Building Really Big Systems with Lean-Agile Practices and SAFe

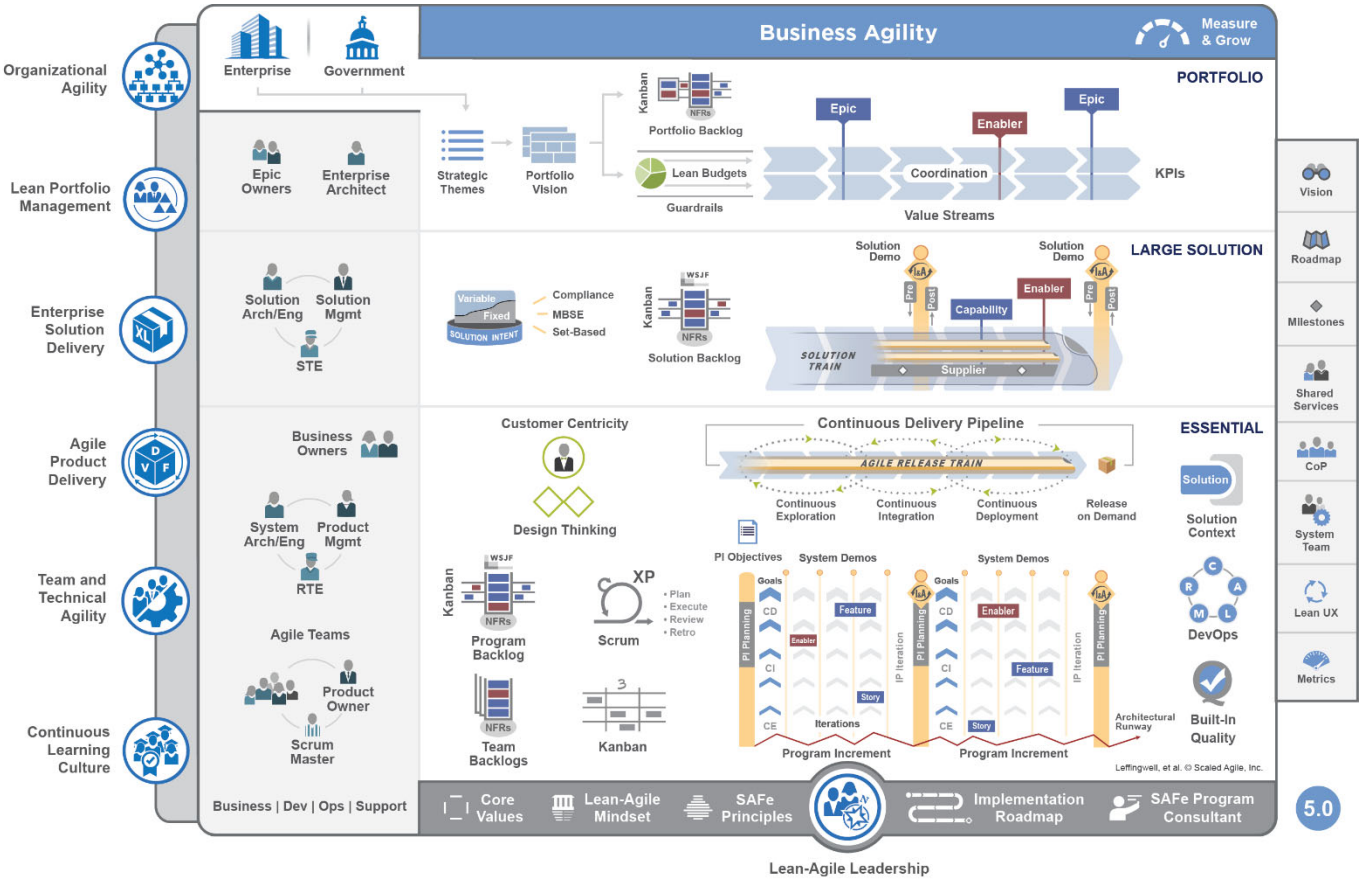
Harry Koehnemann
SAFe Fellow, SPCT

Learning Objectives

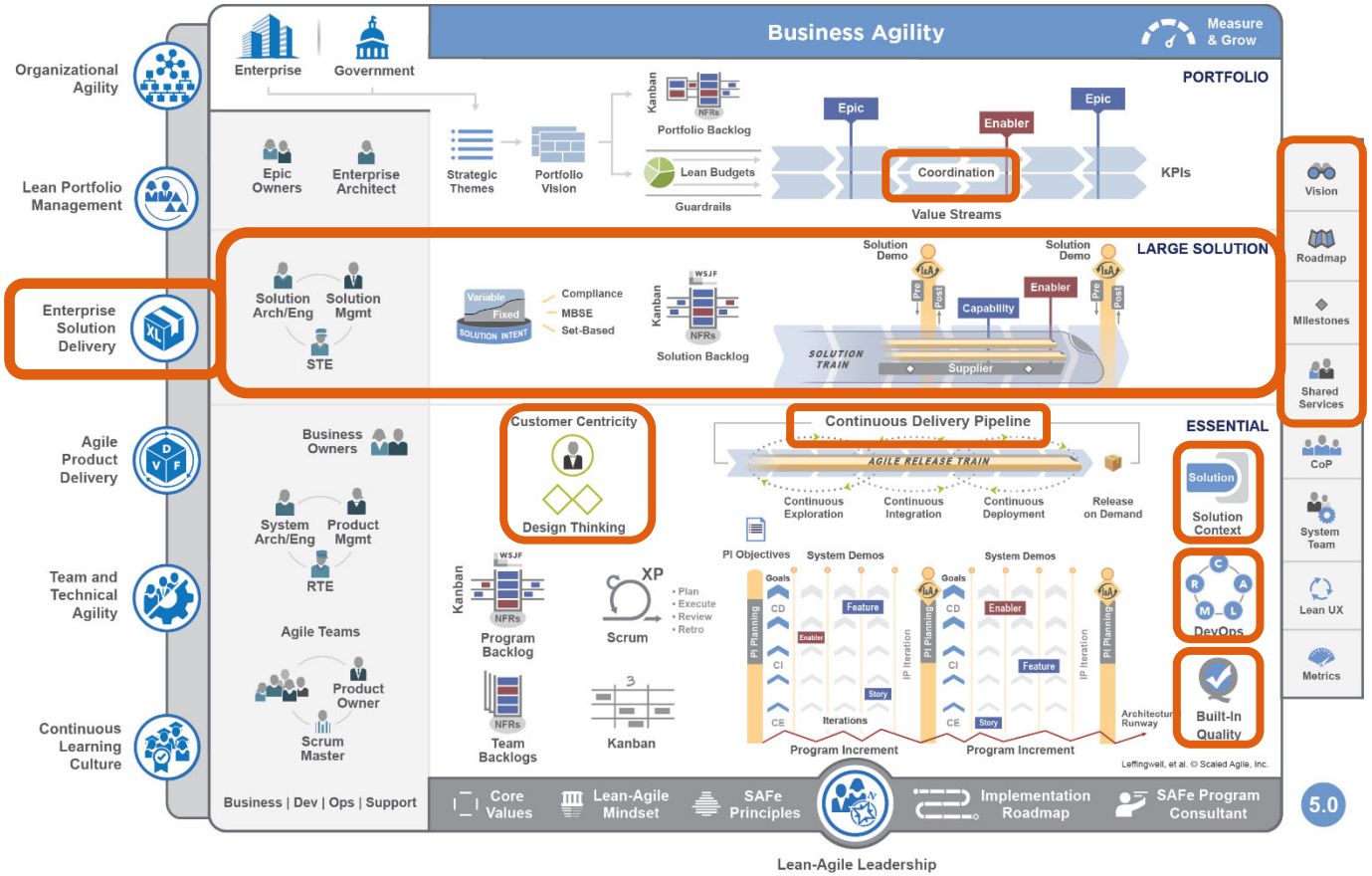
- Introduction to Really Big Systems in SAFe
- Apply 7 practices to building big systems using SAFe

Introduction to Really Big Systems in SAFe

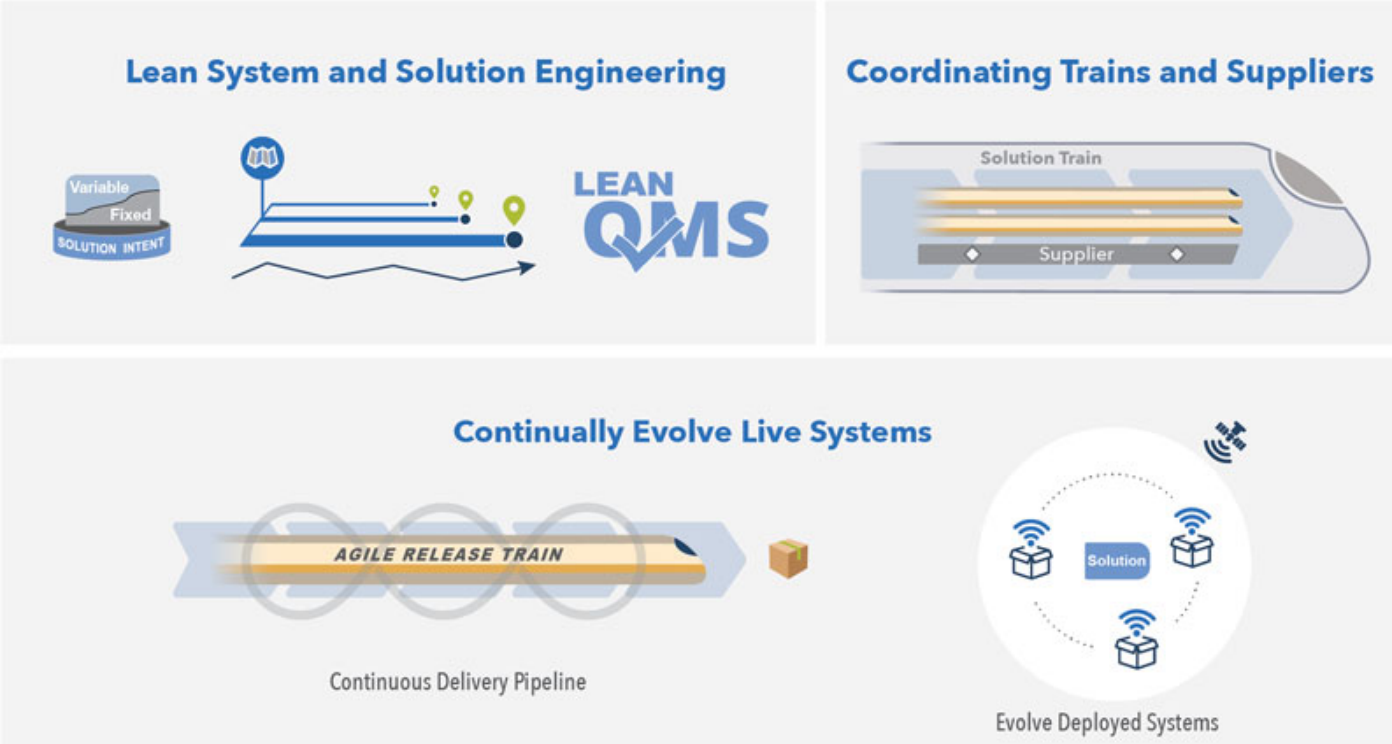
Introduction to SAFe 5.0



Applying building big systems to SAFe 5.0



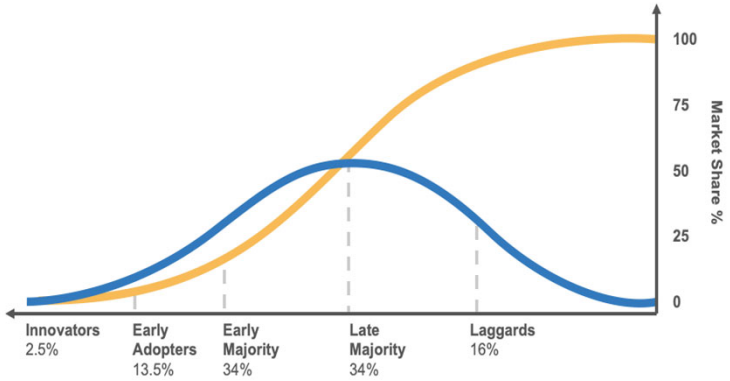
SAFe Enterprise Solution Delivery competency

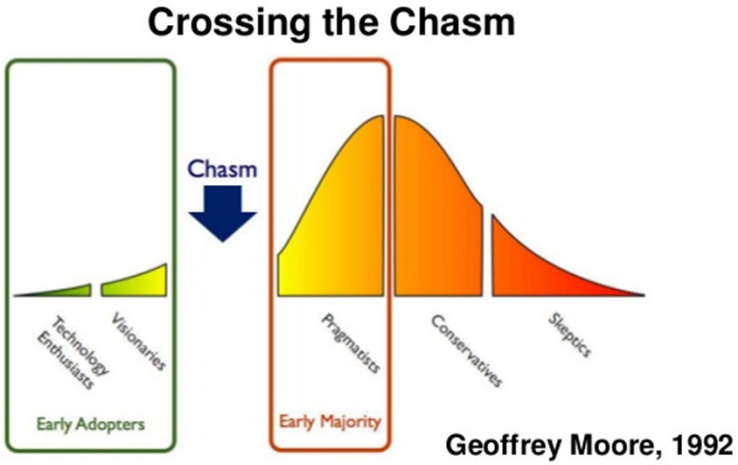


Nine practices for building really big systems

- Lean Systems Engineering
  1. Continually refine the fixed/variable Solution Intent
  2. Apply multiple planning horizons
  3. Architect for scale, modularity, releasability, and serviceability
  4. Continually address compliance concerns
- Coordinating Trains and Suppliers
  5. Build and integrate solution components and capabilities with ARTs and Solution Trains
  6. Apply ‘continuish’ integration
  7. Manage the supply chain with systems of systems thinking
- Continually Evolve Live Systems
  8. Build a Continuous Delivery Pipeline
  9. Evolve deployed systems

