by **mandate** *(specification)* for **what system must do**

How to relax?
- *Can:* **Negotiate the mandate**, or change "Acceptance Criteria"

Constraint — You are limited by **what is possible** *(limits of technology or physics)*

How to relax?
- *Can:* **Explore alternative technology**

Preference — You are limited by bias expressing preferred tradeoffs

How to relax?
- *Can:* **Negotiate different tradeoff ranking**

Capability — You are limited by the **implementation team** *(what they understand, and can deliver; their availability and capabilities)*

How to relax?
- *Can:* **Train the team**, or **use different team**

Review: Development Model

CPP-Summit

Requirements: Is all about **system purpose** ← *…So we can address our Business Need*

System Analysis: Is all about **technical behavior** ← *…So system purpose can be achieved*

Architecture & Design: Is all about **Theory of Operation** ← *…For a chosen approach*

Implementation: Is all about **construction and delivery** ← *…To manifest Theory of Operation*

Which of these is **optional**?

(Which do you want to leave out?)

*What is **System Development**?*

CPP-Summit

The **delivery** of
identified and managed **dependencies**
with **respect to limits** of
requirement, preference, constraint, and capability.

Who manages this?

*The **Architect**.*

Tough job?

Yes, look what you are balancing:

Delivery of the right dependencies,
given the reality of your constraints.

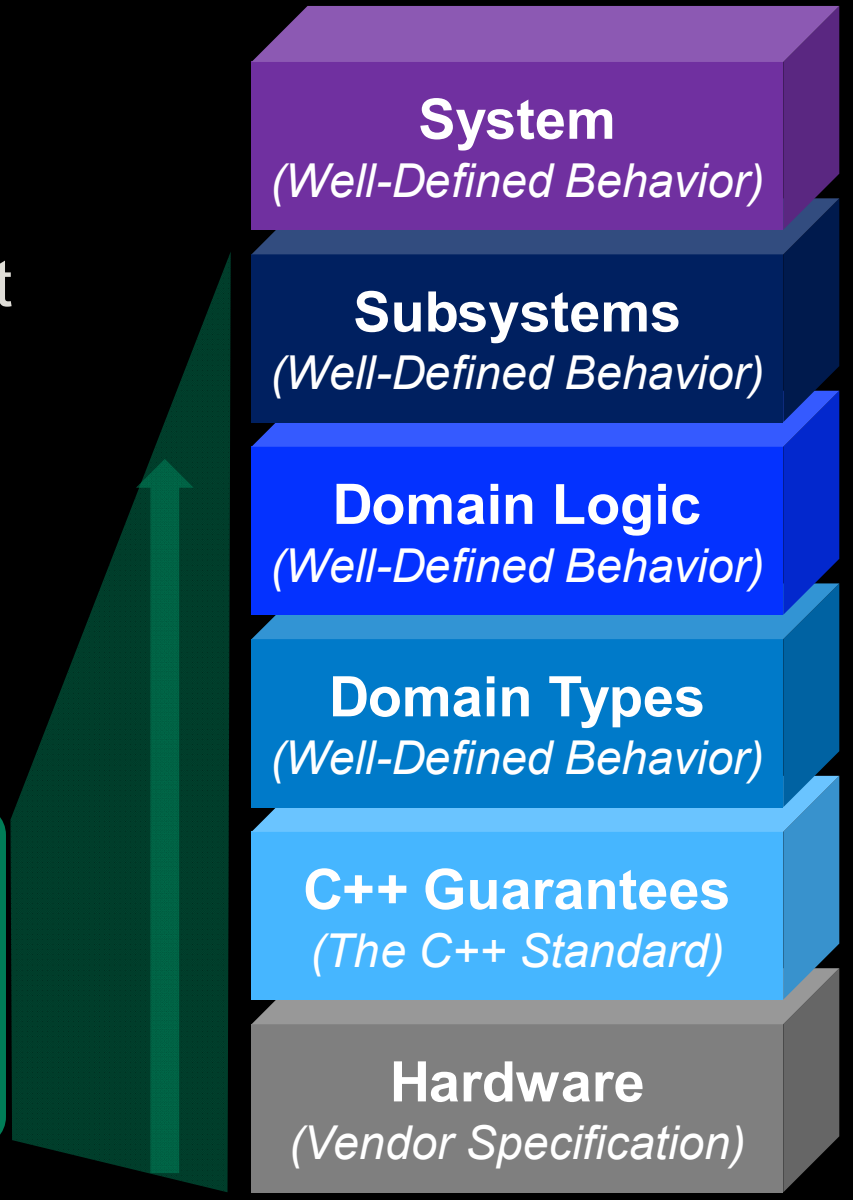
That's Engineering 101

Using C++ Looks Like...

Because of **direct reasoning** through the **Abstract State Machine**, C++ development looks like the following:

"Bottom-Up": Developer maps from "hardware-up" through C++ Guarantees

1. Define **domain types** *(from fundamental types)*
2. Implement **domain algorithms** *(from domain types)*
3. Implement **domain subsystems** *(from domain algorithms)*



Using C++ Looks Like...

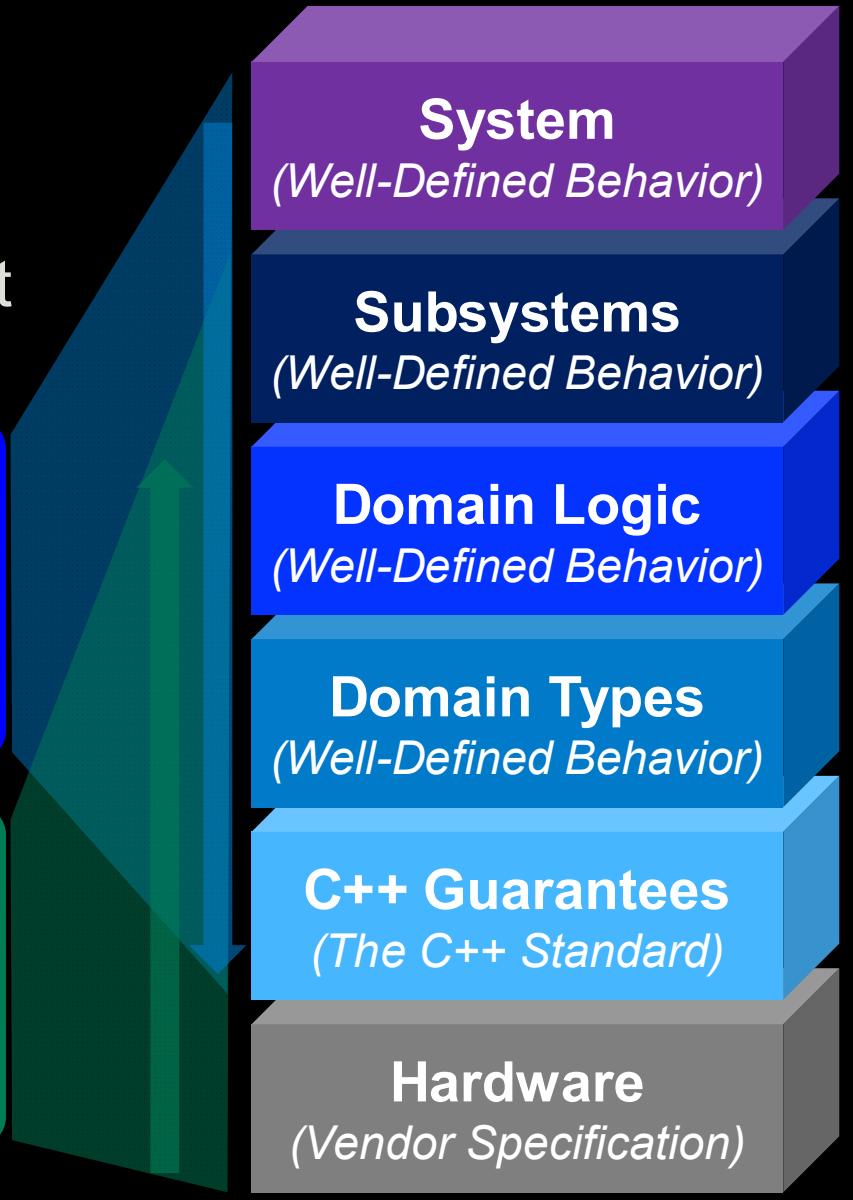
Because of **direct reasoning** through the **Abstract State Machine**, C++ development looks like the following:

"Top-Down": Developer maps from conceptualized system through domain types

1. Define **Theory of Operation**
2. Implement **domain subsystems** *(for Theory of Operation)*
3. Implement **domain algorithms** *(from domain subsystems)*

"Bottom-Up": Developer maps from "hardware-up" through C++ Guarantees

1. Define **domain types** *(from fundamental types)*
2. Implement **domain algorithms** *(from domain types)*
3. Implement **domain subsystems** *(from domain algorithms)*



Development Axes

Many Degrees Of Freedom

Real-World Development Stages

CPP-Summit

- What To Expect In Development *(Real-World):*

1. **System Analysis** *(**What** do you want to build?)*

Real-World Development Stages

CPP-Summit

- What To Expect In Development *(Real-World):*

1 **System Analysis** *(**What** do you want to build?)*

No, seriously, what are we being paid to deliver?

Real-World Development Stages

CPP-Summit

- What To Expect In Development *(Real-World):*

1 **System Analysis** *(**What** do you want to build?)*

No, seriously, what are we being paid to deliver?

2 **Conceptualize**
*(Define **Theory of Operation** by contrasting the system you **will** build against systems you will **not** build)*

Feasibility *(What **specific** technical questions must be answered before you understand the target system, and you **know the system can be built**?)*

Prototype *(Demonstrate the system can be viably built)*

1. Can be combined with Feasibility
2. Can be skipped if you have an existing system, and new system uses similar technologies

Real-World Development Stages

CPP-Summit

- What To Expect In Development *(Real-World):*

1 **System Analysis** *(**What** do you want to build?)*

No, seriously, what are we being paid to deliver?

2 **Conceptualize**
*(Define **Theory of Operation** by contrasting the system you **will** build against systems you will **not** build)*

Feasibility *(What **specific** technical questions must be answered before you understand the target system, and you **know the system can be built**?)*

Prototype *(Demonstrate the system can be viably built)*
1. Can be combined with Feasibility
2. Can be skipped if you have an existing system, and new system uses similar technologies

*The only **Hard-Barrier!***

```
if(prototype_works)
  goto 3;
goto 2;
```

3 **Development** *(Build It!)*
- **Design** *(Implement Theory of Operation)*
- **Implement** *(Code / Implement Design)*
- **Validate** *(Confirm need is addressed, Verify technical behavior)*

Real-World Development Stages

CPP-Summit

- What To Expect In Development *(Real-World):*

1 **System Analysis** *(**What** do you want to build?)*

No, seriously, what are we being paid to deliver?

2 **Conceptualize**
*(Define **Theory of Operation** by contrasting the system you **will** build against systems you will **not** build)*

Feasibility *(What **specific** technical questions must be answered before you understand the target system, and you **know the system can be built**?)*

Prototype *(Demonstrate the system can be viably built)*
1. Can be combined with Feasibility
2. Can be skipped if you have an existing system, and new system uses similar technologies

*The only **Hard-Barrier!***

```
if(prototype_works)
  goto 3;
goto 2;
```

3 **Development** *(Build It!)*
- **Design** *(Implement Theory of Operation)*
- **Implement** *(Code / Implement Design)*
- **Validate** *(Confirm need is addressed, Verify technical behavior)*

4 **Deploy** *(e.g., Handoff to manufacturing, update user docs, train support staff, make available to customer)*

Feedback from later phases can "inform" earlier phases

"Deploy" (customer feedback) can impact System Concept

The Only "Hard Barrier"

The Architect must be the "adult" when considering *"sparkly new things"*

NEVER "Scale Up" a Feasibility exercise

(pretending it to be "Development")

- *Leads to:*

Massive Cost Increases

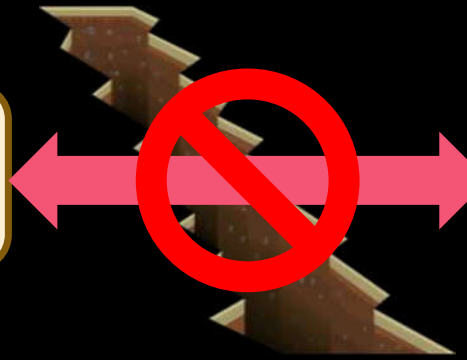
(production-level resources on an unproven experiment)

Amplified Risk

(deployment-level blast radius for an unproven experiment)

Quality Drop, Developer Confusion

(mingles lower-standard experimentation with production-level quality expectations)



FEASIBILITY
Is This Possible?