The lifecycle of product evolution





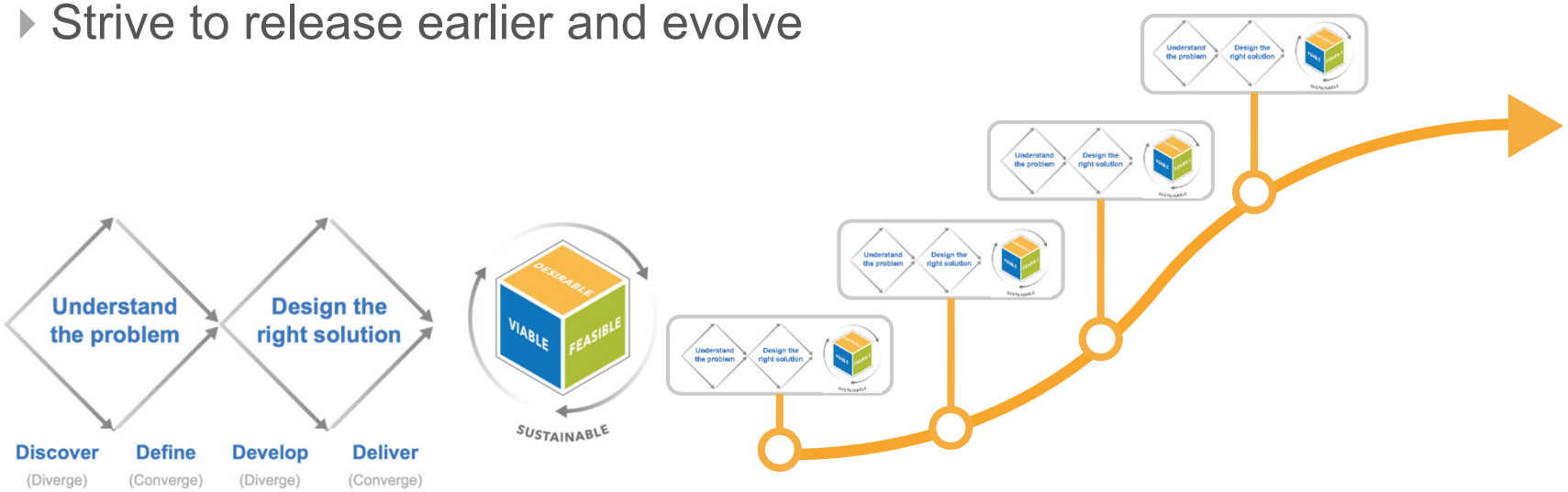
Innovations follow a predictable curve of growth known as the ‘S-shaped’ curve of adoption.

Geoffrey Moore noted that many technology products face a ‘chasm’ between the expectations and requirements of early adopters and the rest of the market.

Moore, Geoffrey. *Crossing the Chasm*. Harper Business, 2006.

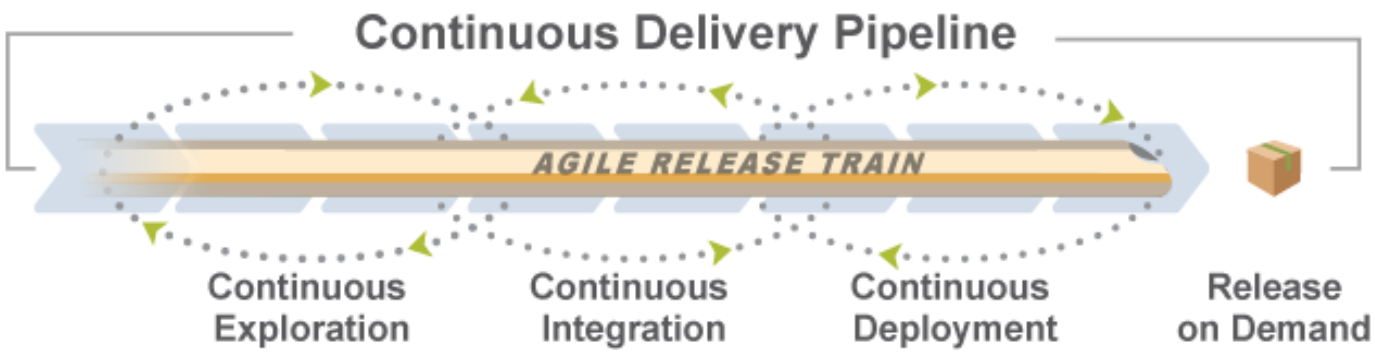
Large systems continuously evolve

- Innovations follow a known, ‘S-shaped’ adoption of growth
- Not one-and-done; purpose and mission change over lifetime
- Strive to release earlier and evolve



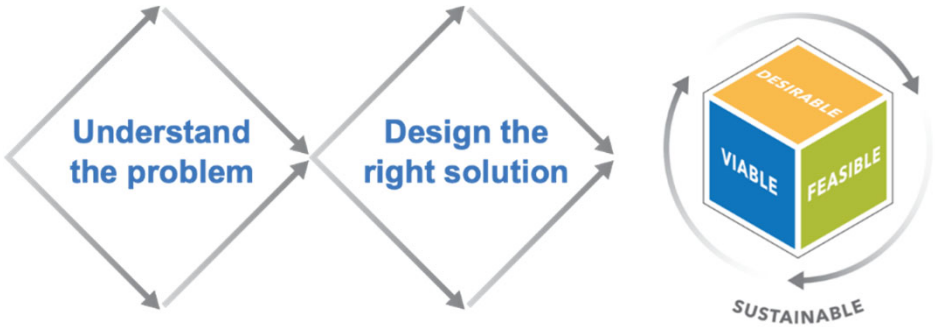
What is a Continuous Delivery Pipeline (CDP)?

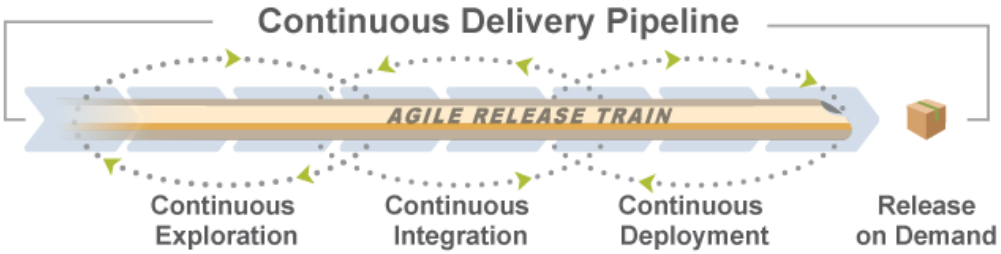
- Workflows, activities, and automation to release new functionality
- Enables business agility by optimizing flow of value delivery



Create the Solution *and* the CDP together

- Both critical to system's long-term success
- Reliable CDP enable trust to deploy and operate system sooner





Discussion: Is the CDP important in your context?

- How much importance is placed on building the CDP in your system development?
- Given a more predictable and reliable delivery system:
  - What value would it add?
  - What attitudes from behaviors from business and customers might change?
  - How might it affect scope, cost, schedule, and quality?

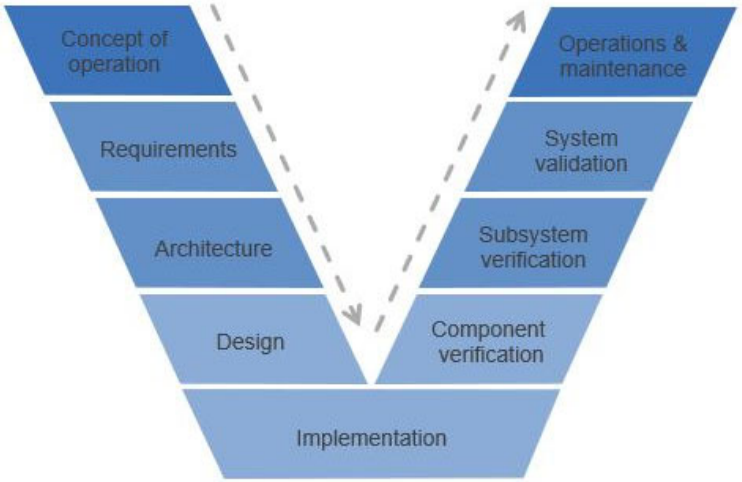
DISCUSS 5 min

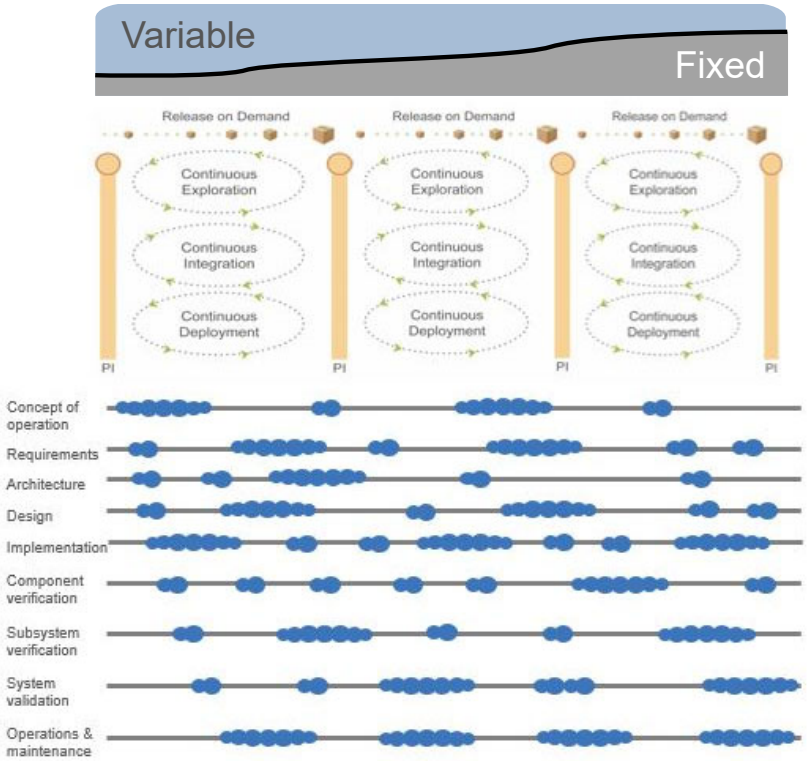
SHARE 3 min

Know 9 practices to building really big systems

1) Capture and refine systems specifications in a fixed/variable Solution Intent

Large system practices don’t go away; they are continuous



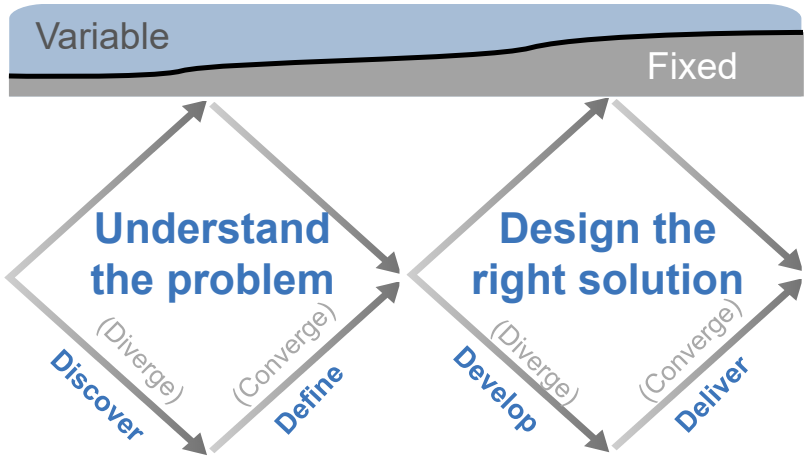


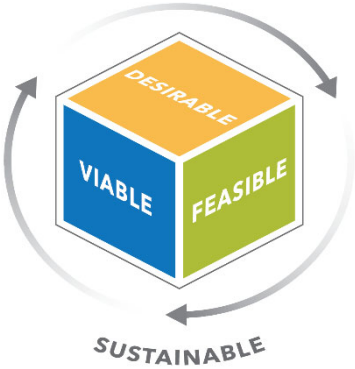
Start by understanding customer needs

- Continuous Exploration uses Design Thinking to understand innovation and builds alignment on what should be built

Customer Centricity

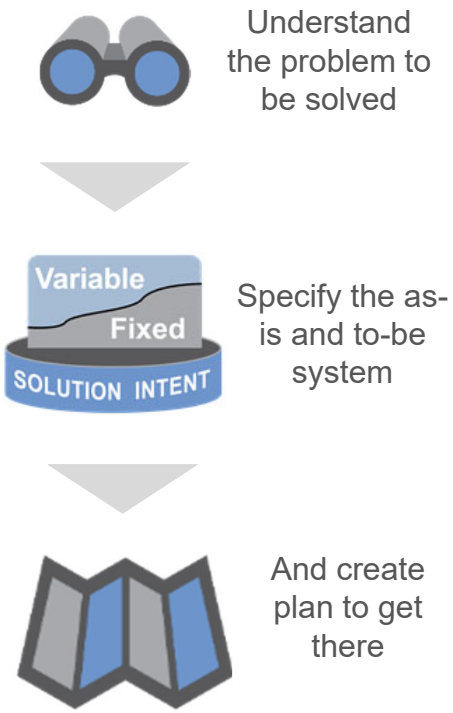
Design Thinking





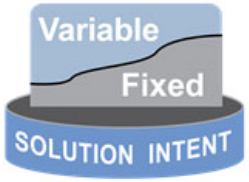
Define Vision, Intent, and Roadmap for large solution

- Vision defines the solution's goals and benefits; the guiding 'north star' for all solution builders
- Solution Intent manages the requirements, designs, constraints (NFRs), tests, etc. for the current and intended solution
- Roadmap forecasts deliverables over a planning horizon



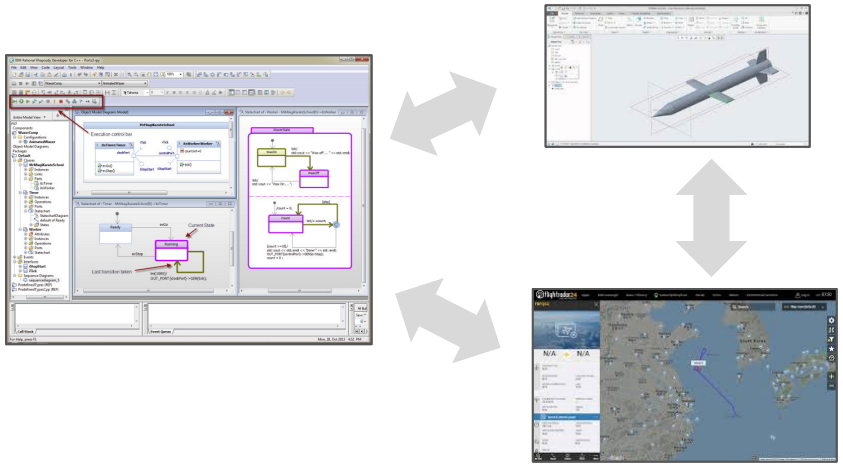
Create and evolve with face-to-face *specification workshops*

- Performed continuously throughout solution development
- Provides real-time decisions and review across multiple disciplines



Use MBSE/Digital Engineering to evolve specifications

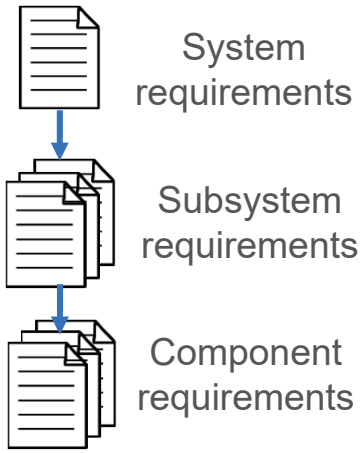
- Drive engineering and development with integrated models
- Keep models synced help downstream when products don't work.
- Integrated models provides quick 'why did we get this problem' resolution.
- Towards digital twins

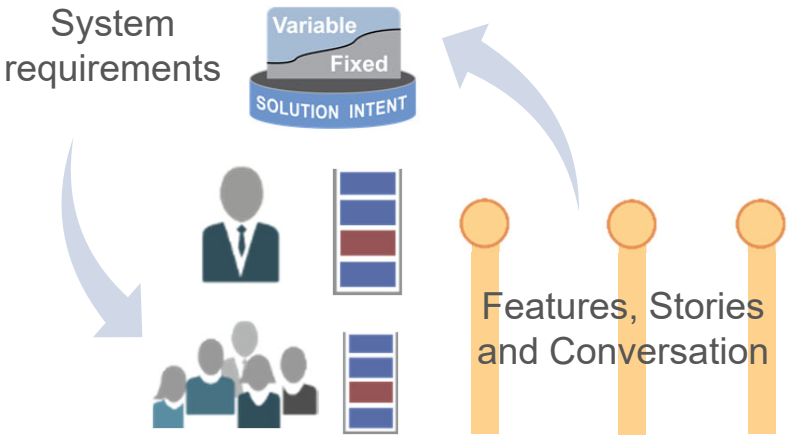


Backlogs replace detailed requirements specifications

Traditional requirements
- Detailed early be senior engineers and handed-off to development
- Little opportunity for feedback or learning
- Slow to adapt

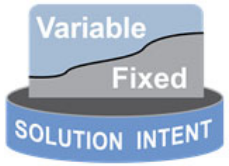
Lean-agile requirements
- Detailed at the appropriate time by the people doing the work
- Short learning cycles provide fast feedback
- Quickly adapt to new knowledge

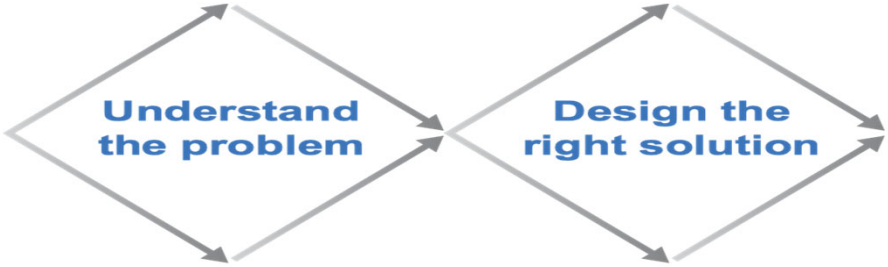




Discussion: Communicating specifications

- How does your organization manage and communicate specifications?
  - How do you *understand the problem* (user-centered?) and *define the solution* (mange and evolve specifications)?
  - What are opportunities to improve either?



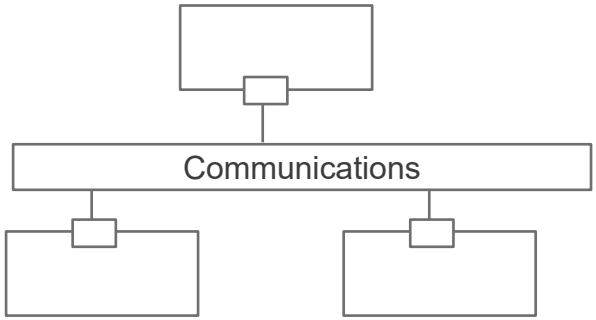


DISCUSS SHARE

2) Architect for scale, modularity, release-ability, and serviceability

Architect for modularity

- Define independent modules that communicate through managed interfaces
- Enable teams and ARTs to independently build, test, deploy, and even release

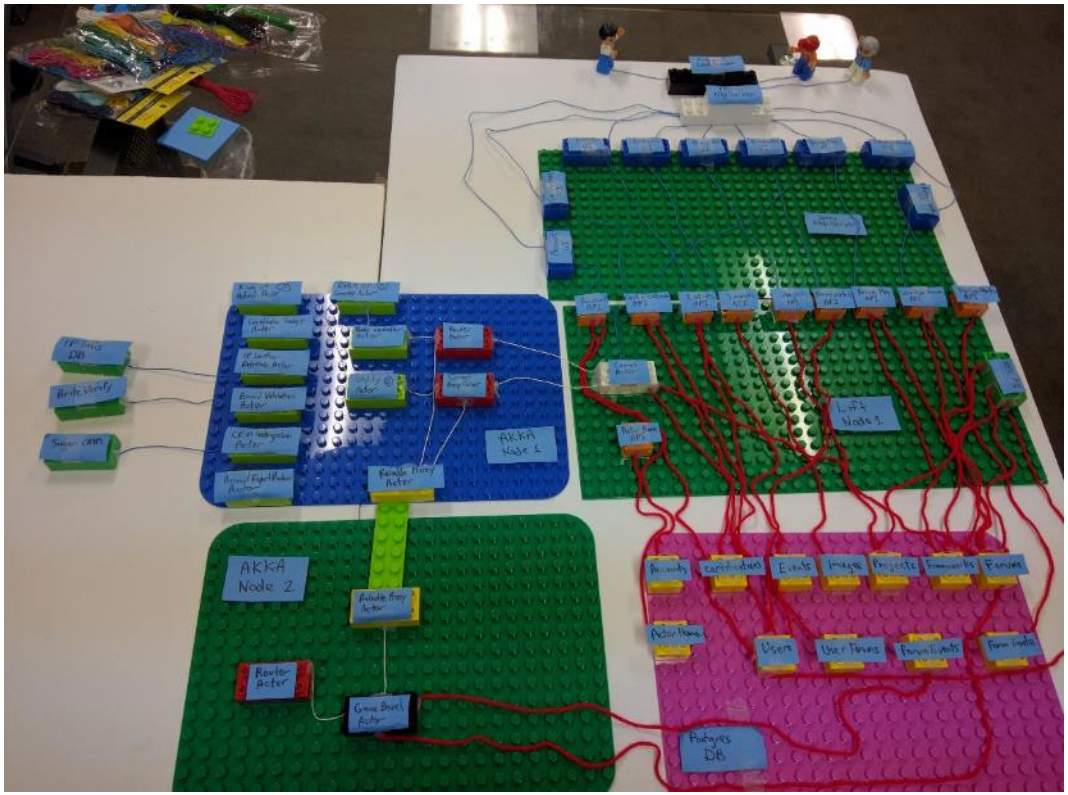


Decoupling considerations

- What is the communication strategy?
- What components can be released or serviced separately?
- How are manufacturing costs impacted?

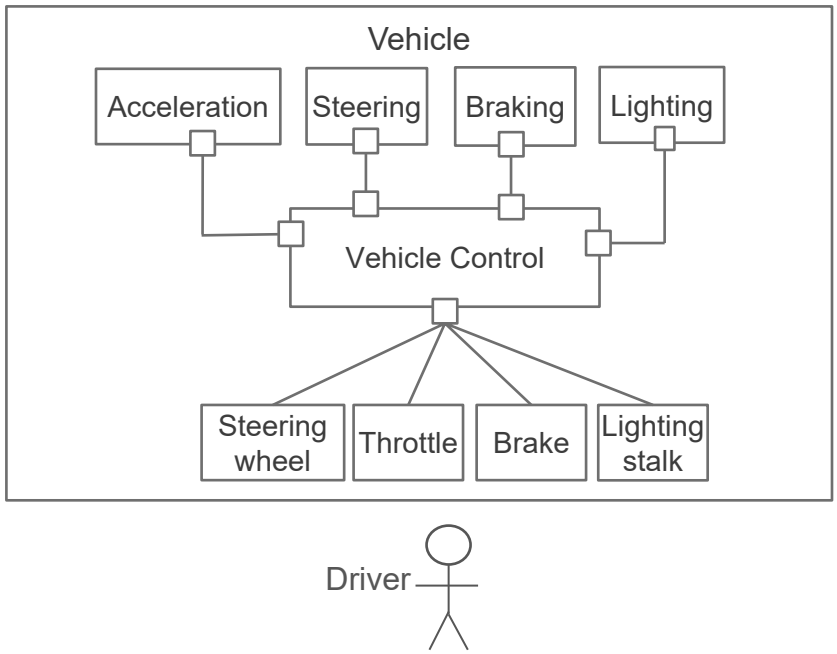
Use Visible Architectures to create shared understanding

- Visible architectures communicate system's structure and communication paths
- Creates alignment, shared understanding