DEVELOPMENT
Construct The System

Parallel Progression

CPP-Summit

May evolve "in parallel":

What are you building?

Concept Progression
(evolve across phases):

➔Requirements
➔Conceptualization
➔Development

Surprise!

- Key Customer ($) redirects effort
- Target use-case changed
- Business focus changed

Perhaps due to acquisition, buyout, or merger

Perhaps due to "fast-moving" market

How is the building going?

Development Progression
(evolve across phases):

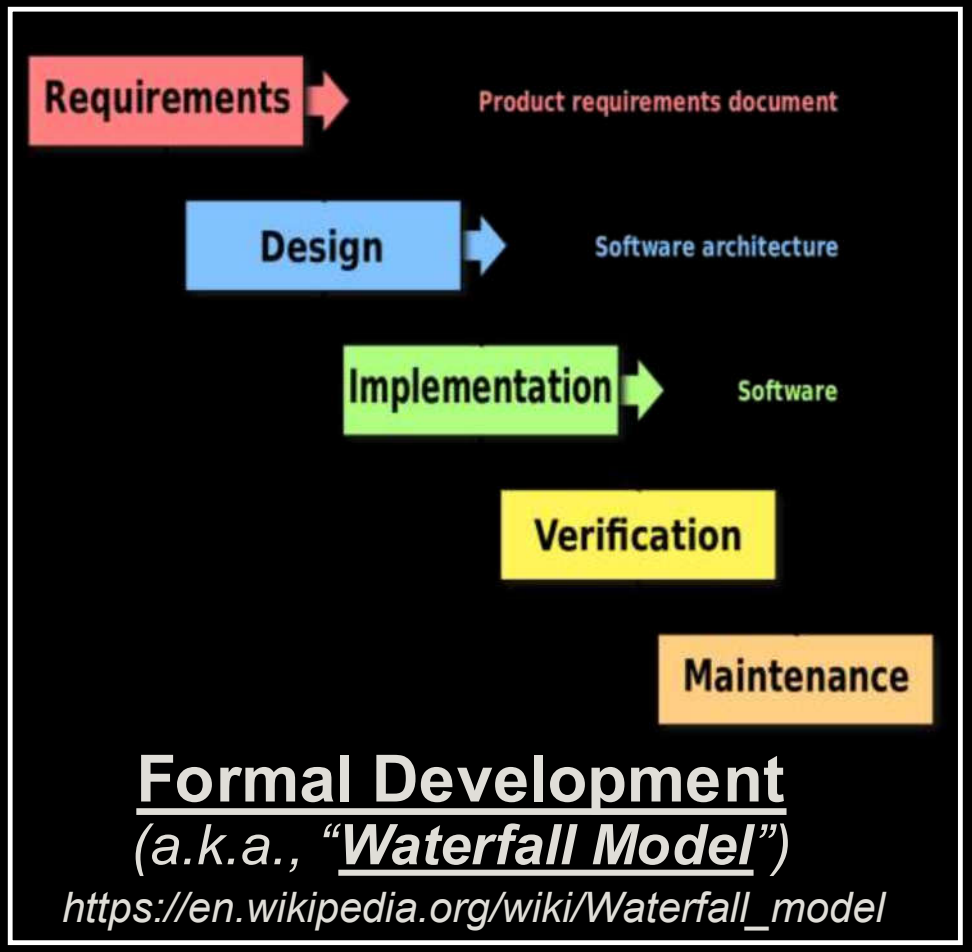
➔Feasibility / Prototyping
➔Development
➔Productization

Surprise!

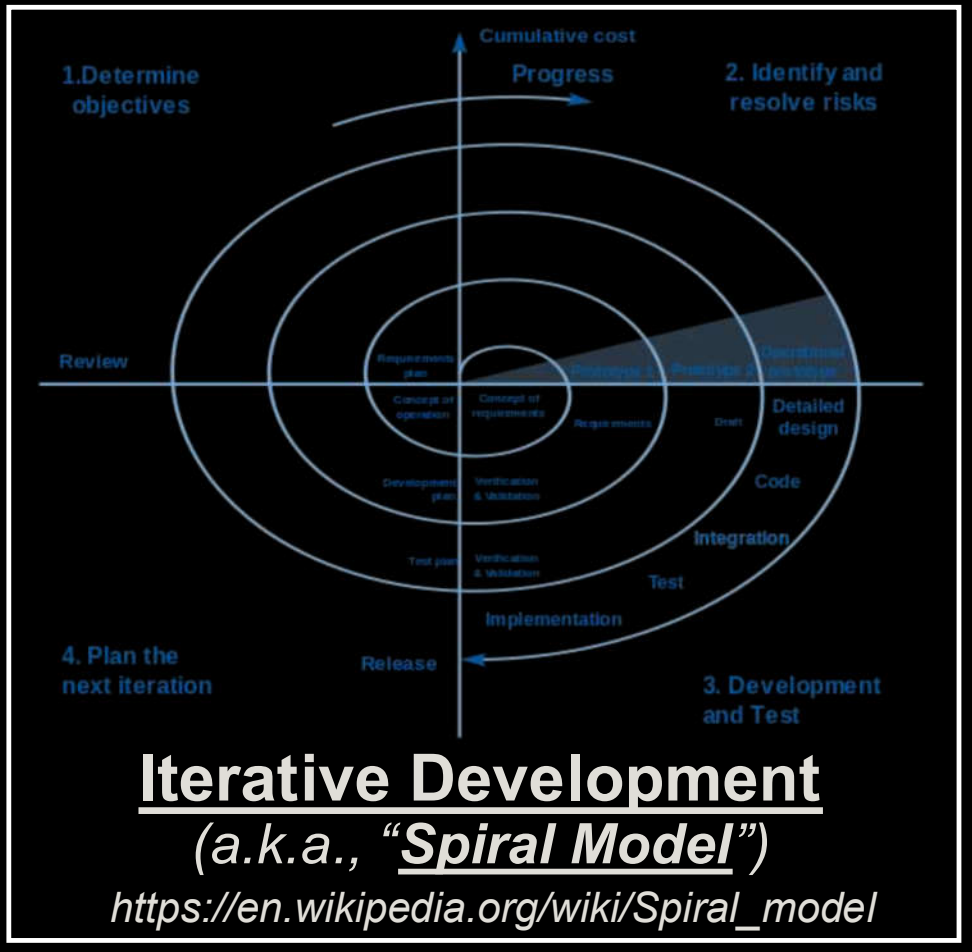
- Port to new platform / OS
- Port to new compiler / toolchain
- Change / Add 3rd party subsystem

Two Models Exist

- In general, Two (2) Development Models exist:



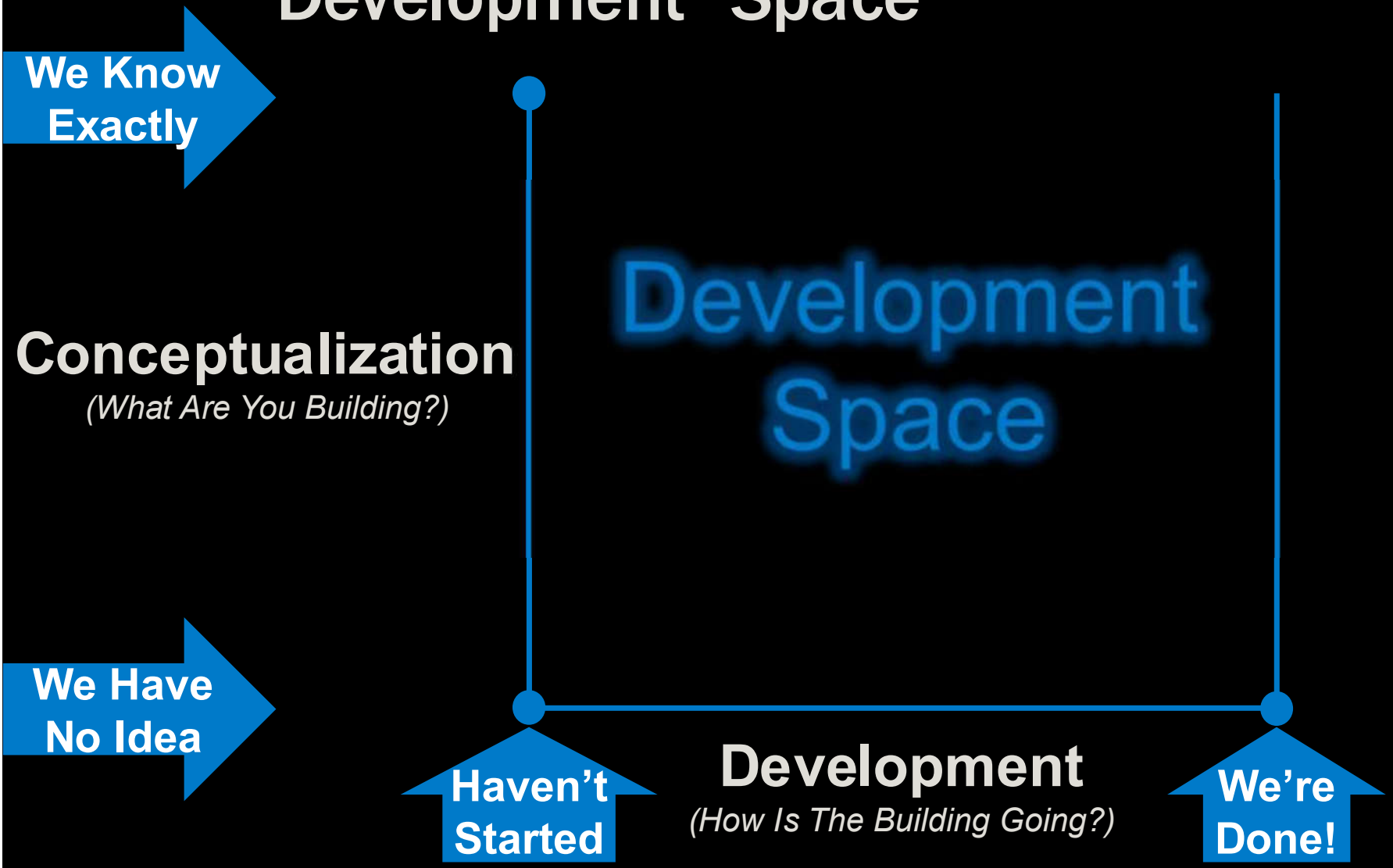
Formal Development
*(a.k.a., "**Waterfall Model**")*
https://en.wikipedia.org/wiki/Waterfall_model