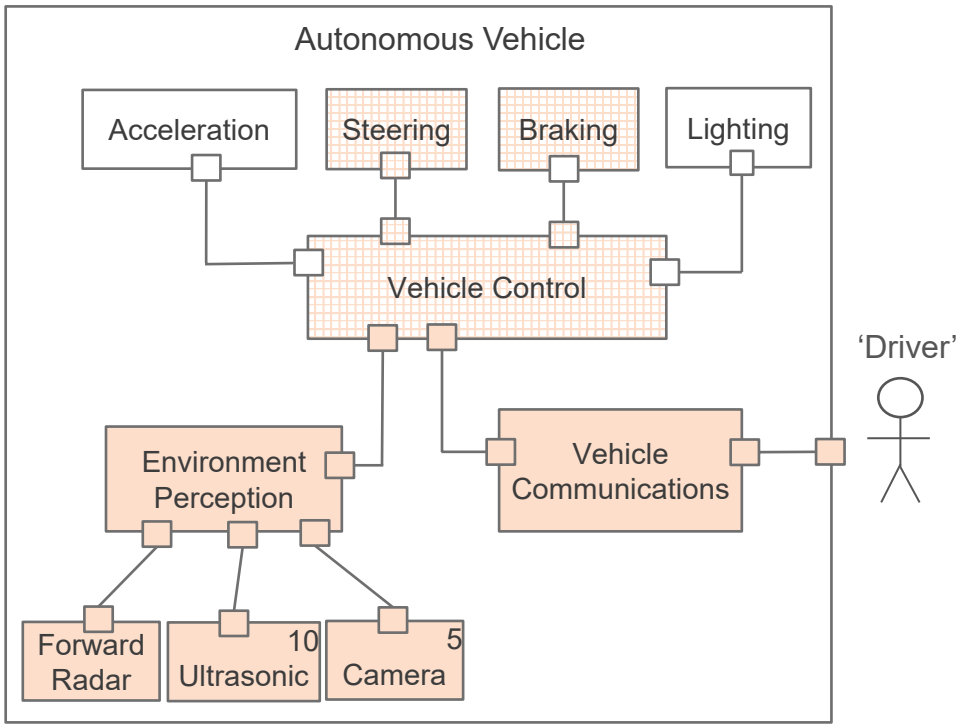


Create two visible architectures – as-is and to-be

As-Is

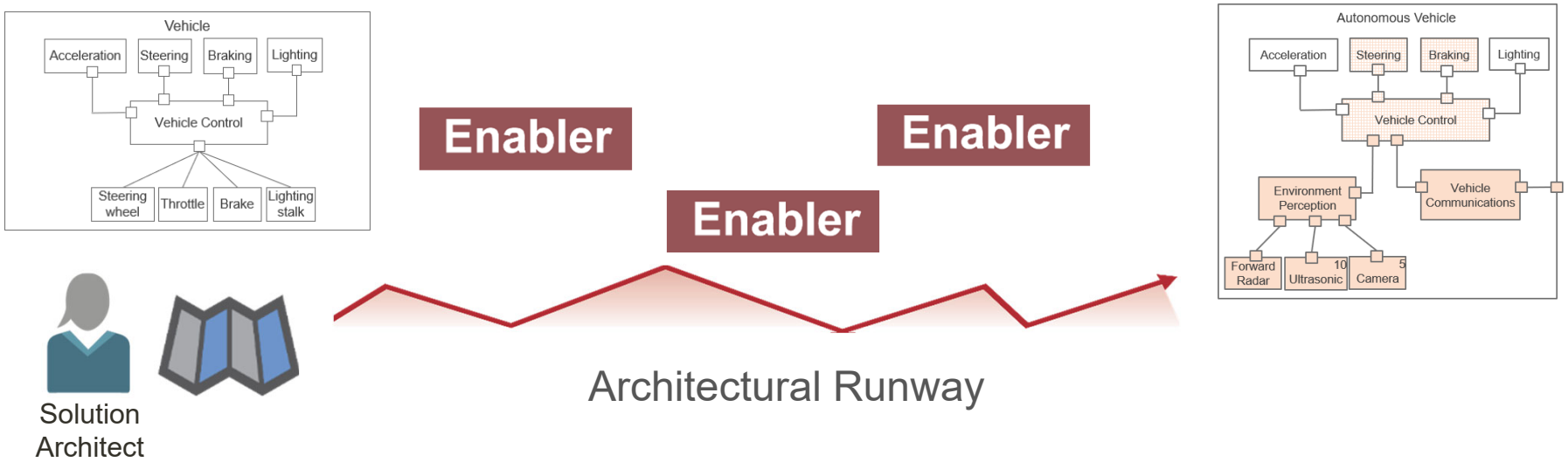


To-Be



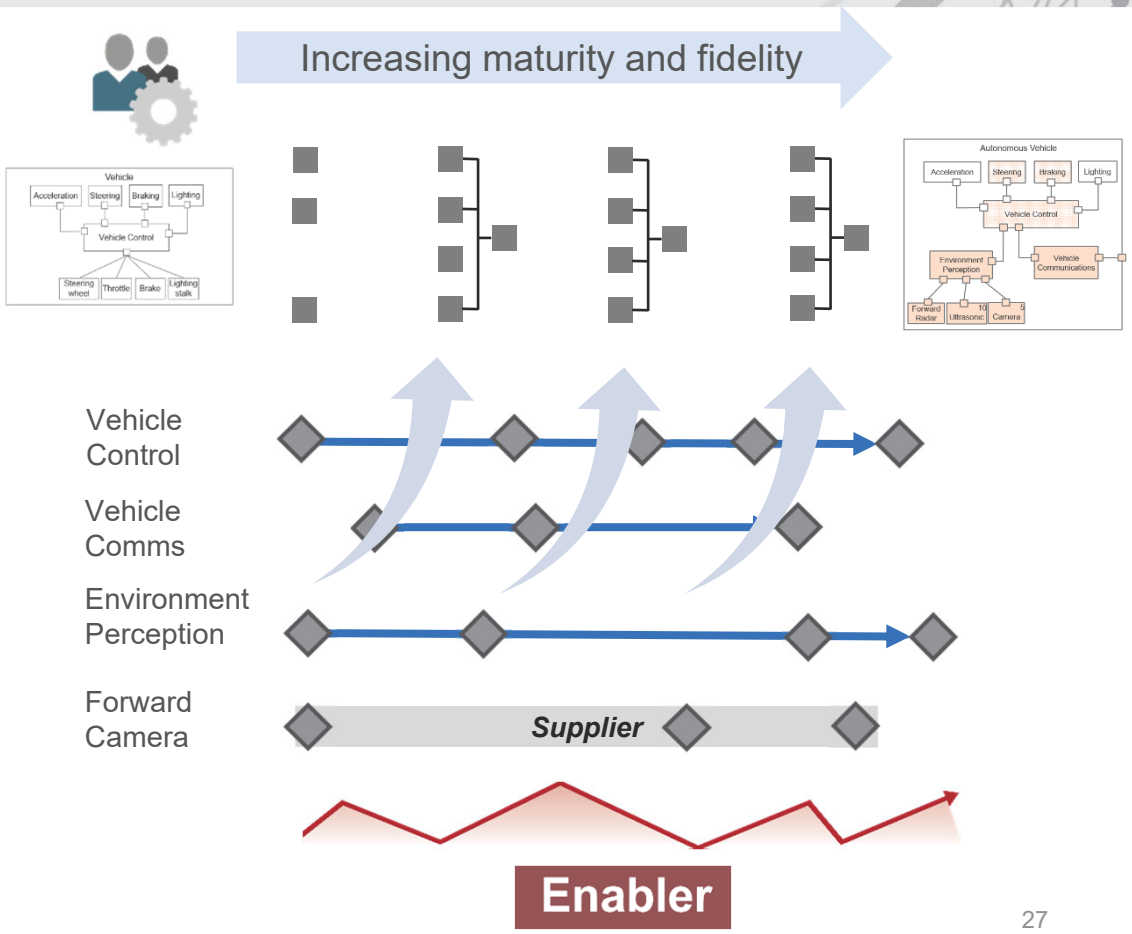
Architecture Runway evolves the systems from as-is to to-be

- *Enablers* build the components and infrastructure for new functionality



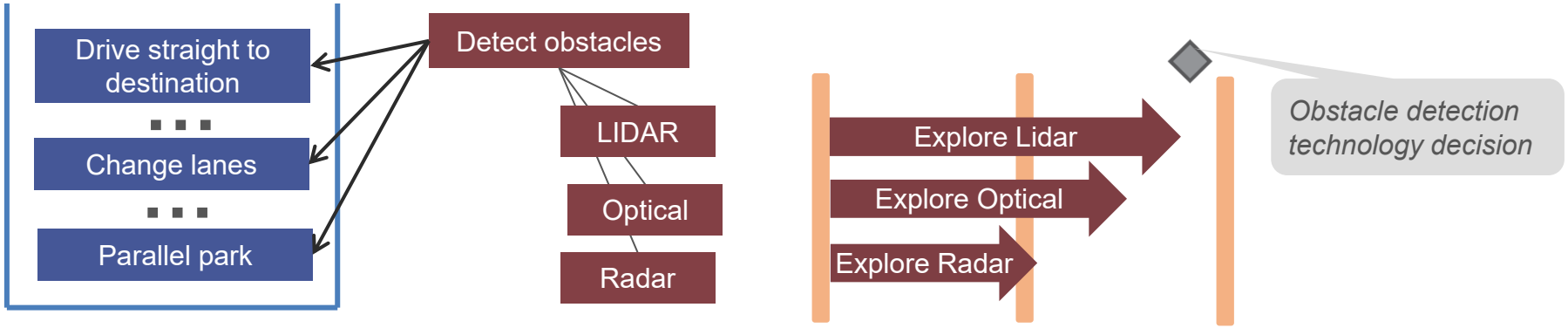
Architecture runway evolves platform from as-is to to-be

- Milestones represent new prototype revisions
- Prototype versions include simulations, models, new SW or HW revisions



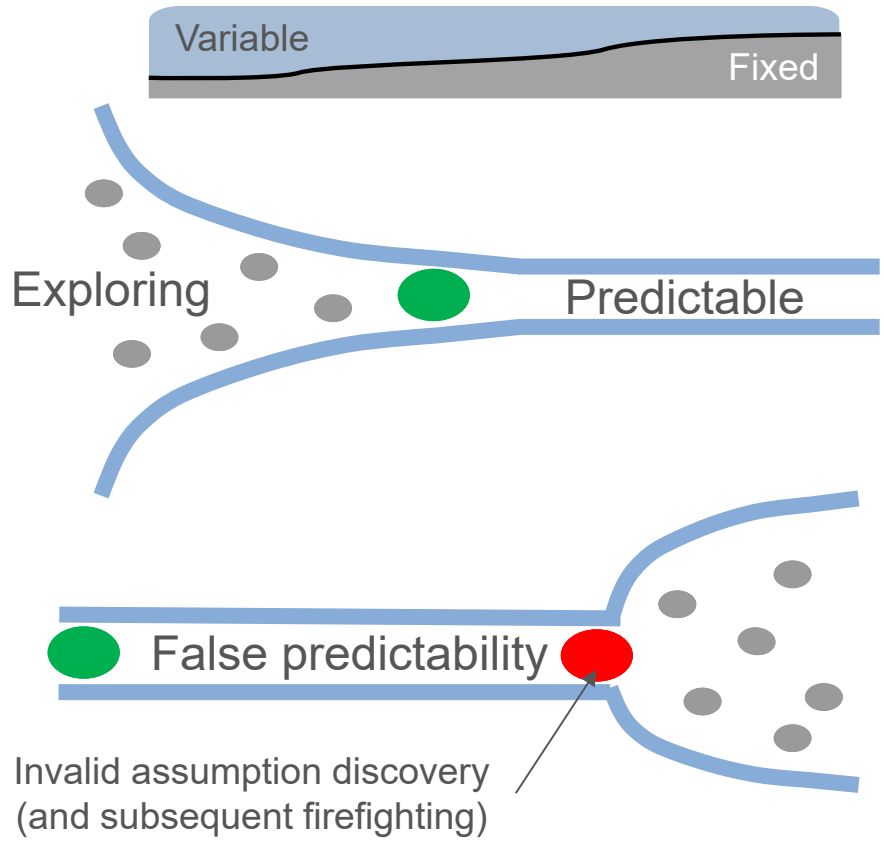
Enablers may explore alternatives

- Risk mitigation strategies define backlog items necessary to reduce risks
- Decision may need to address broader concerns



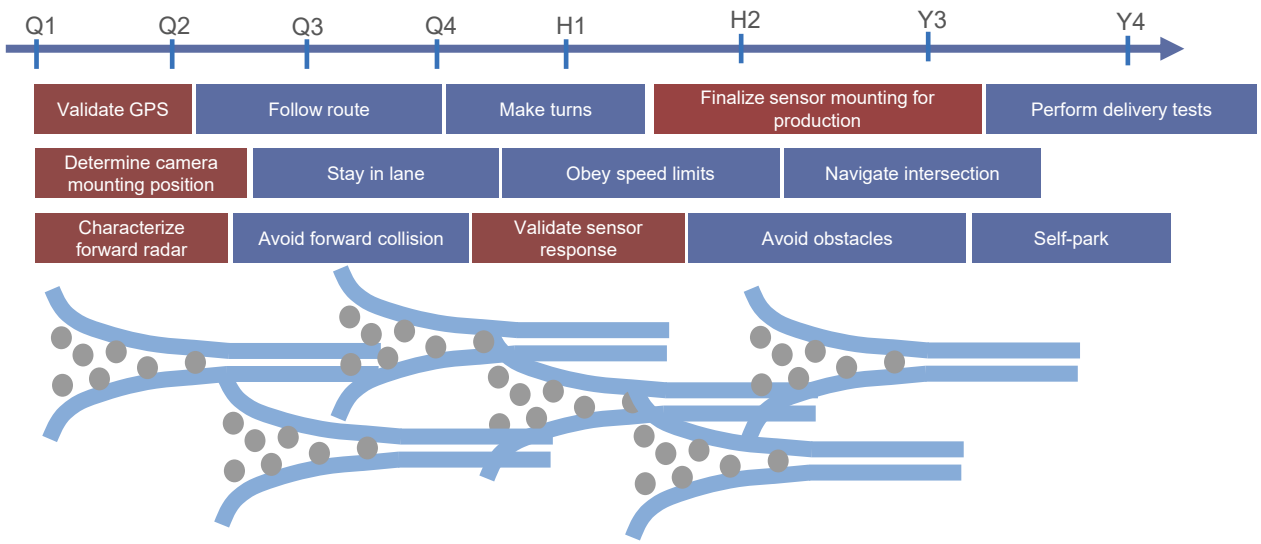
Arrive at the best decision with Set-Based Design (SBD)

- Explore alternatives to arrive at the *optimal* decision, not the *first* decision
- Keep requirements and design options open as long as possible