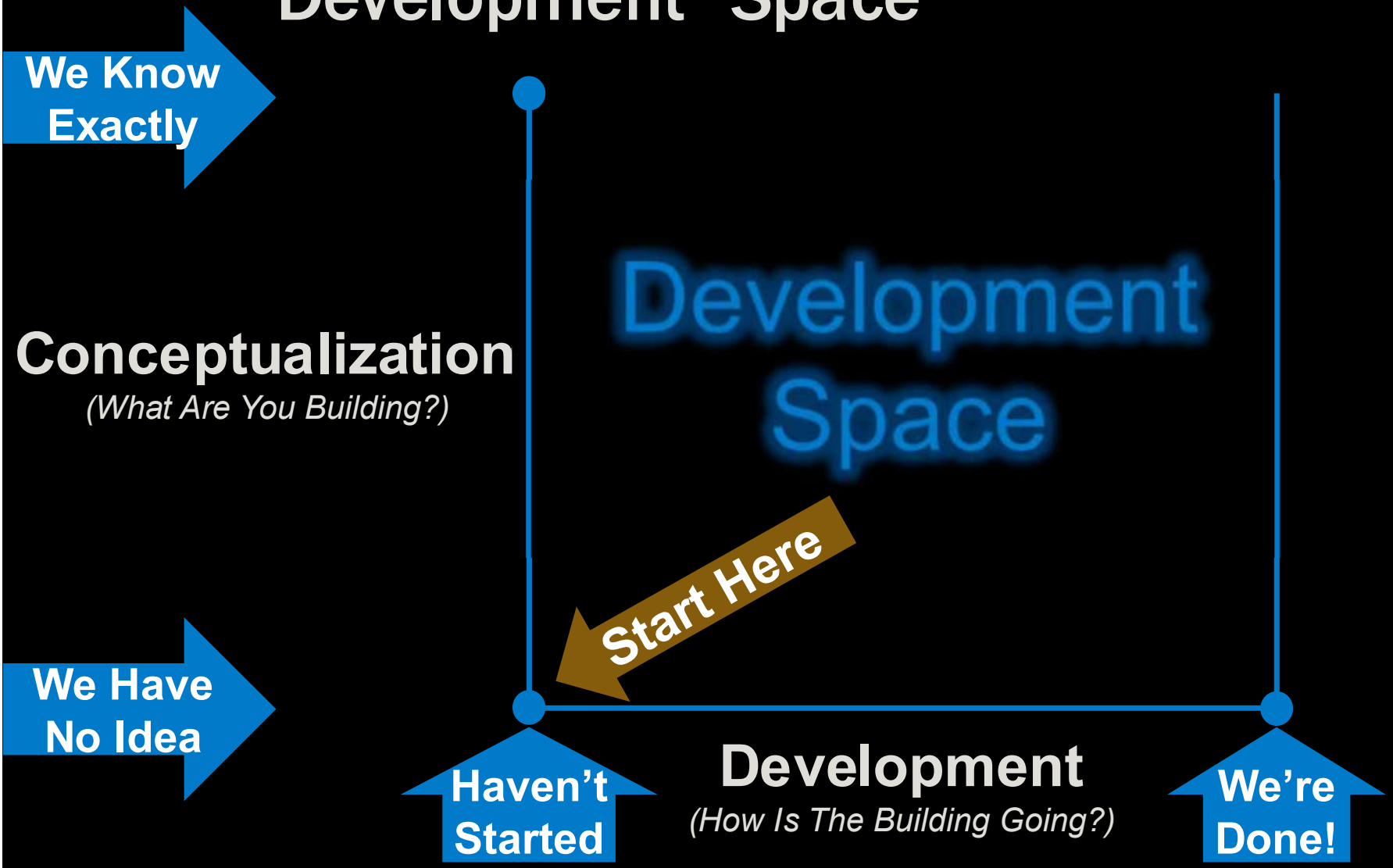
Iterative Development
*(a.k.a., "**Spiral Model**")*
https://en.wikipedia.org/wiki/Spiral_model

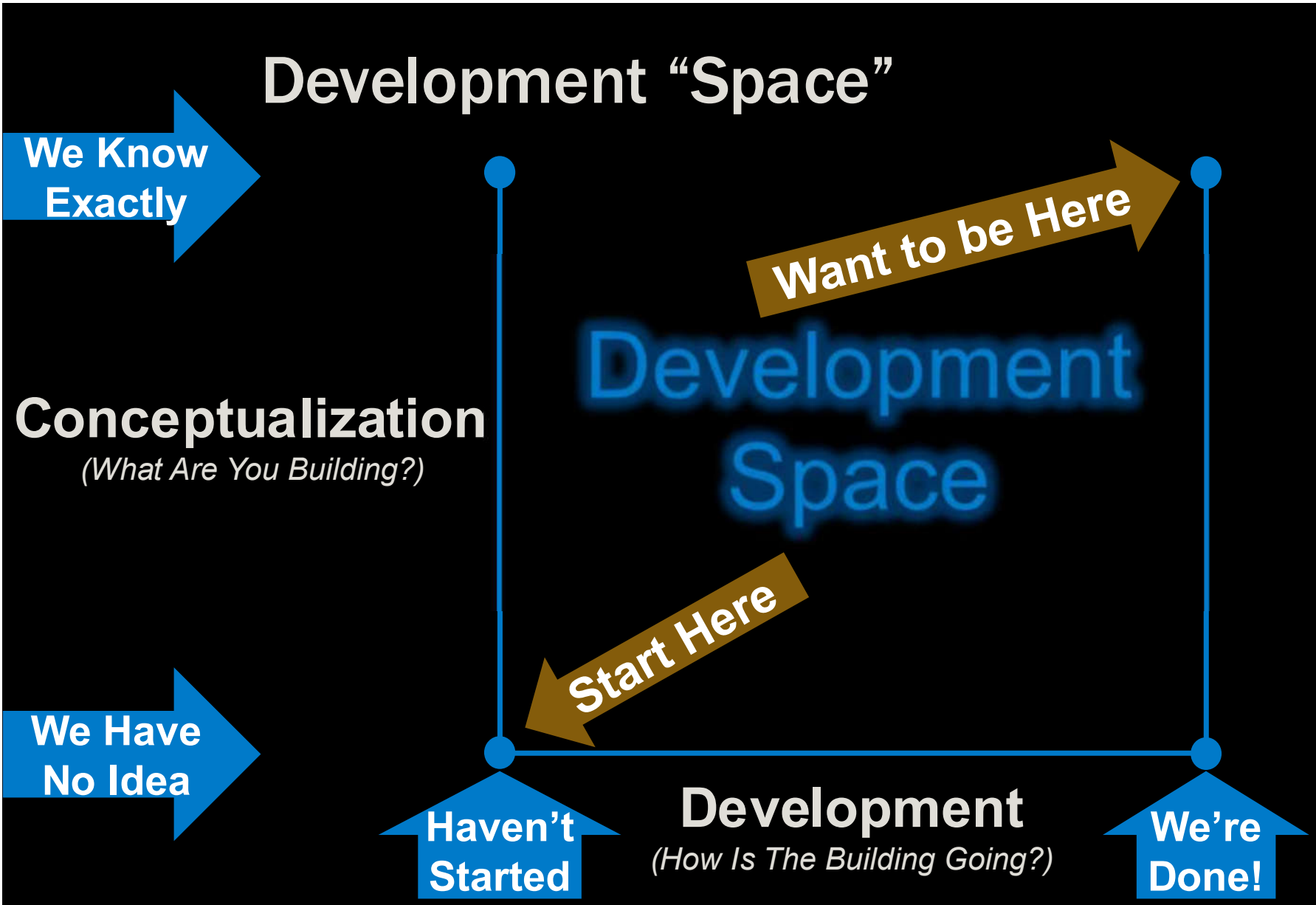
Development “Space”

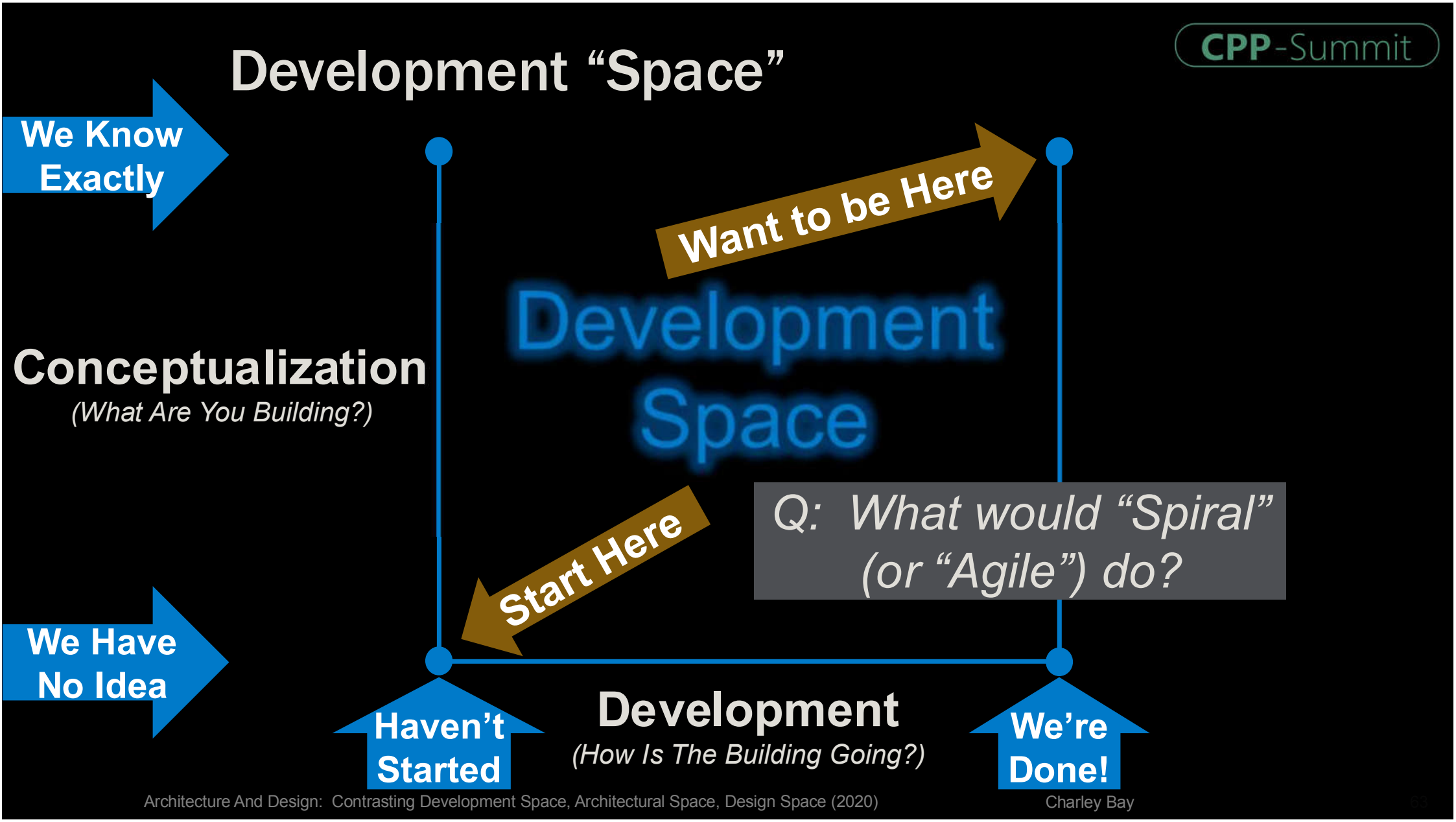
CPP-Summit



Development "Space"

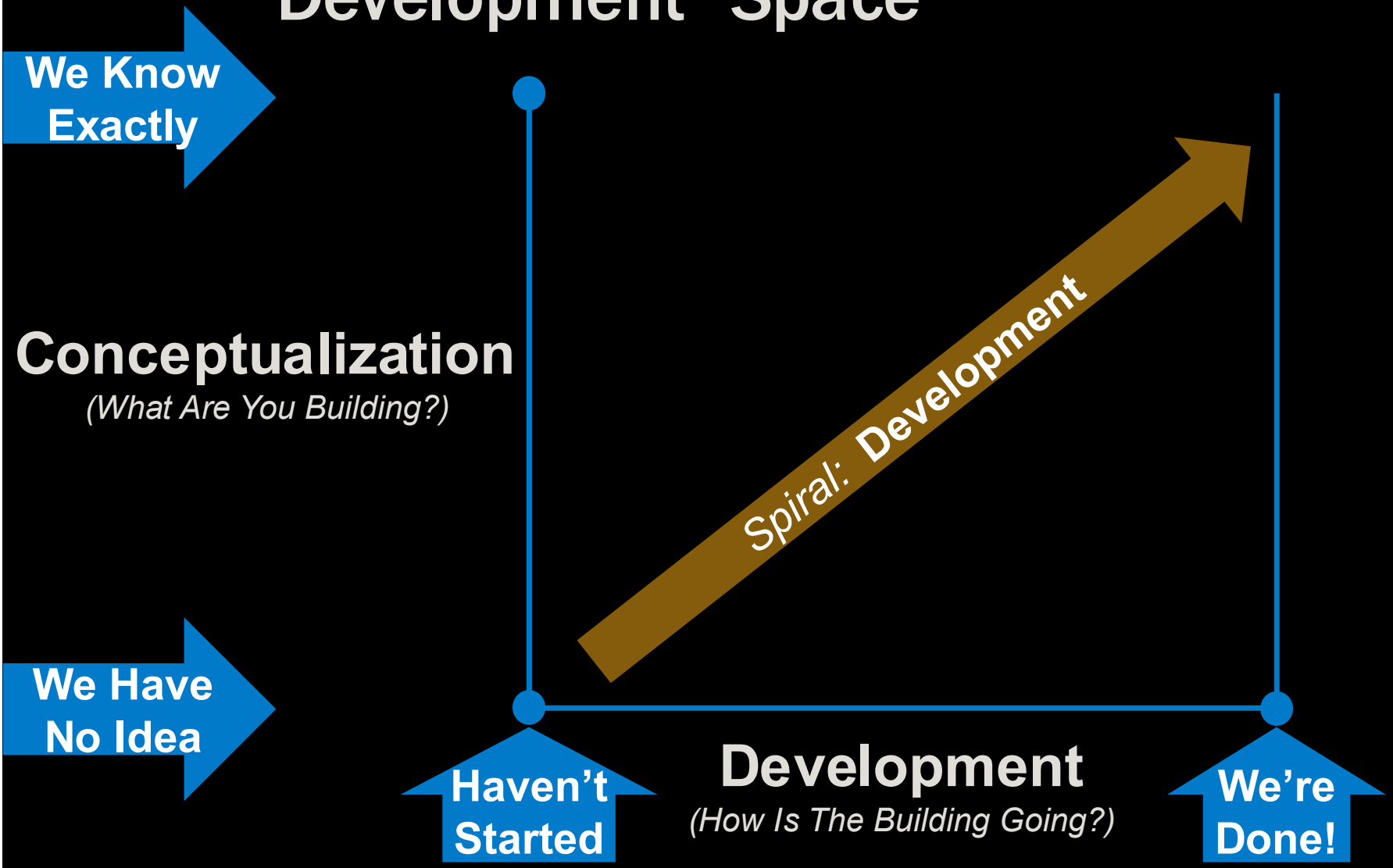


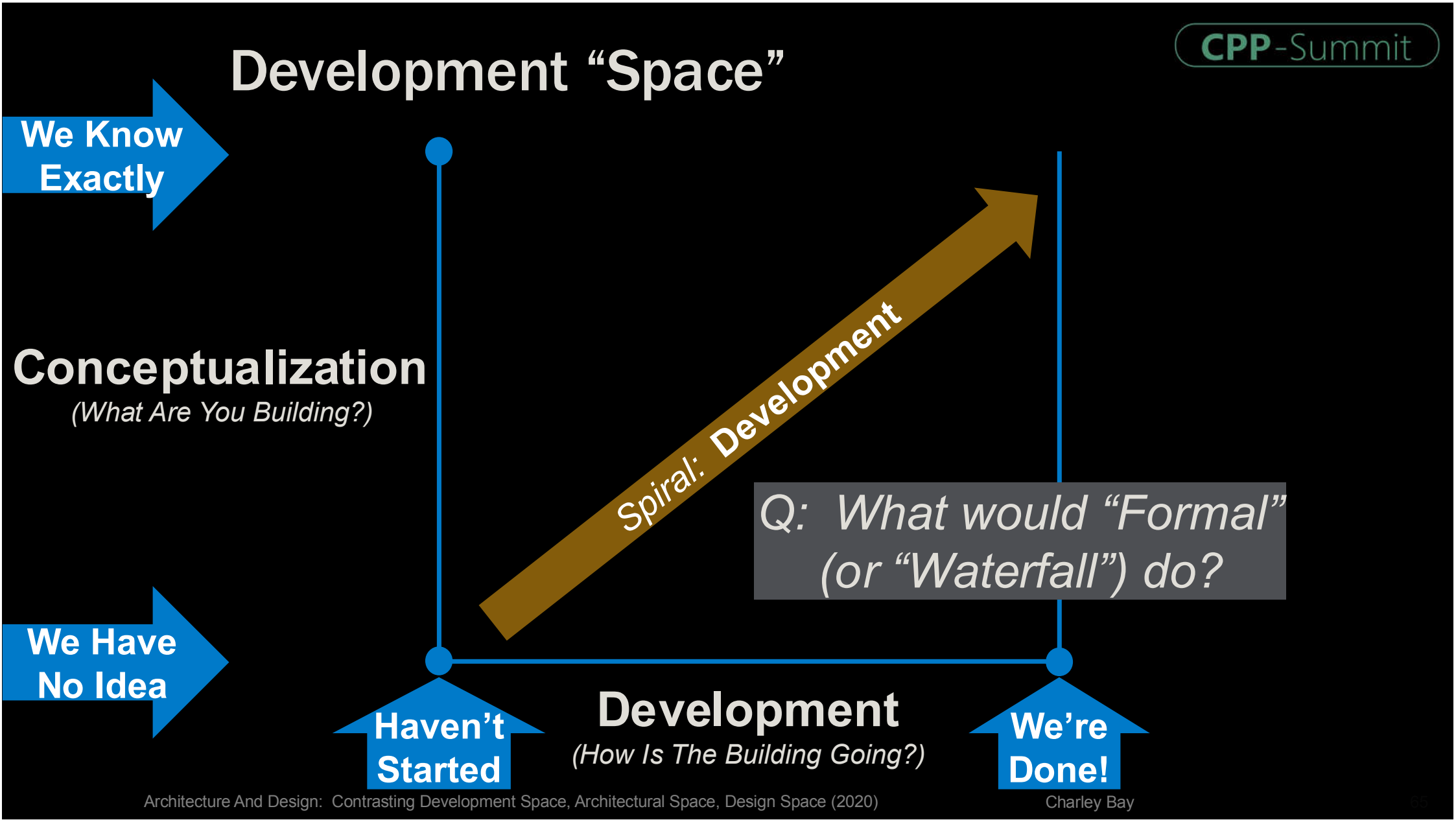

Development "Space"
We Know Exactly
Conceptualization
(What Are You Building?)
We Have No Idea
Want to be Here
Development Space
Start Here
Haven't Started
Development
(How Is The Building Going?)
We're Done!

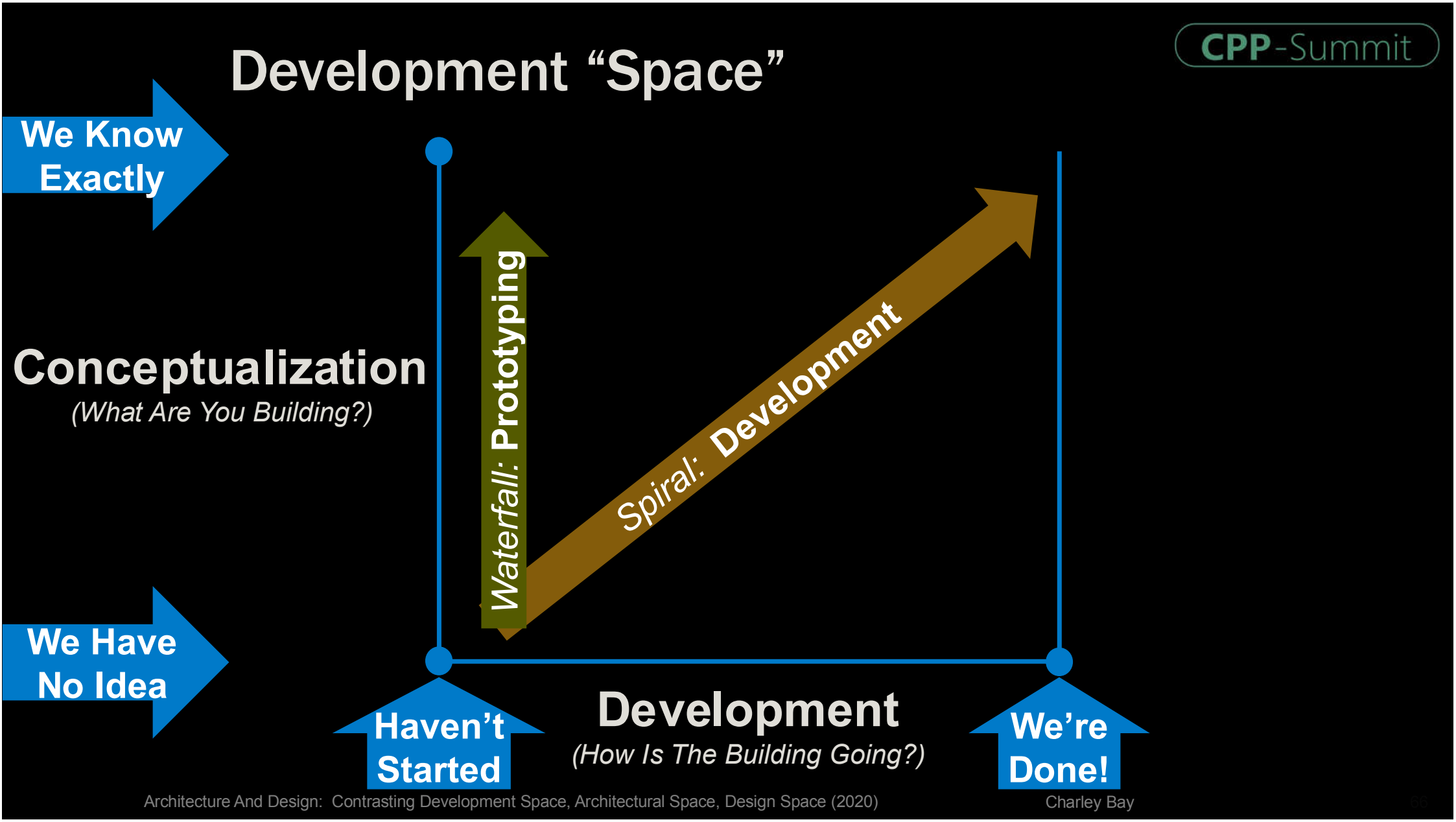

CPP-Summit
Development "Space"
We Know Exactly
Want to be Here
Development Space
Conceptualization
(What Are You Building?)
Start Here
Q: What would "Spiral" (or "Agile") do?
We Have No Idea
Haven't Started
Development
(How Is The Building Going?)
We're Done!

CPP-Summit

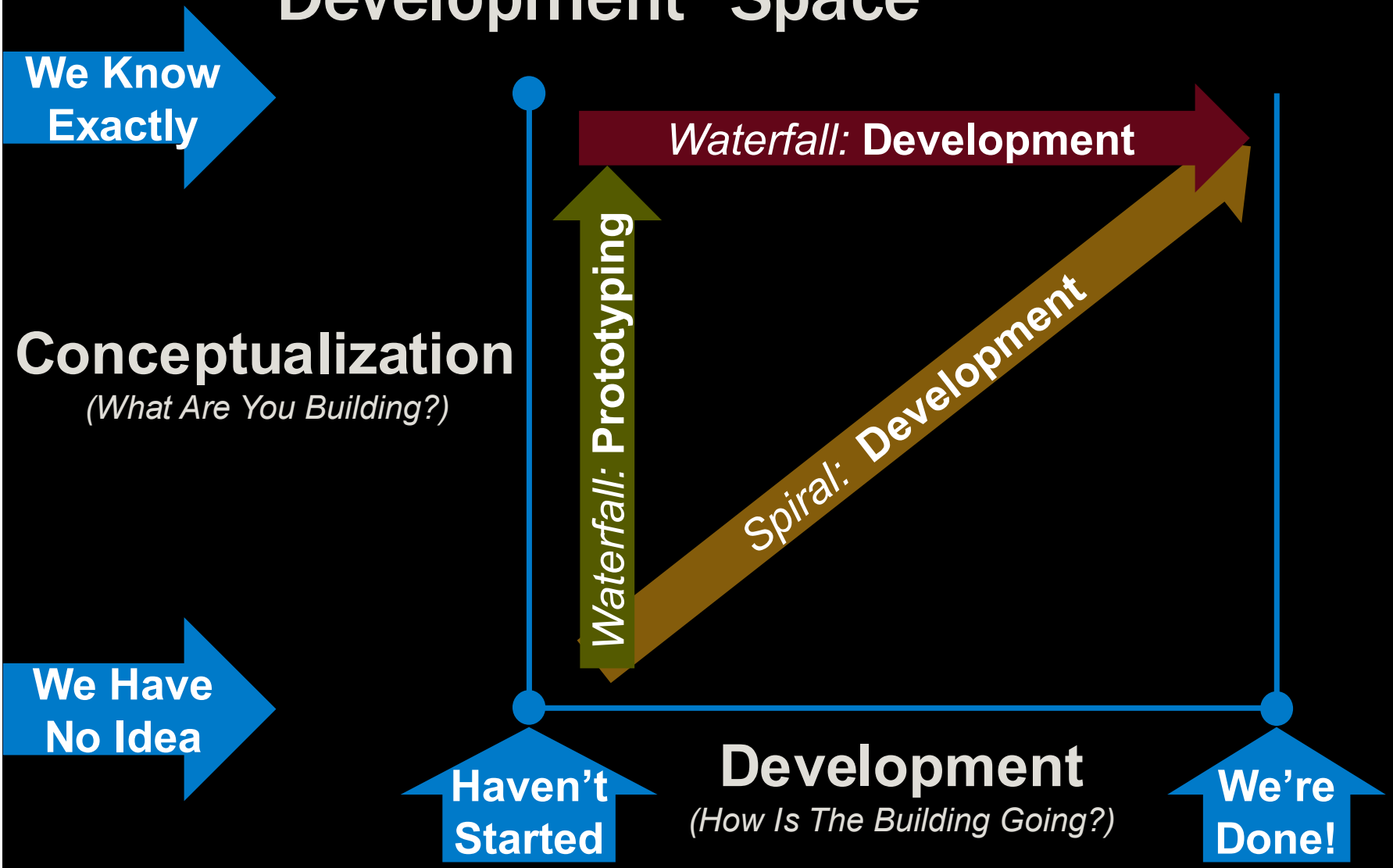
Development “Space”