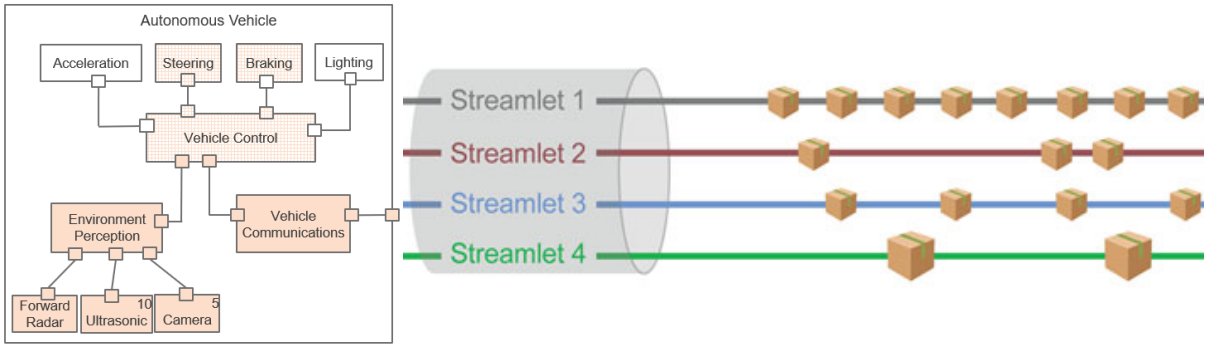
Exploration is continuous

- Learning performed in small batches
- Managed and coordinated by the Architect/Engineer roles in SAFe



3) Architect for release-ability

- Different parts of the solution require different release strategies
- Architect the solution to enable the various strategies and to shift them over time based on business demand
- Isolate structural complexity



How do your architectures support continuous delivery

- Individually, consider a system architecture you have worked with.
  - How well did that architecture enable changes to flow from developers through test, validation, and into operation?
  - Where were the primary bottlenecks?
- Discuss at your table

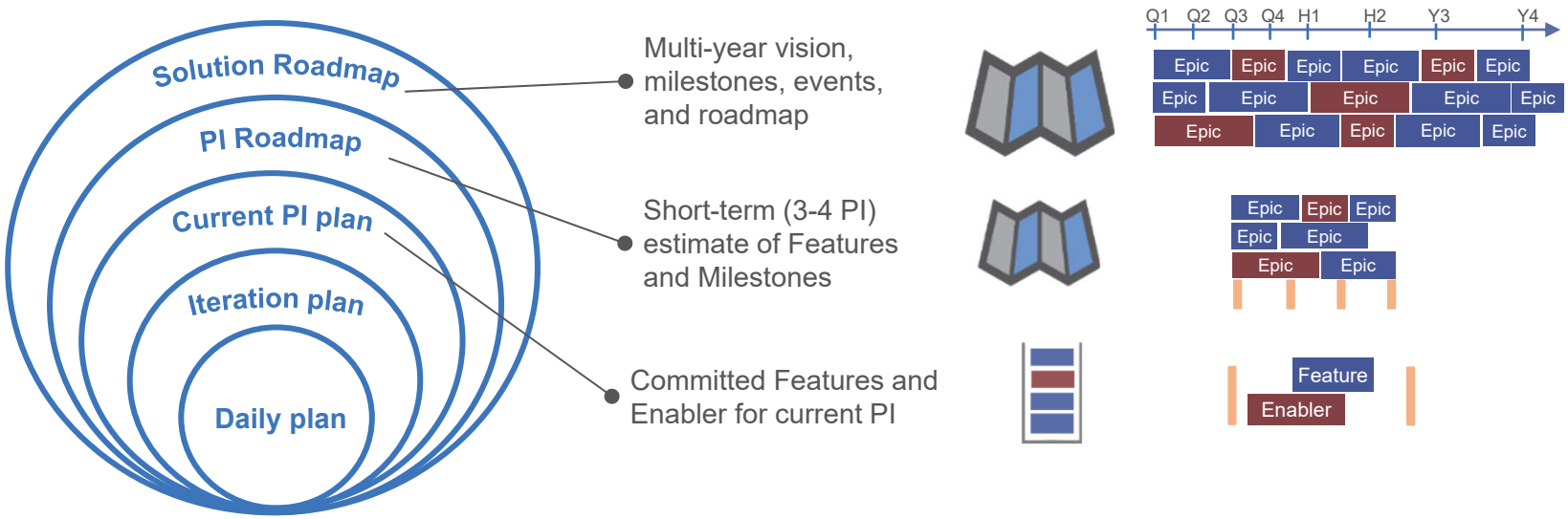
CONSIDER 3 min

DISCUSS 2 min

3) Apply multiple planning horizons

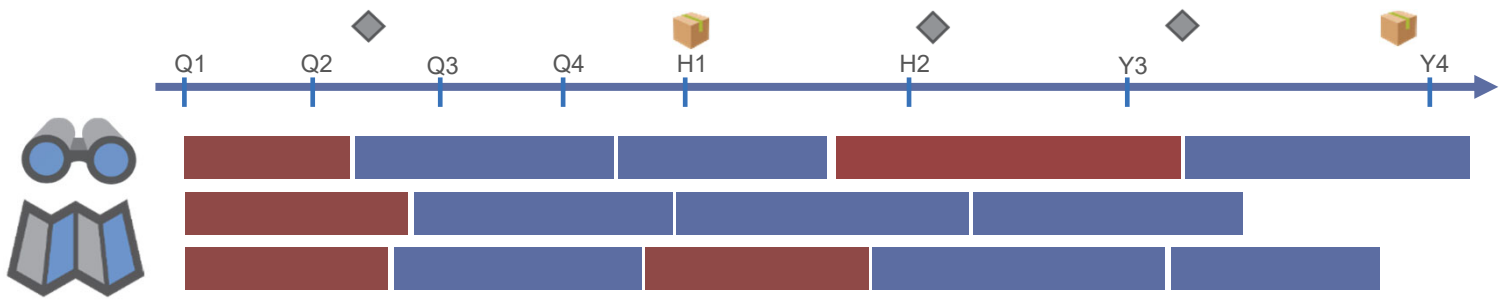
Planning occurs at multiple levels

- Outer levels less defined, committed
- Inner levels more understood, detailed



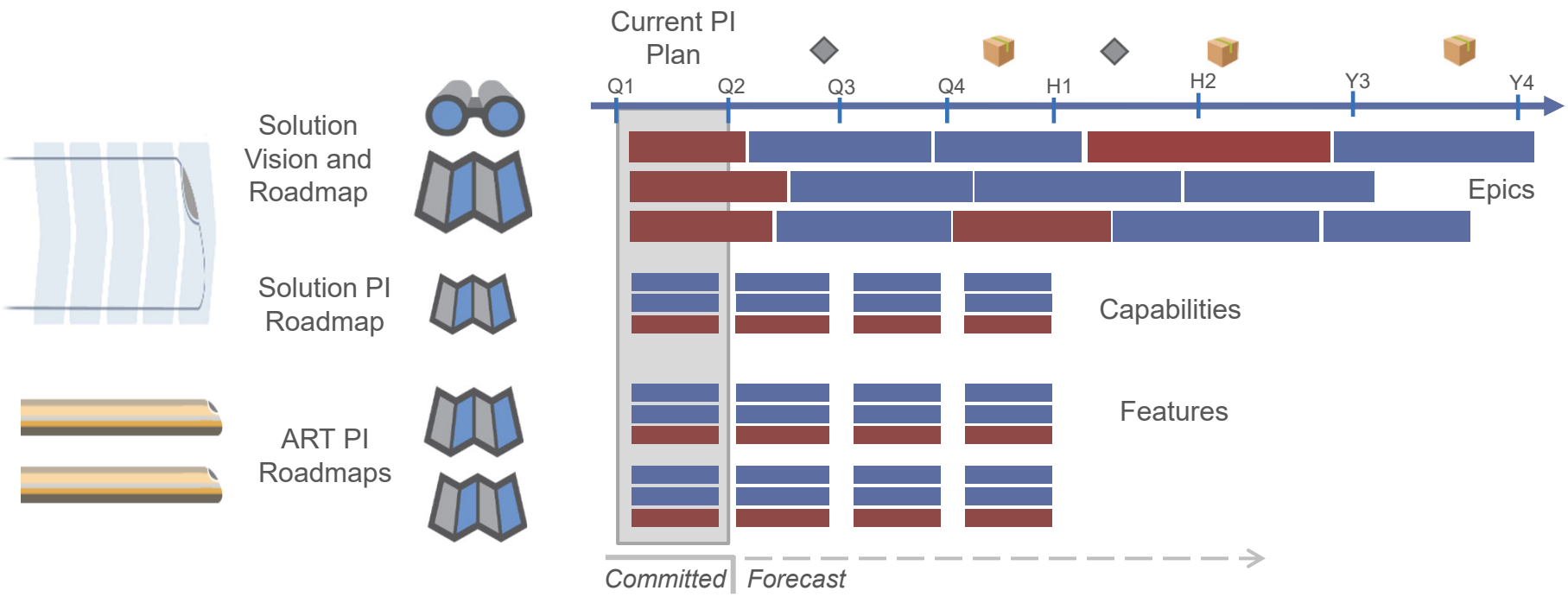
Use Solution Roadmap to forecast scheduled activities

- Shows Epics sequenced over time
- Depicts highly-visible milestones and releases
- Describes a forecast, not a commitment



PI Roadmaps detail next 3-4 PIs

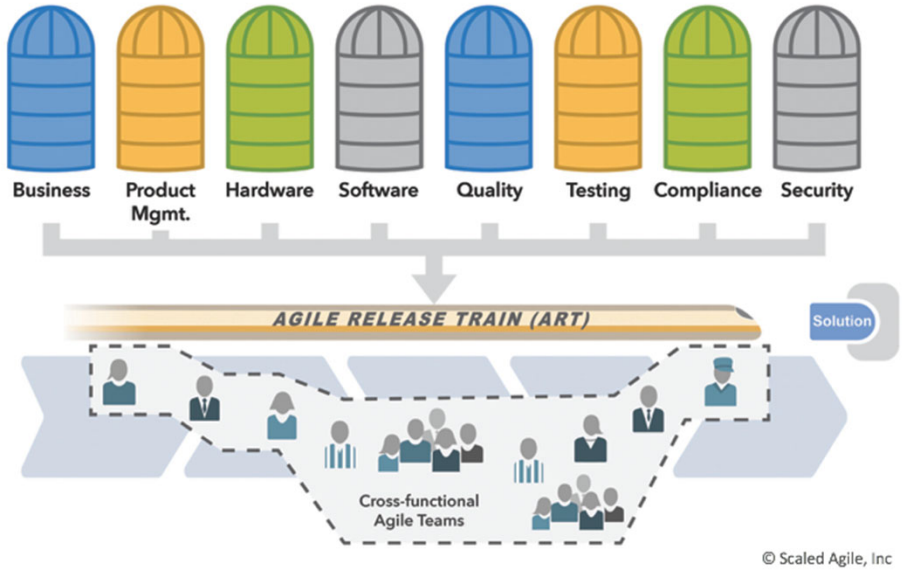
- At scale, Solution Train and ARTs synchronize on PI Roadmaps



5) Build and integrate solution components and capabilities with ARTs and Solution Trains

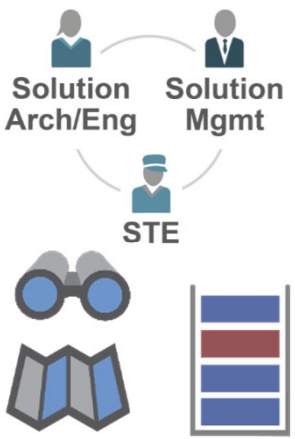
Use ARTs to build large features or components

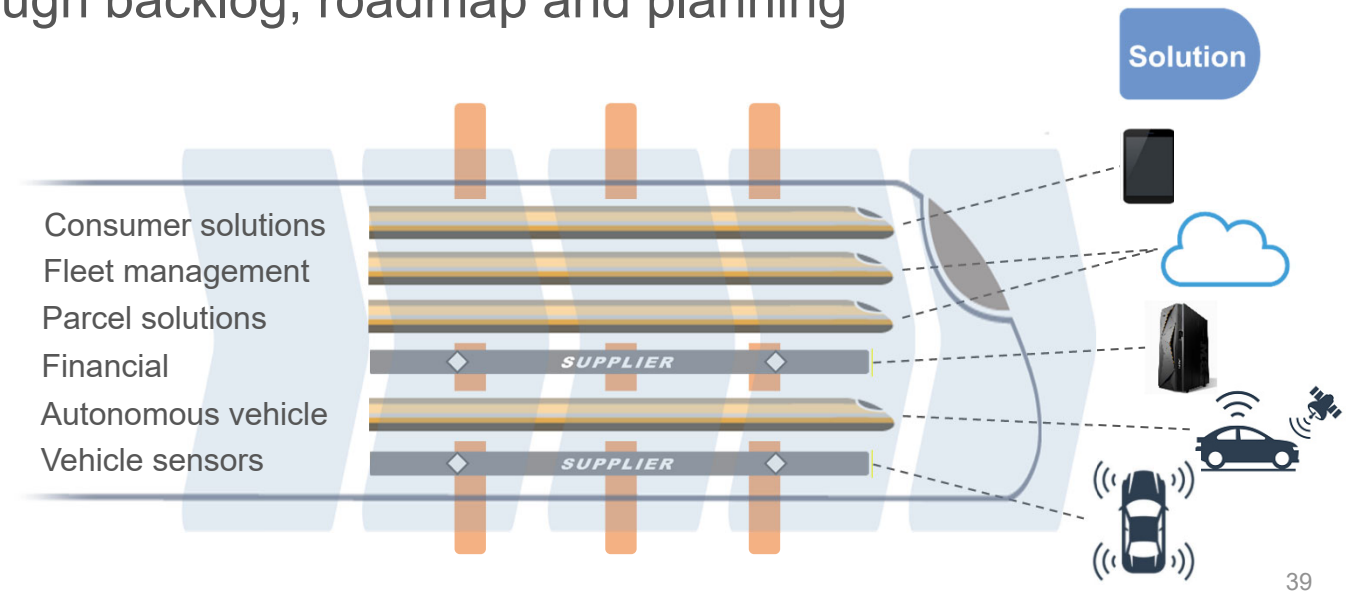
- Virtual organization (50-125 people) that create a solution or capability
- Can independently explore, specify, build, integrate, test, deploy, and (possibly) release