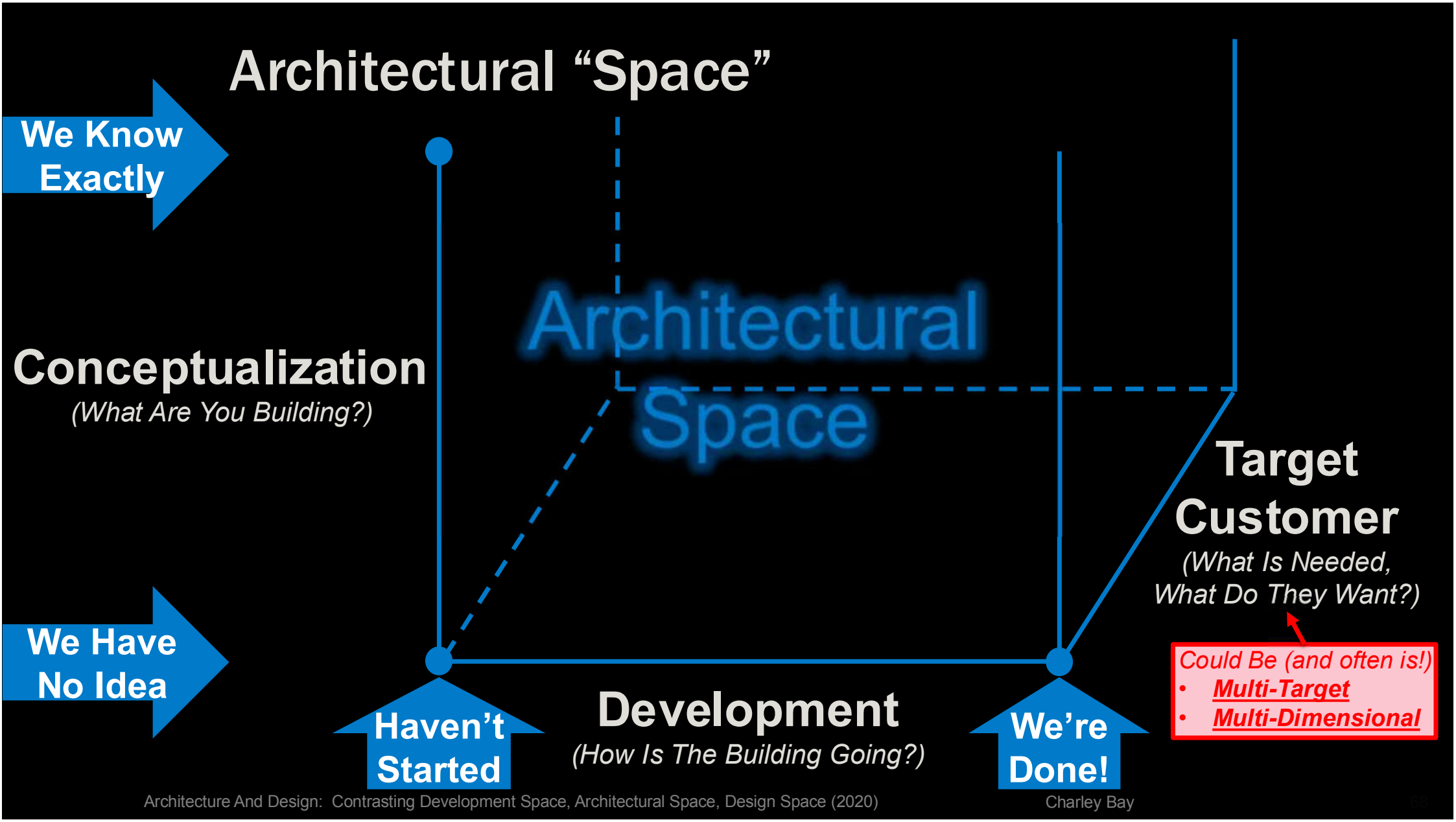


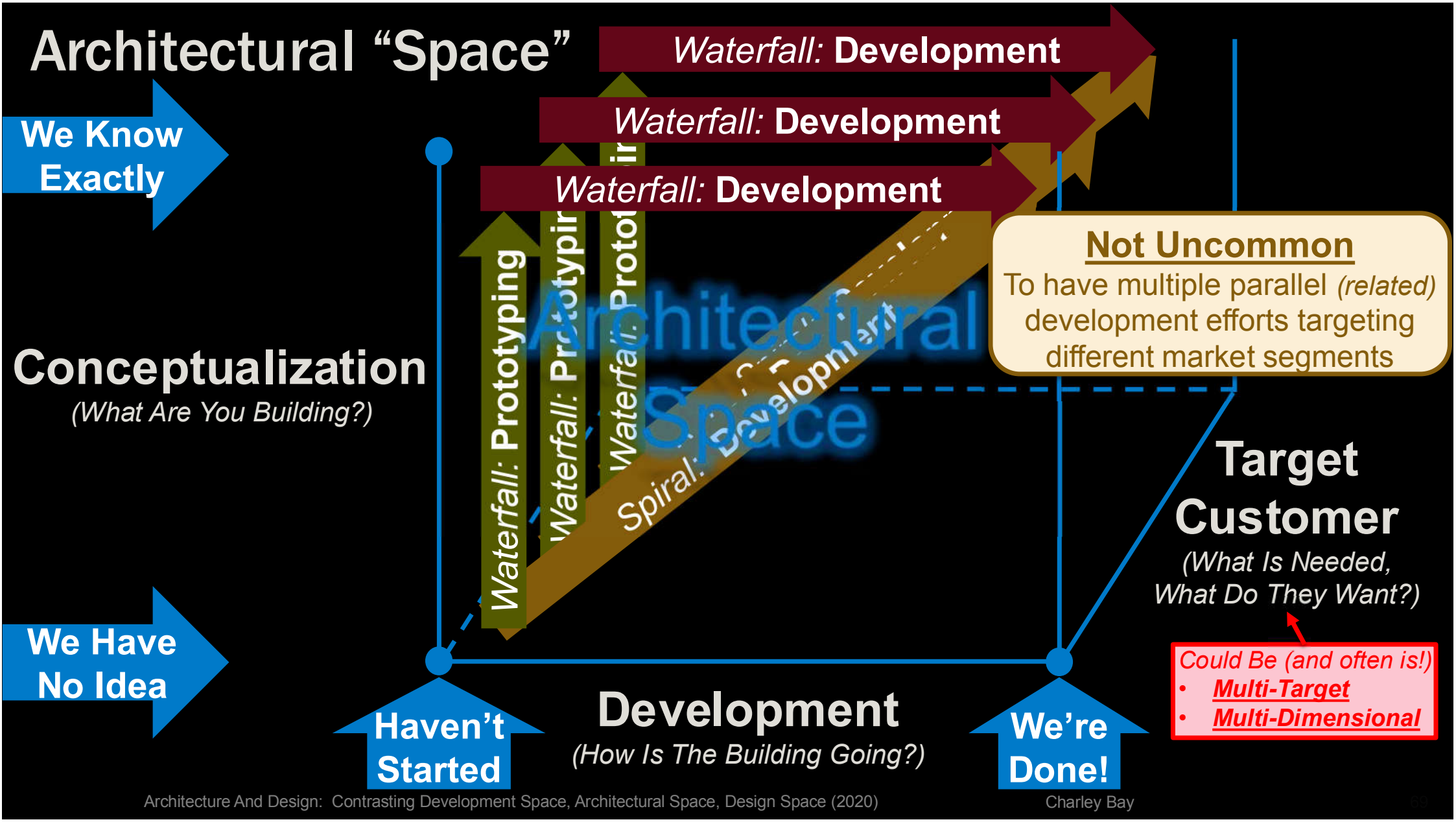

CPP-Summit
Development "Space"
We Know Exactly
Conceptualization
(What Are You Building?)
We Have No Idea
Spiral: Development
Q: What would "Formal" (or "Waterfall") do?
Haven't Started
Development
(How Is The Building Going?)
We're Done!

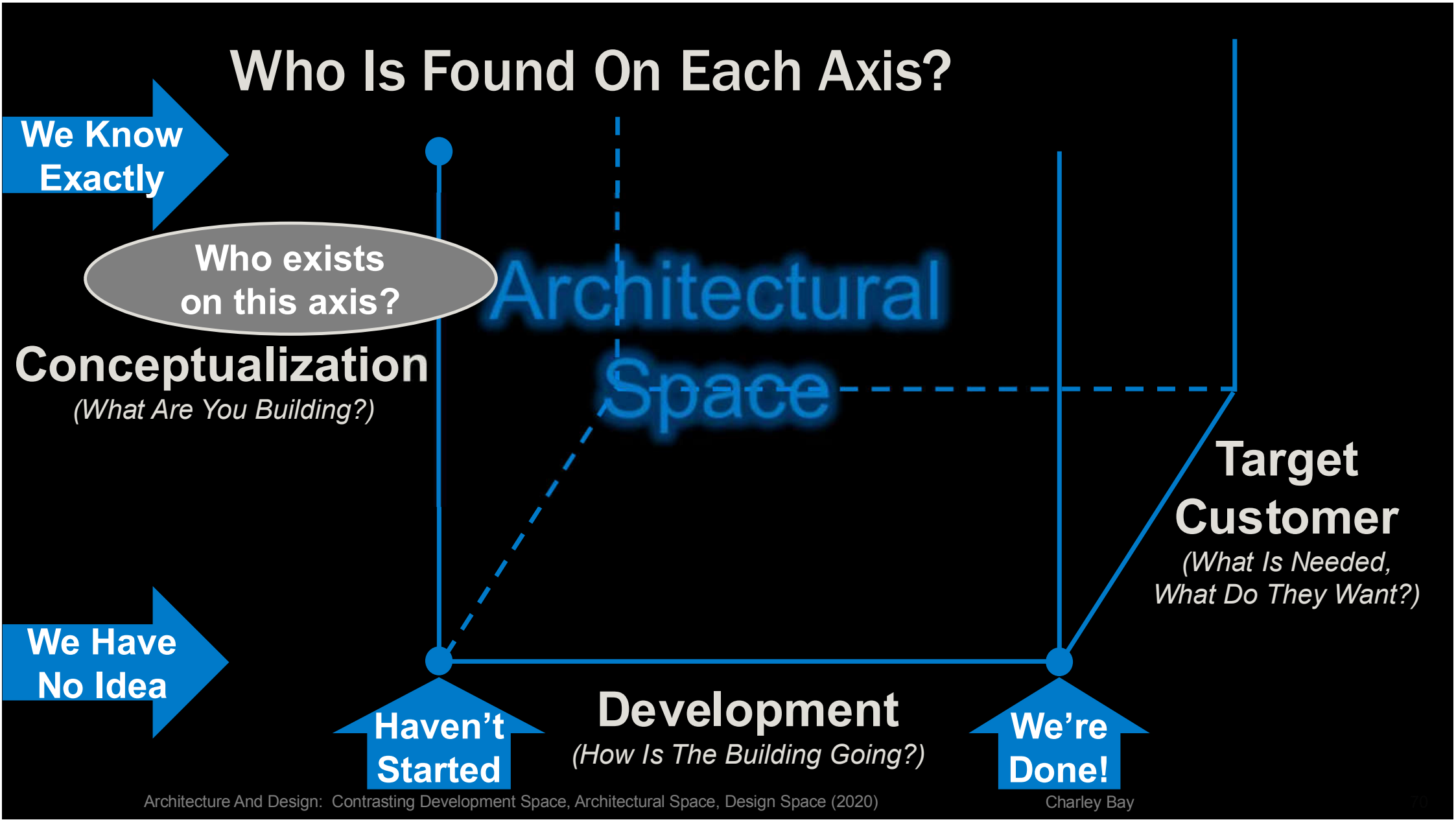

Development "Space"
We Know Exactly
Conceptualization
(What Are You Building?)
We Have No Idea
Waterfall: Prototyping
Spiral: Development
Haven't Started
Development
(How Is The Building Going?)
We're Done!

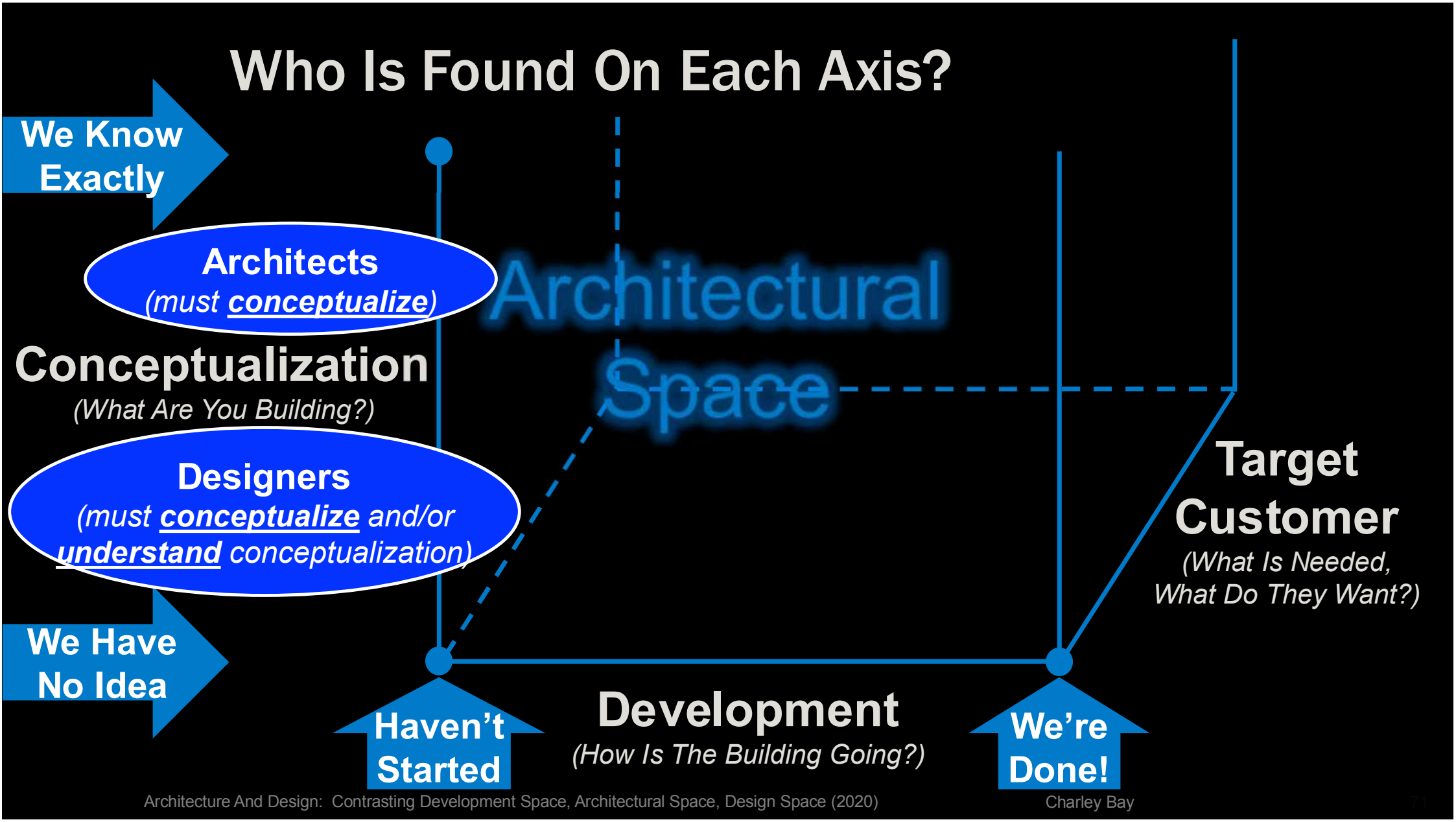
Development "Space"



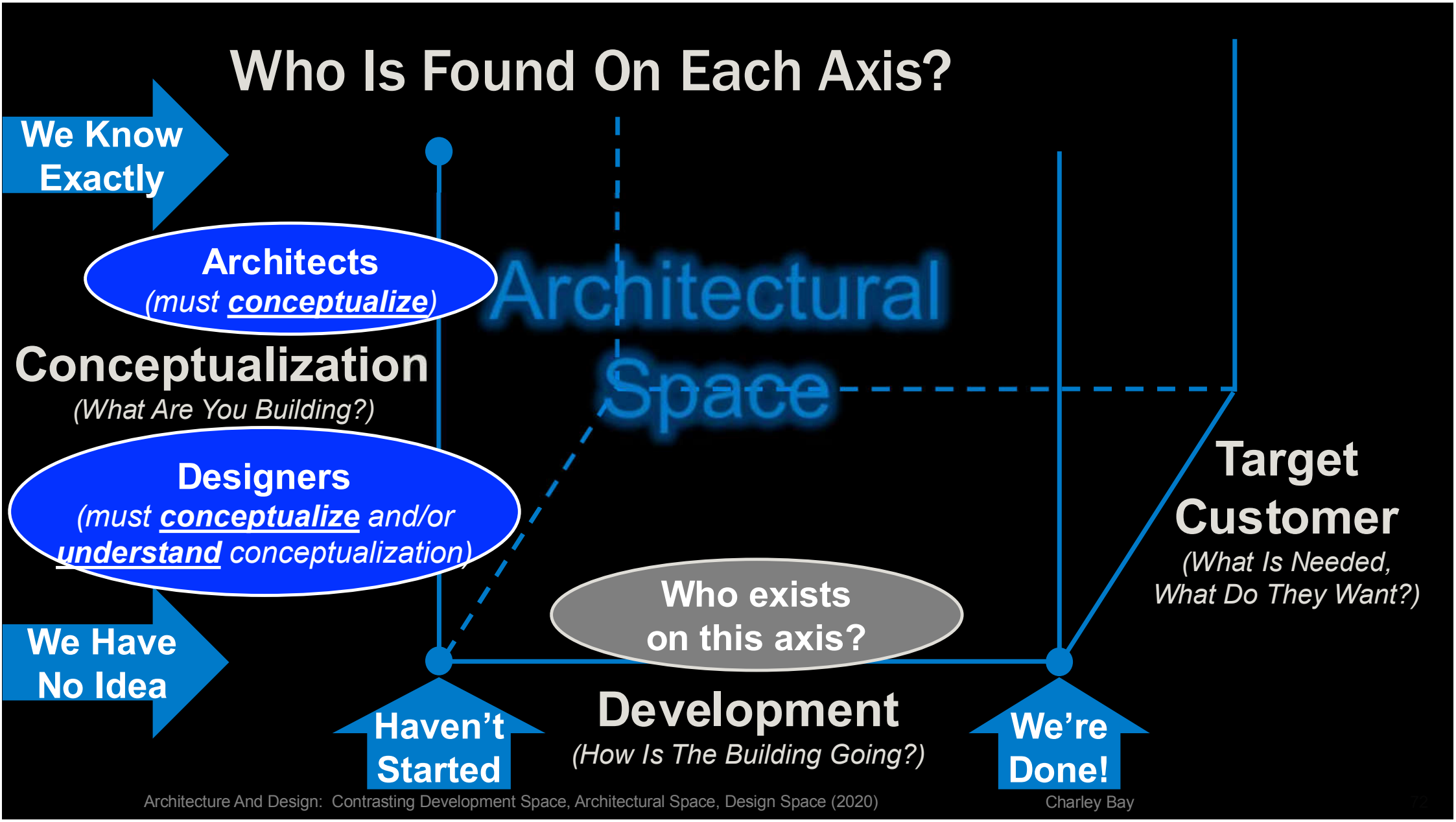

Architectural "Space"
We Know Exactly
Conceptualization
(What Are You Building?)
We Have No Idea
Architectural Space
Target Customer
(What Is Needed, What Do They Want?)
Could Be (and often is!)
• Multi-Target
• Multi-Dimensional
Haven't Started
Development
(How Is The Building Going?)
We're Done!

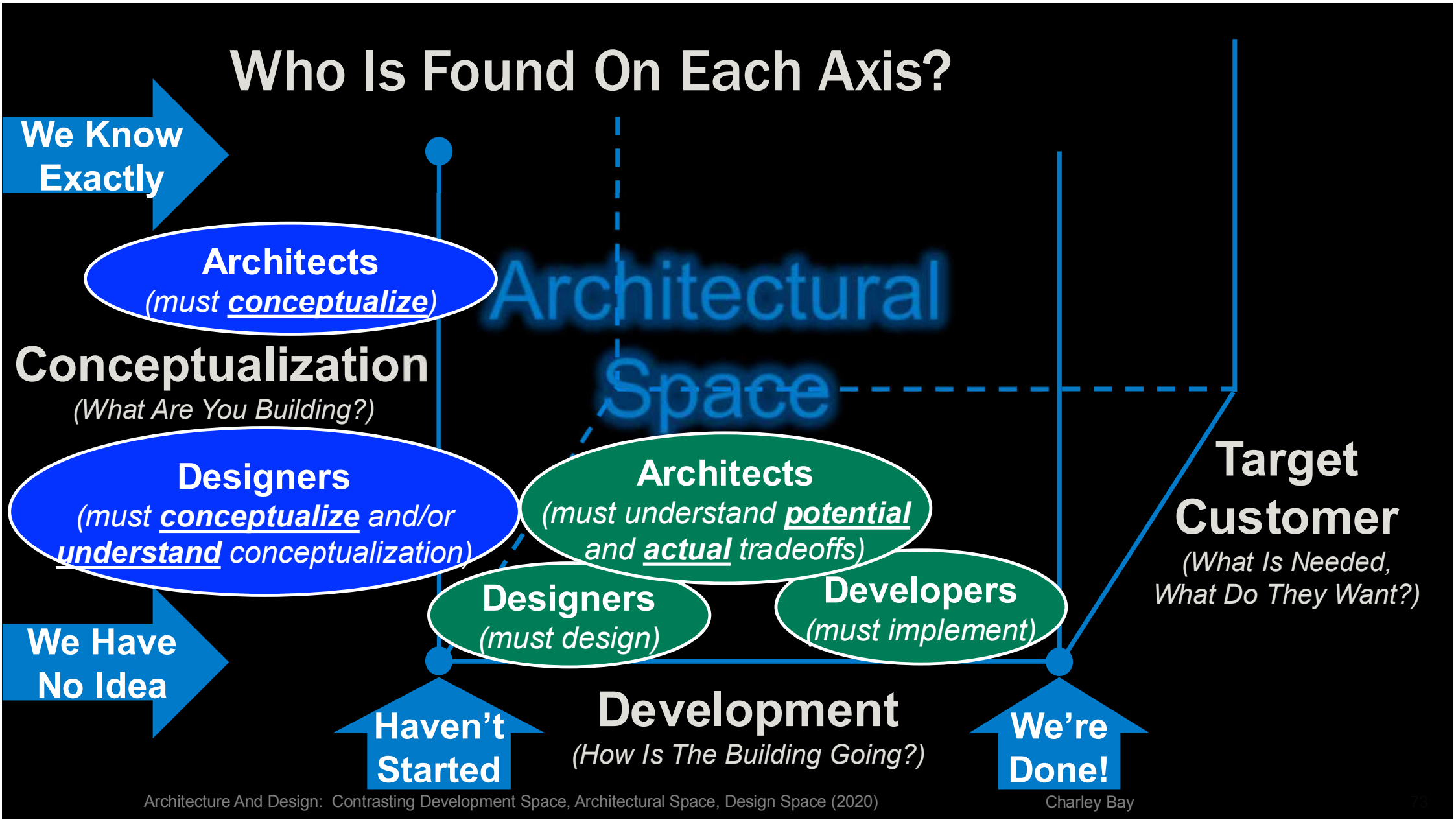

Architectural "Space"
We Know Exactly
We Have No Idea
Conceptualization
(What Are You Building?)
Waterfall: Development
Waterfall: Development
Waterfall: Development
Waterfall: Prototyping
Waterfall: Prototyping
Waterfall: Prototyping
Spiral: Development
Architectural Space
Not Uncommon
To have multiple parallel (related) development efforts targeting different market segments
Target Customer
(What Is Needed, What Do They Want?)
Could Be (and often is!)
• Multi-Target
• Multi-Dimensional
Haven't Started
Development
(How Is The Building Going?)
We're Done!