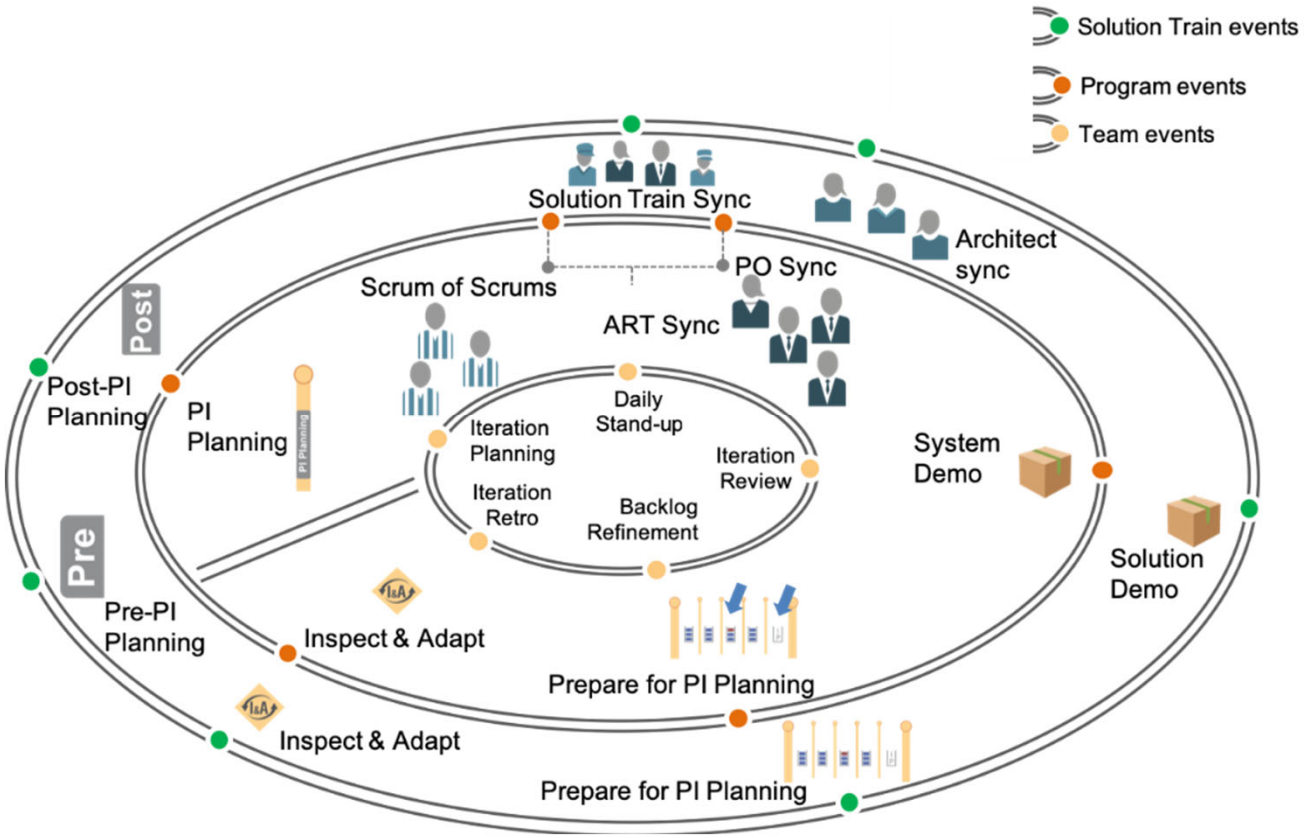
Coordinate ARTs with a Solution Train

- Aligned with common vision and shared business and technical mission
- Provide content, architecture, and execution leadership (triad)
- Coordinate through backlog, roadmap and planning





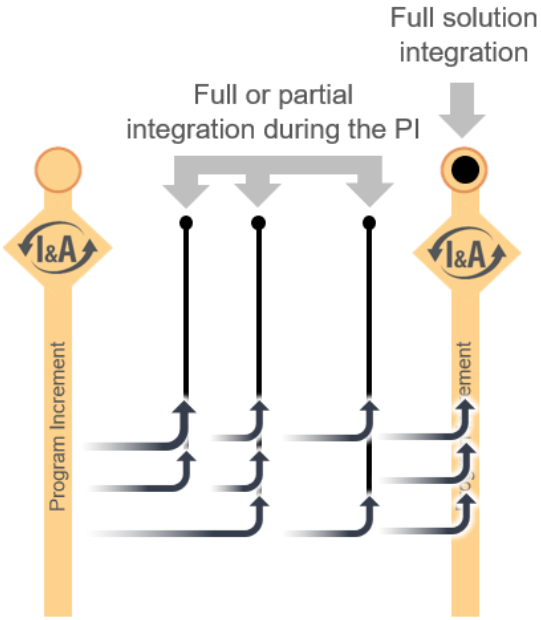
Solution Trains and ART coordination events



6) Apply ‘continuish integration’

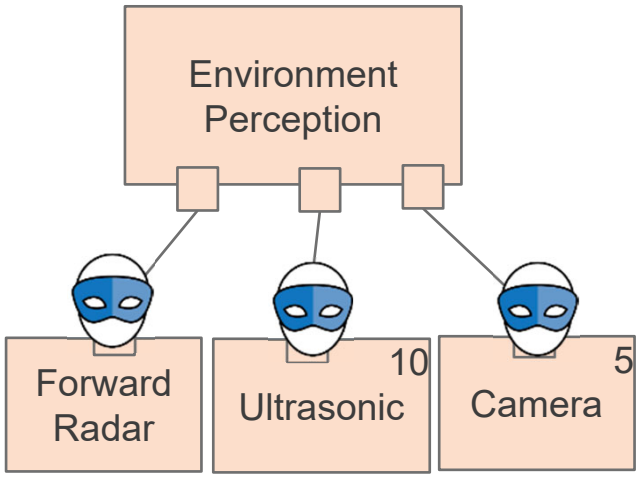
Support frequent integration and testing in large systems

- Frequent integration and testing provides fast feedback
- Do not let small changes sit idle; find a way to integrate with other changes
- Economic trade-offs are inevitable in terms of:
  - Frequency of integration
  - Depth of integration
  - Fidelity of feedback



Test Double provide early, end-to-end system integration

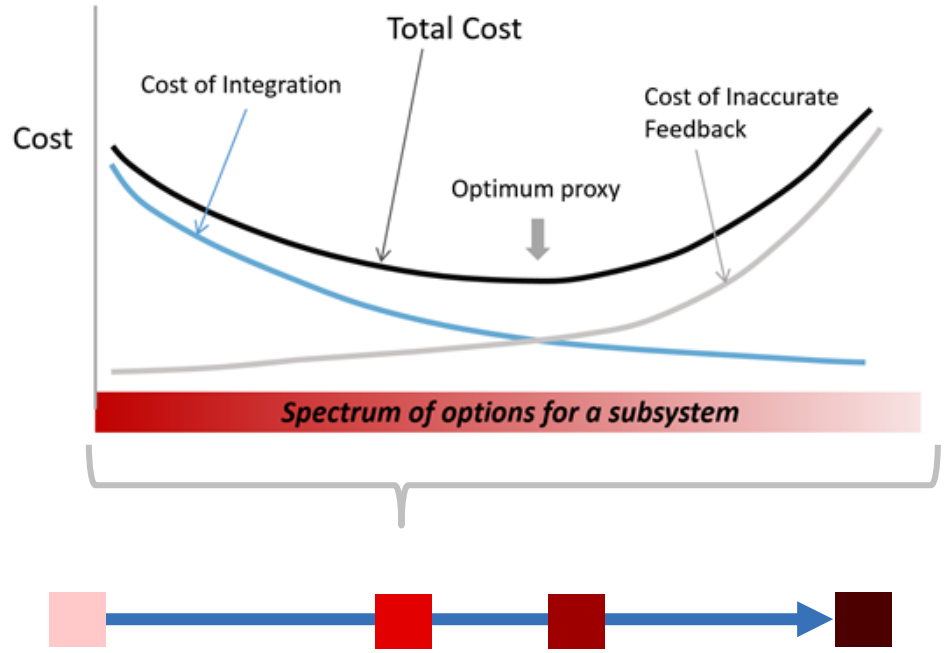
- During construction, solution integration is realized as a combination of real subsystems and Test Doubles of varying maturity
- Component teams and ARTs create Test Doubles as they progress towards their final solutions
- Interfaces allows components to mature independently



Test Doubles at evolving levels of maturity (software stub, simulation, previous system, development kit, wood mockup, etc.)

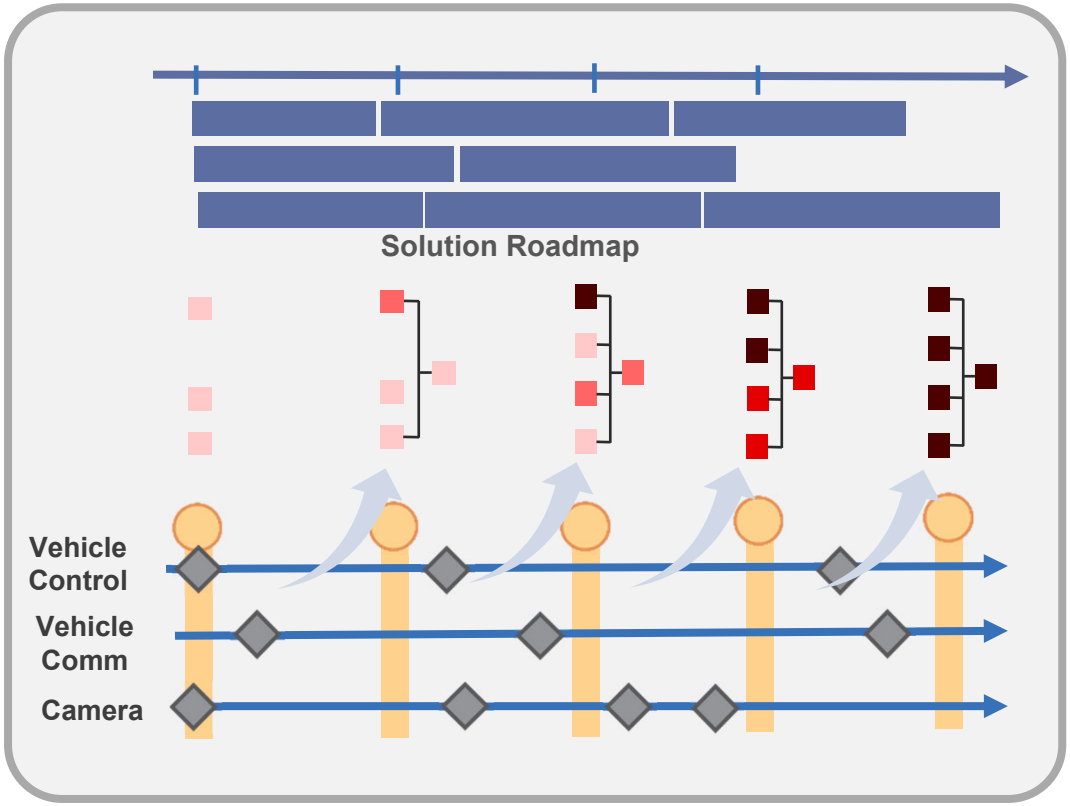
Evolve the system by evolving the Test Doubles

- Mature Test Doubles over time to increase the fidelity of feedback
- Economics determine how quickly we create new revisions
- At any point in time, the tradeoff is a function of less accurate feedback vs. the cost of creating and integrating the next revision



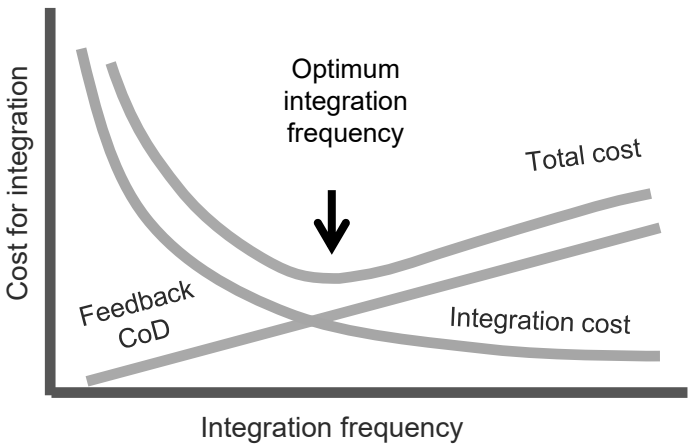
Integrate the full system to support the product roadmap

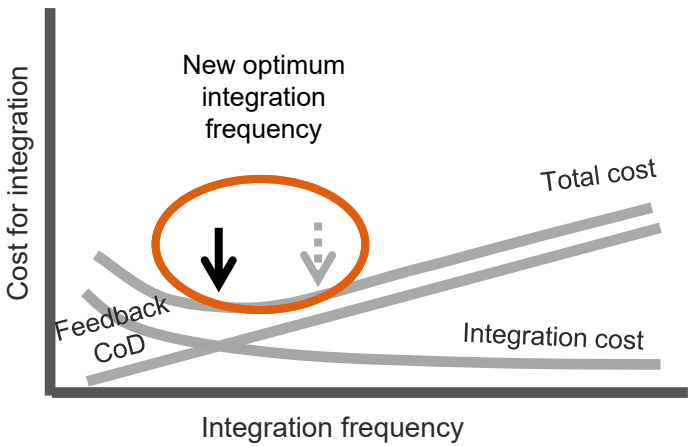
- Provide the technical environment incrementally to demonstrate the system
- Strive for early, end-to-end solution integration that matures in fidelity over time



Invest in automation and infrastructure to lower integration cost

- Reducing integration cost allows more frequent integration





Principles of Product Development Flow, Don Reinertsen

Enabling ‘continuish’ integration

- Discuss your current integration practices
  - How frequent are end-to-end system integrations?
  - What are the challenges to integrating more frequently?
  - What benefits could be achieved by integrating more frequently?