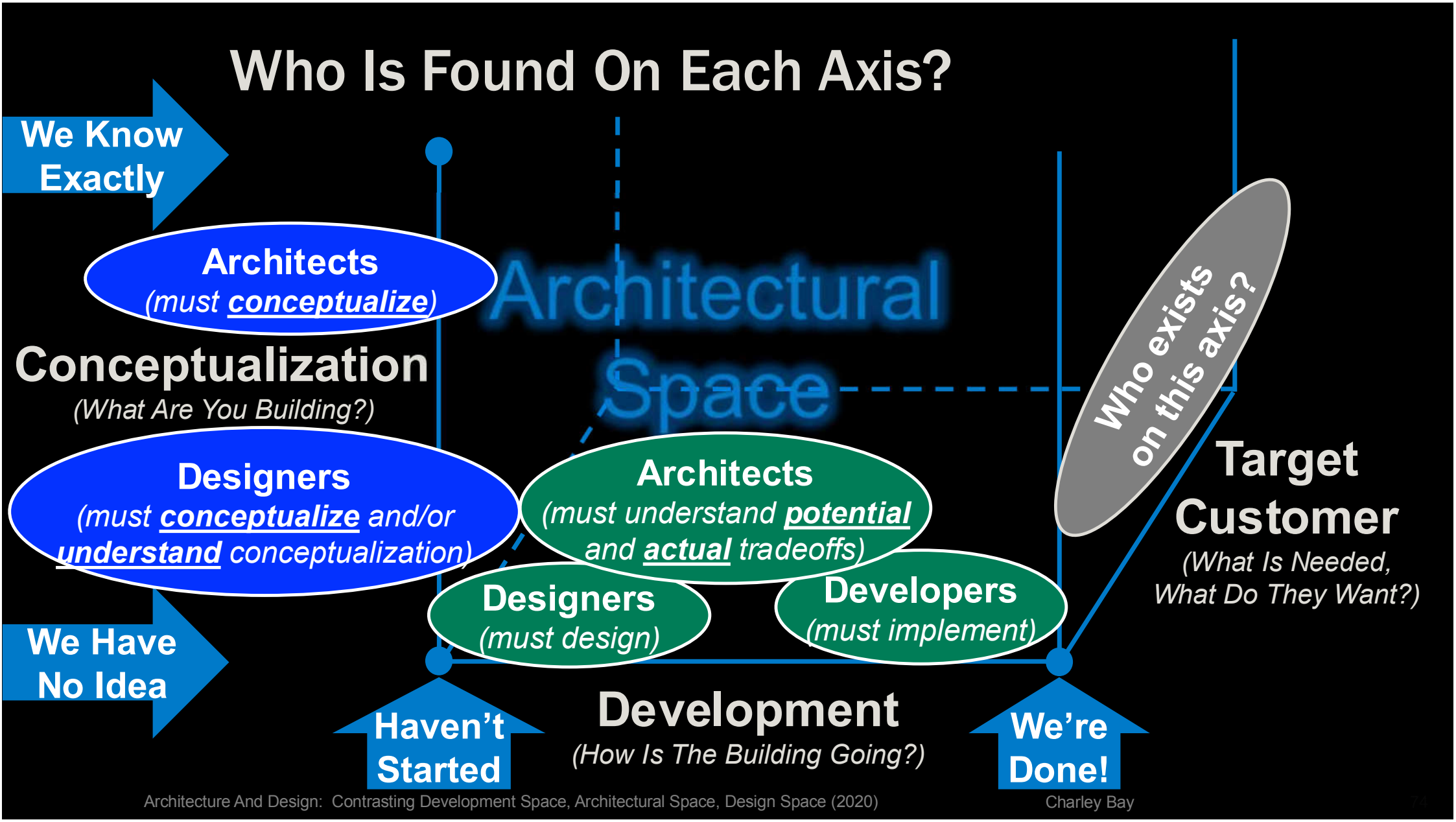

Who Is Found On Each Axis?
We Know Exactly
Who exists on this axis?
Architectural Space
Conceptualization
(What Are You Building?)
Target Customer
(What Is Needed, What Do They Want?)
We Have No Idea
Haven't Started
Development
(How Is The Building Going?)
We're Done!

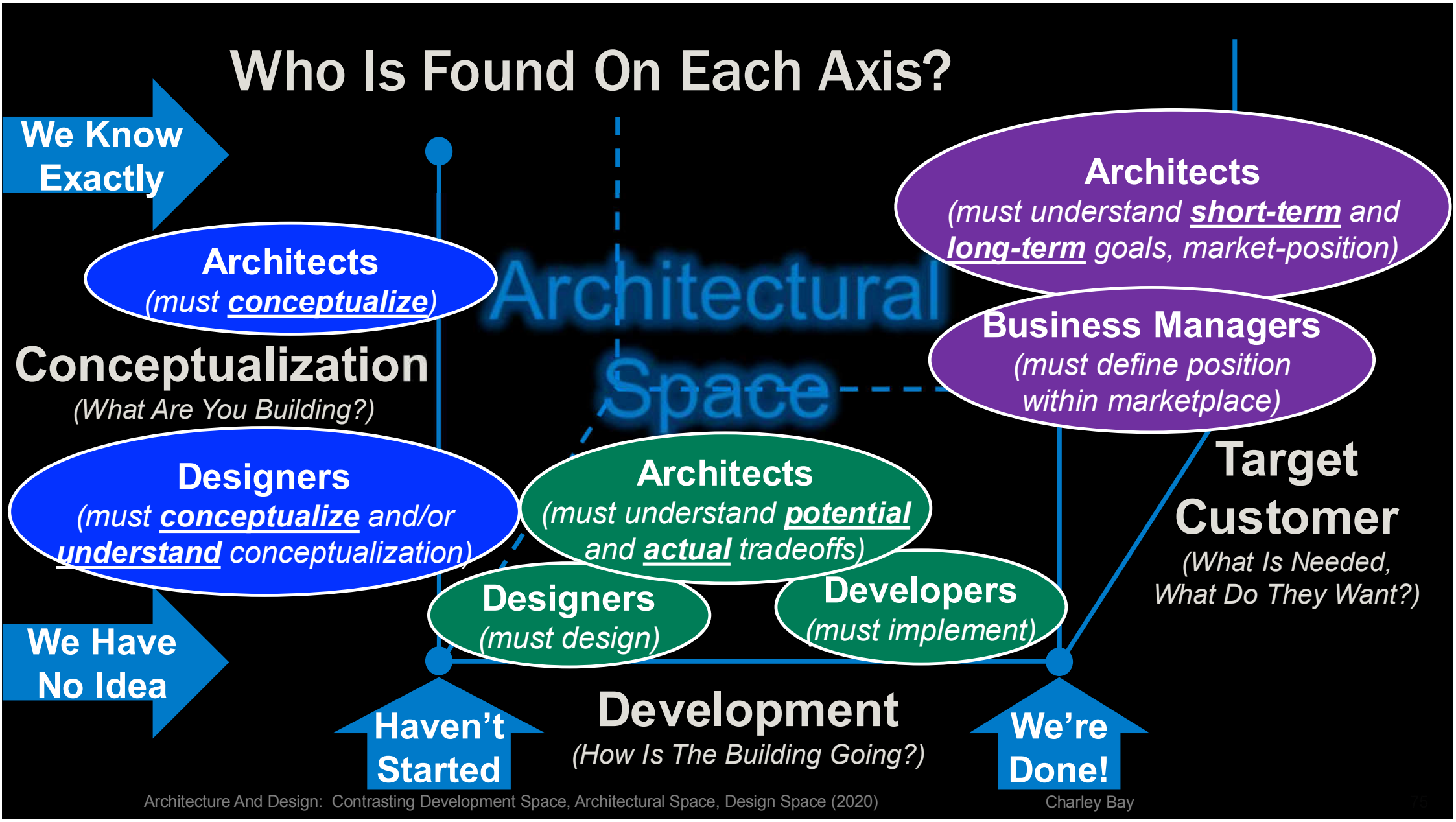


Architecture And Design: Contrasting Development Space, Architectural Space, Design Space (2020) Charley Bay

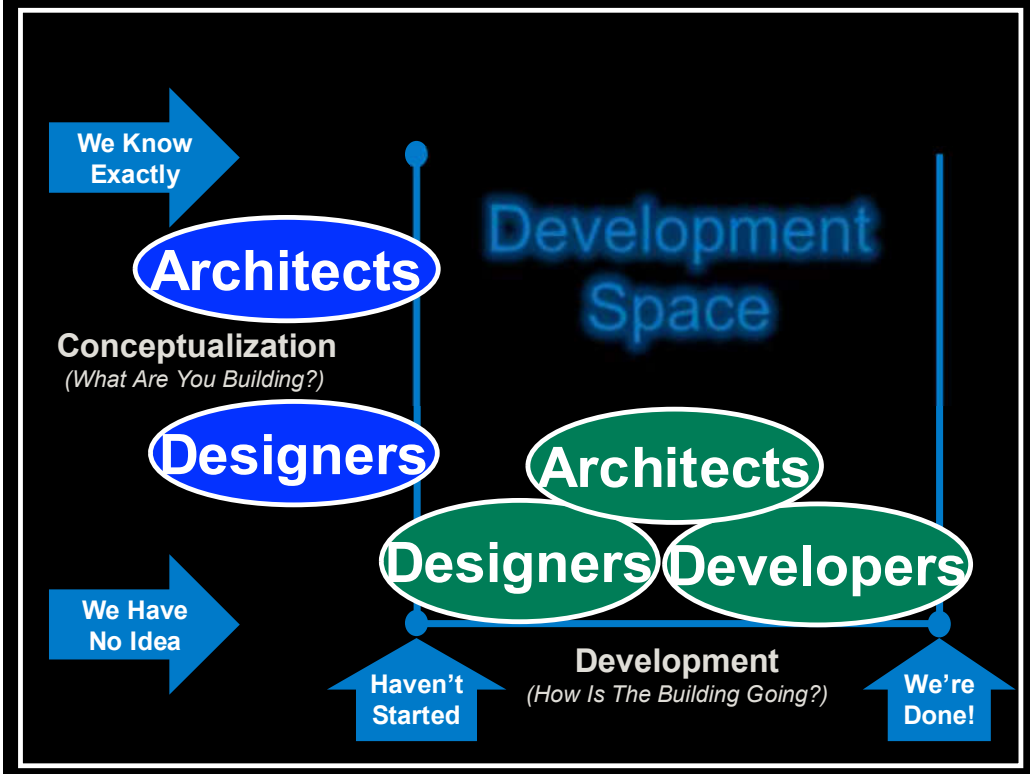

Who Is Found On Each Axis?
We Know Exactly
Architects
(must conceptualize)
Conceptualization
(What Are You Building?)
Designers
(must conceptualize and/or understand conceptualization)
We Have No Idea
Architectural Space
Who exists on this axis?
Target Customer
(What Is Needed, What Do They Want?)
Haven't Started
Development
(How Is The Building Going?)
We're Done!

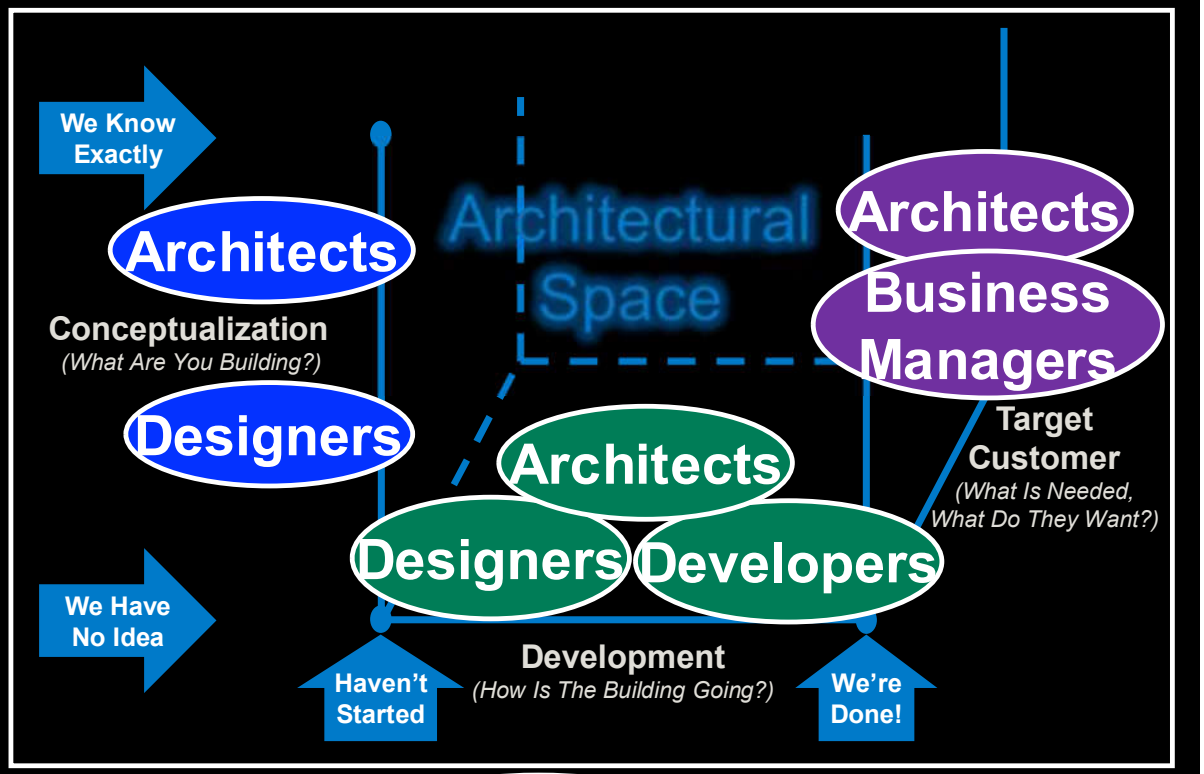

Who Is Found On Each Axis?
We Know Exactly
We Have No Idea
Architects
(must conceptualize)
Conceptualization
(What Are You Building?)
Designers
(must conceptualize and/or understand conceptualization)
Architectural Space
Architects
(must understand potential and actual tradeoffs)
Designers
(must design)
Developers
(must implement)
Target Customer
(What Is Needed, What Do They Want?)
Haven't Started
Development
(How Is The Building Going?)
We're Done!

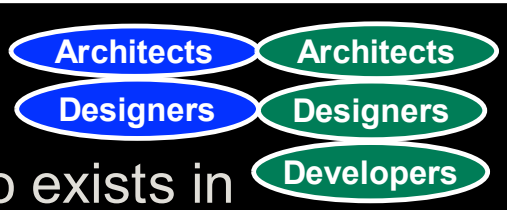

Who Is Found On Each Axis?
We Know Exactly
We Have No Idea
Architects
(must conceptualize)
Conceptualization
(What Are You Building?)
Designers
(must conceptualize and/or understand conceptualization)
Architectural Space
Architects
(must understand potential and actual tradeoffs)
Designers
(must design)
Developers
(must implement)
Who exists on this axis?
Target Customer
(What Is Needed, What Do They Want?)
Haven't Started
Development
(How Is The Building Going?)
We're Done!


Who Is Found On Each Axis?
We Know Exactly
We Have No Idea
Architects
(must conceptualize)
Conceptualization
(What Are You Building?)
Designers
(must conceptualize and/or understand conceptualization)
Architectural Space
Architects
(must understand potential and actual tradeoffs)
Designers
(must design)
Developers
(must implement)
Development
(How Is The Building Going?)
Haven't Started
We're Done!
Architects
(must understand short-term and long-term goals, market-position)
Business Managers
(must define position within marketplace)
Target Customer
(What Is Needed, What Do They Want?)

Development vs. Architectural "Space"

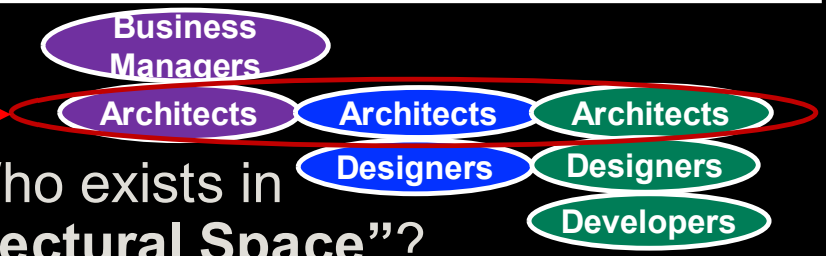






Who exists in "**Development Space**"?

Only Architect
bridges conceptualization, development, target customer



Who exists in "**Architectural Space**"?

Reuse Across Product Lines

CPP-Summit

- The **more variations** in product offering *(or target customer)*, the more motivation for **reuse**

Product *(def):* A unit offered for delivery

Product Line *(def):* A collection of related-but-unique product offerings

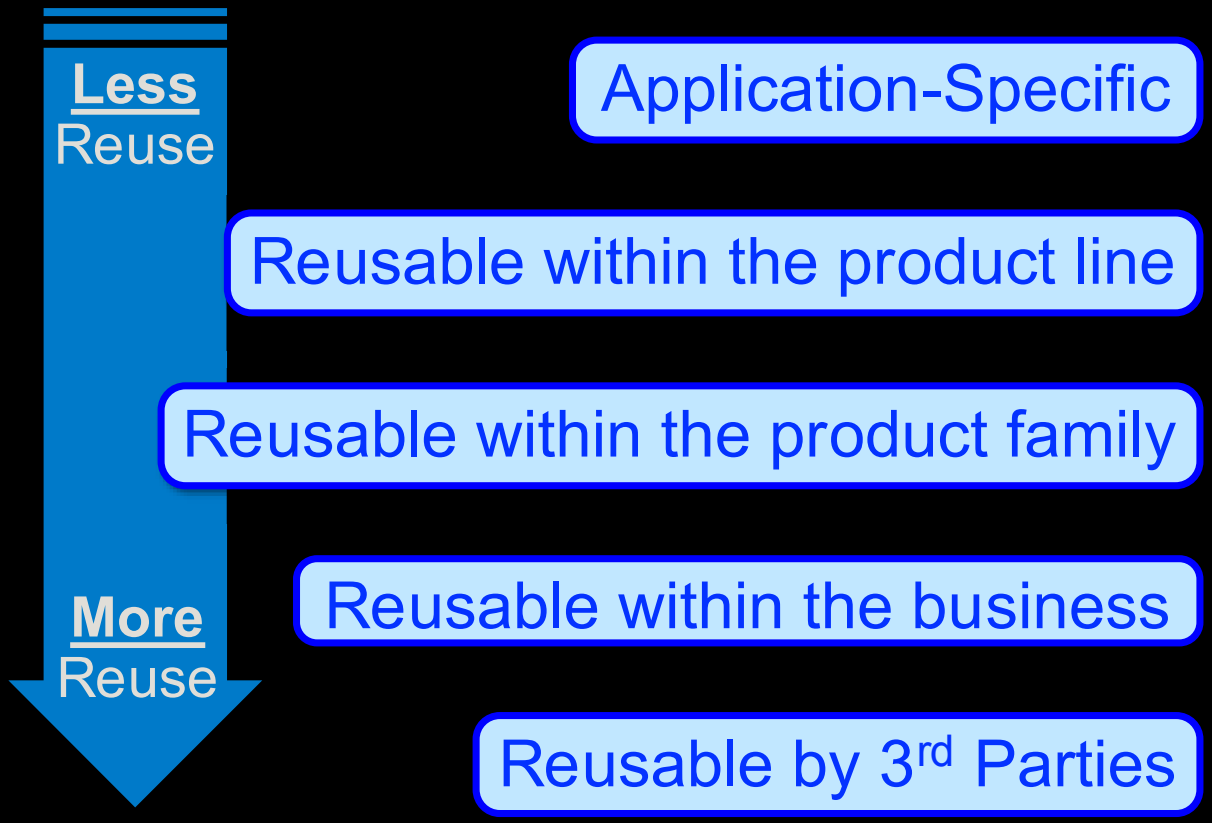
Product Family *(def):* A collection of related-but-unique product lines

Market Gap Analysis *(def):* Compared placement of your product families within the superset of product families offered in your market segment

Each "related-but-unique" motivates some form of **architectural reuse**

Physical Dimensions Within Architectural Space

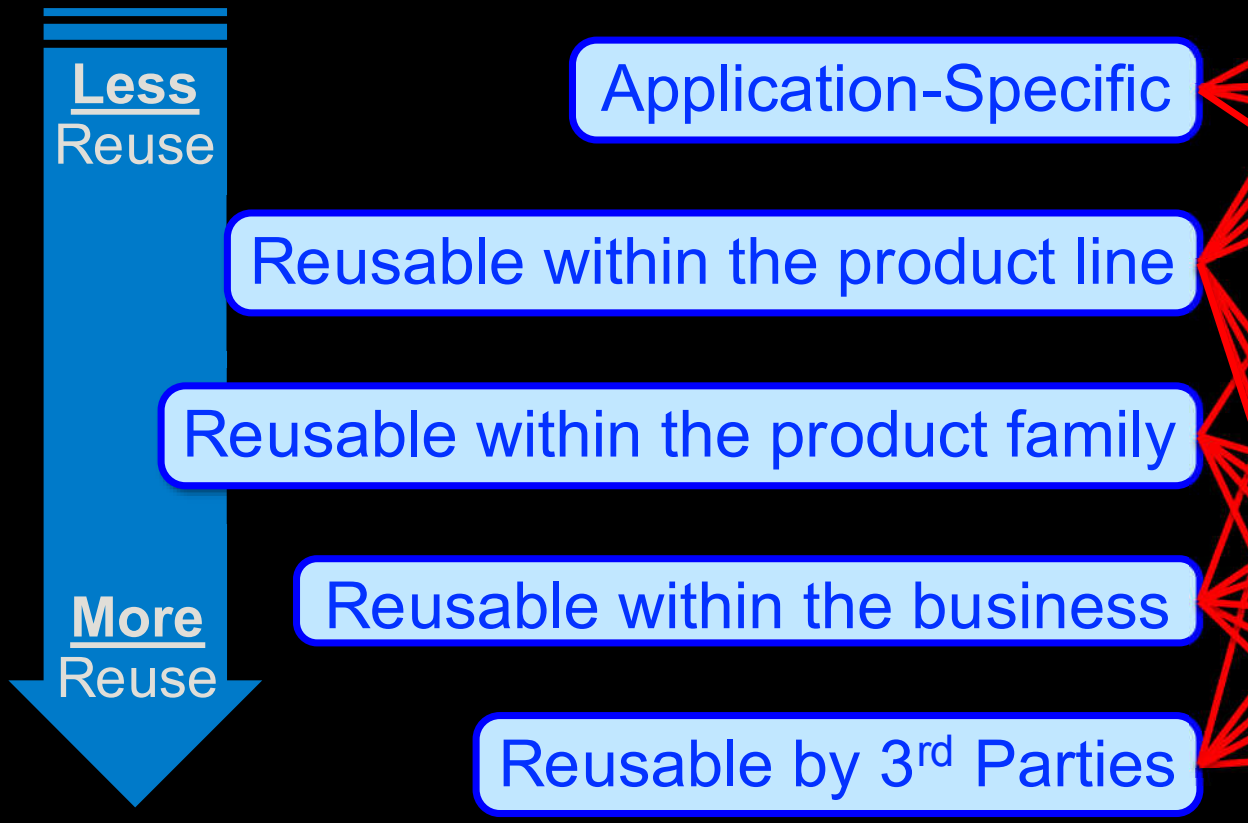
- *(Reuse)* **What code is…**



- Specific Considerations:
  - User interface
  - Subsystem Configuration
  - System Configuration
  - Business Logic Invariants:
    - Types
    - Processing Models
      - Control Flows
      - Data Flows
  - Serviceability / Support Interfaces
  - Domain-Specific Data Handling
  - Logging
  - Serialization
  - RPC, Distributed processing models

Physical Dimensions Within Architectural Space

- *(Reuse)* **What code is…**



- Specific Considerations:
  - User interface
  - Subsystem Configuration
  - System Configuration
  - Business Logic Invariants:
    - Types
    - Processing Models
      - Control Flows
      - Data Flows
  - Serviceability / Support Interfaces
  - Domain-Specific Data Handling
  - Logging
  - Serialization
  - RPC, Distributed processing models

Reuse Has A Cost

Importance varies among your diverse interested parties *(must **rank** and **balance** priorities!)*

- **Ties developer progression**
  - *Good:* Promotes **consistency**, "everybody knows how it always works"
  - *Bad:* Forces same solution to different problem, **decreases innovation**
- Can **slow or speed development**
  - *Good:* Every Developer **benefits** from library update
  - *Bad:* Every Developer **waits** on library update
- Can **raise or lower cost**
  - *Good:* Lowers cost of new systems, provides **economies of scale**
  - *Bad:* **Cannot "just fix it"** in one system, because of behavior impact in other systems *(higher coupling adds complexity, time, and risk)*

Who cares? ***Managers***

Who cares? ***Security Professionals***

Who cares? ***Innovators***

Who cares? ***3rd Parties***

Who cares? ***Big Companies***

Who cares? ***Support Engineers***

Mature, Large organizations tend to prioritize for: **Consistency** and **Maintainability**

Thank you!
for listening

Questions?