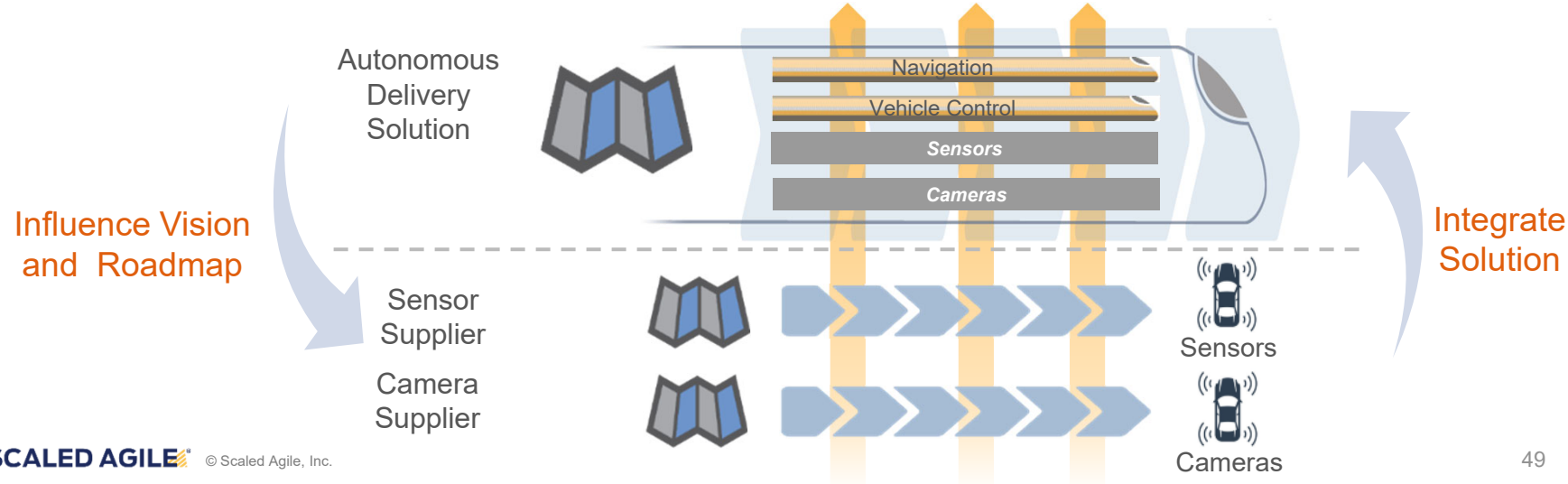
EXPLORE 5 min

SHARE 2 min

7) Manage the supply chain with system-of-systems thinking

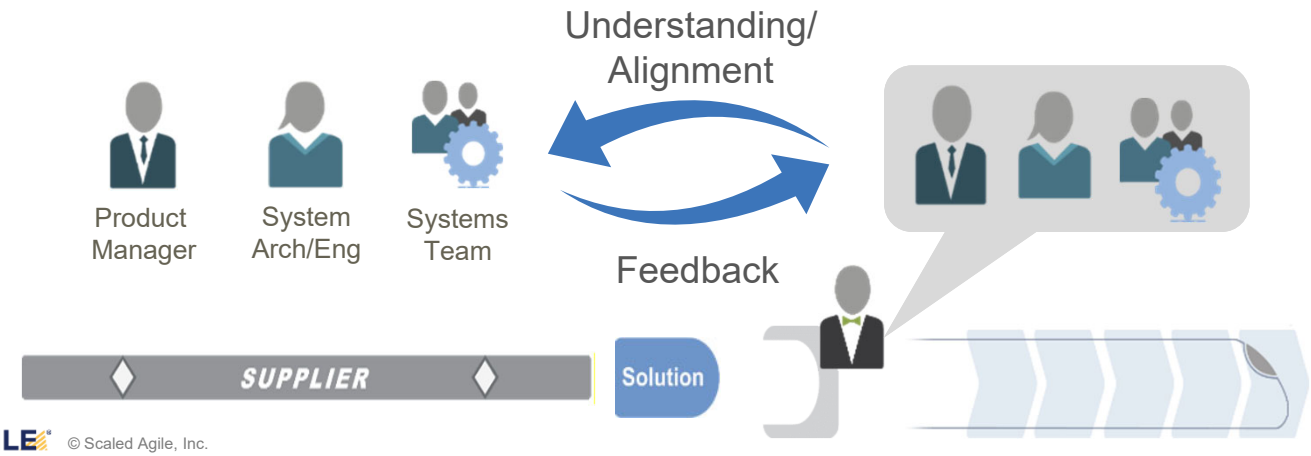
Strive to have Suppliers behave like ARTs

- Ideally aligned on same cadence and driven by common backlog
- Contractor *independence* may vary (are we their only customer?)
- Require more strategic partners to act like ARTs



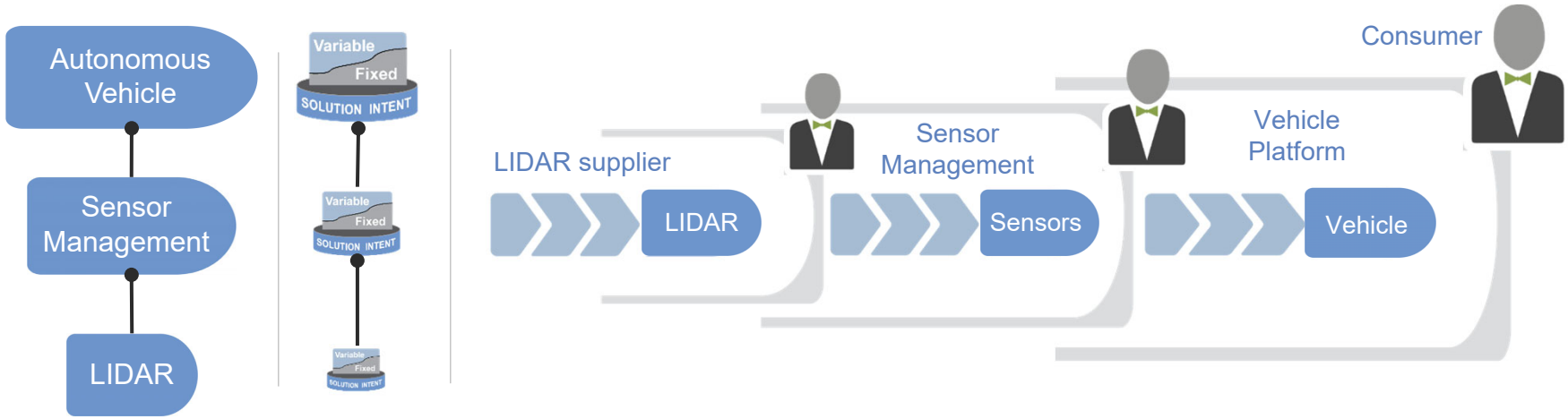
Customer-Supplier is a multi-dimensional relationship

- Continuously collaborate on multiple dimensions:
  - Content – align work through shared roadmaps and backlogs
  - Technical – align technical strategy and share specifications
  - CD Pipeline – share CI/CD assets (scripts, systems, tests) to ensure flow



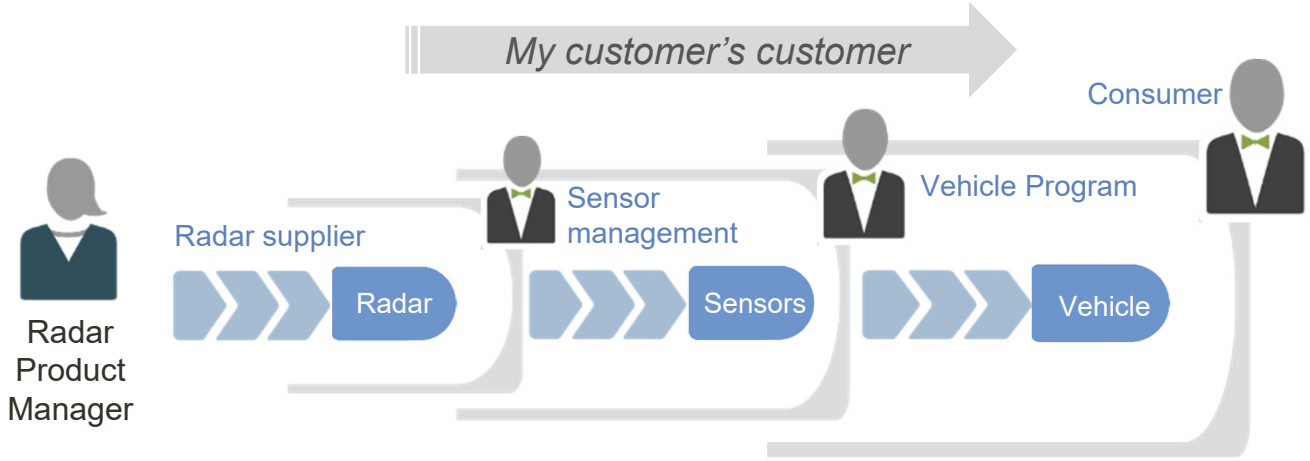
Solution Contexts Wrap for System of System/Supply Chains

- Solution requires continuous delivery across entire supply chain
- Aggregate Solution Intents to support development and compliance
- Align *everyone* on a common cadence

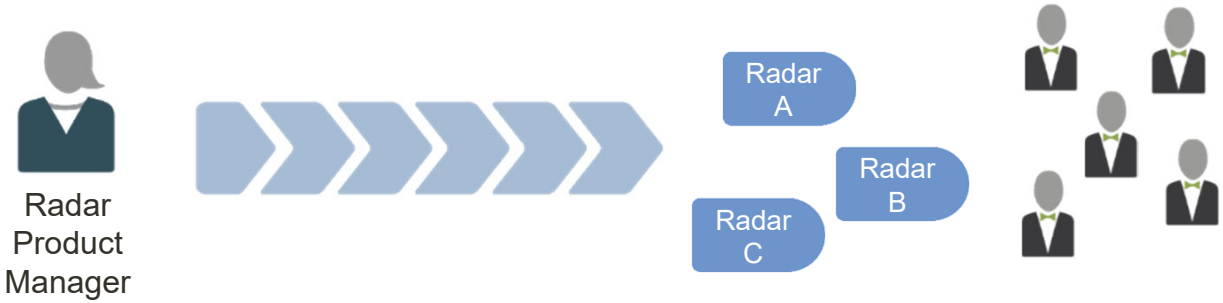


Large supply chains complicate value stream relationships

Solutions are constrained by supply chain contexts

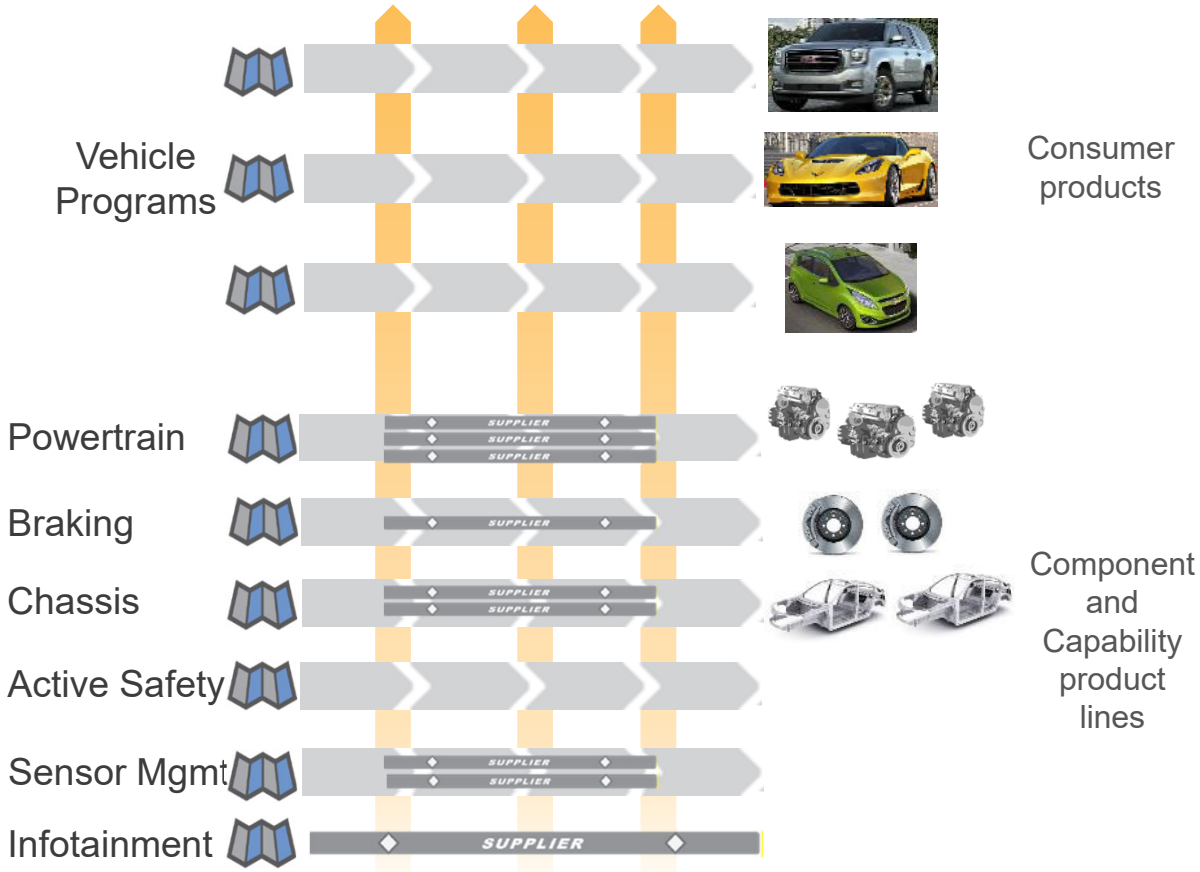


Product lines support multiple customer segments



At scale, Value Stream relationships are complex

- Solutions integrate components from other value streams
- Suppliers may be internal or external

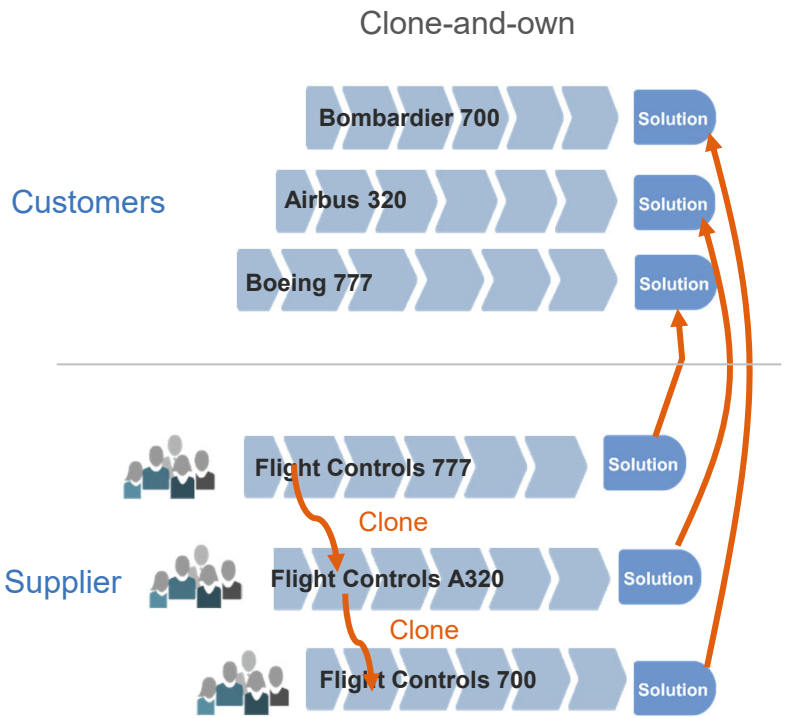


Value stream networks increase the complexity

- Multi-dimensional customer-supplier relationships
- Spans many organizational boundaries

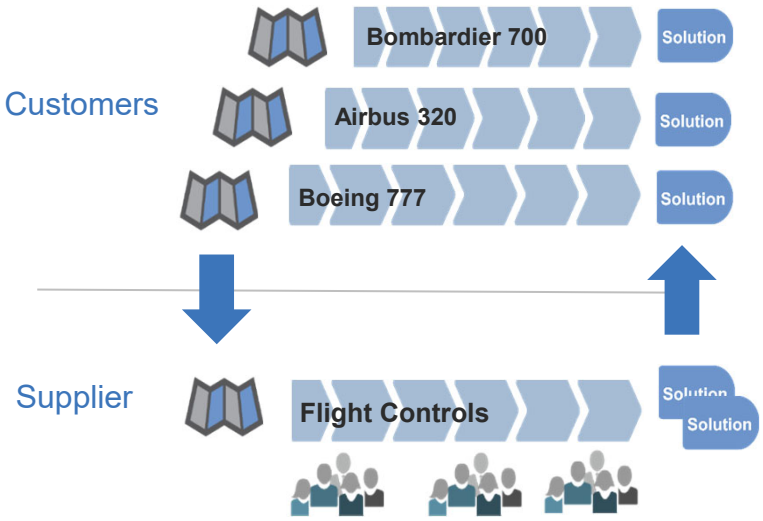
Coordinating value streams – Clone-and-own model

- No platform (yet)
- Versions of the solution a *cloned-and-owned*
- Skills distributed across multiple VSs
- Provides fast execution but no economies of scale



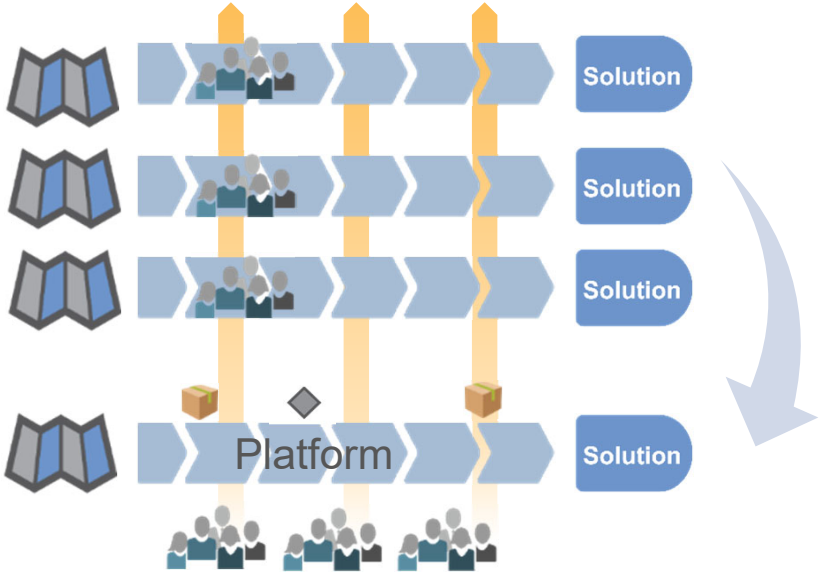
Coordinating value streams – Platform model

- Platform publishes a schedule based on input from dependent VSs
- Dependent VSs plan accordingly
- Good economies of scale, but platform can bottleneck dependent VSs



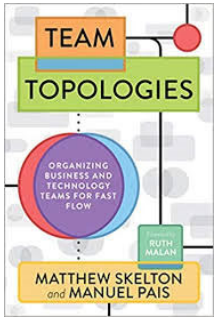
‘Internal open source’ model

- Dependent VSs make changes locally
- Platform integrates all as *pull-requests* to ensure platform integrity
- *If it’s good for anyone its good for everyone*



Align work with ART/Solution Area/Team expertise

- Strive for plans that keep teams and ARTs stable
- Creating ‘T’ and ‘E’ skills simplifies stable team planning



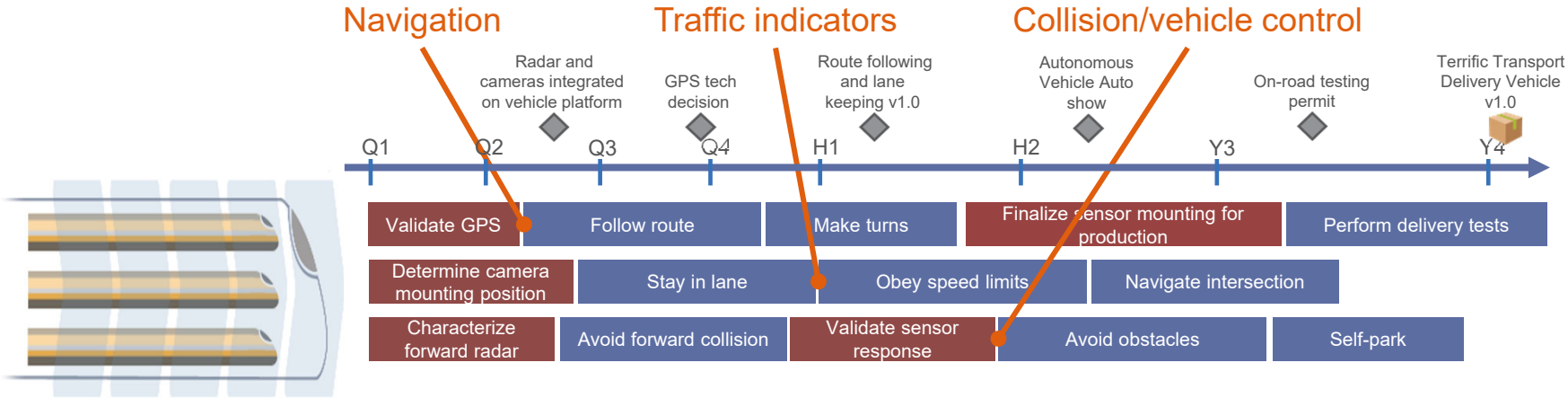
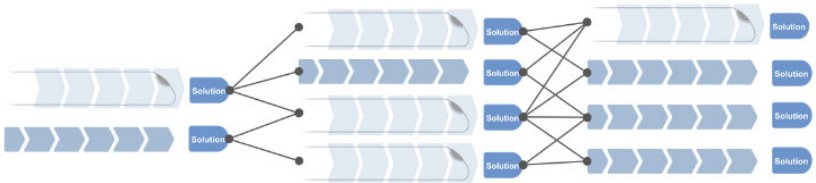
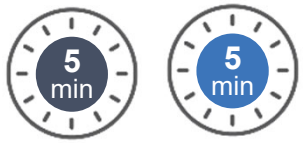


Table Discussion: Coordinating value streams at scale

- Discuss experiences coordination value streams:
  - What is the context? Suppliers, hardware, infrastructure, packaged apps?
  - Is everyone on a common cadence? If not, how do they differ?
  - How are roadmaps aligned
  - How frequently are solutions integrated? Is there shared support/infrastructure?



DISCUSS SHARE



8) Build a Continuous Delivery Pipeline

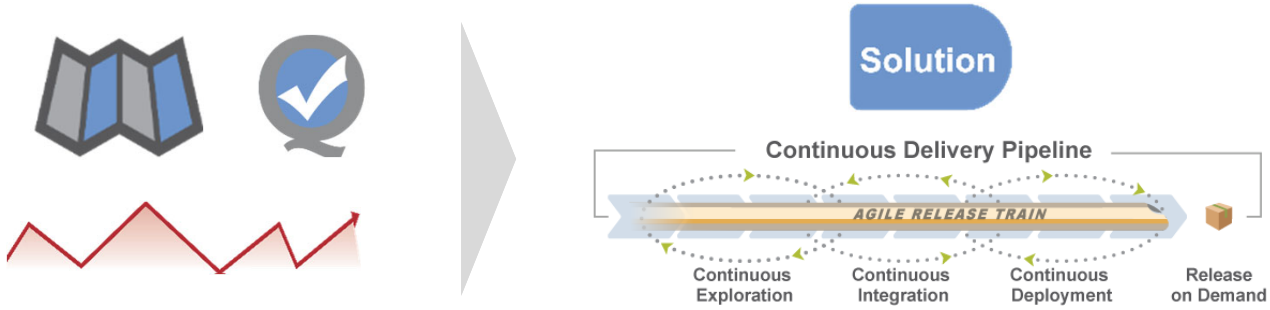
Business focus has traditionally been deploying the system

- Goal is to fund and create the right solution one time
- Any upgrades and modifications will be determined and funded later

What are the problems with this process?

Build the system & and the CDP together

- The pipeline is a product too – requires roadmap, architecture, BiQ, etc.
- New mindset: deploy early and evolve vs. deploy once and support



See 'Inside Elon Musk's plan to build one Starship a week'
– ARS Technica

Modern technologies enable continuous delivery

Service virtualization

Containerization

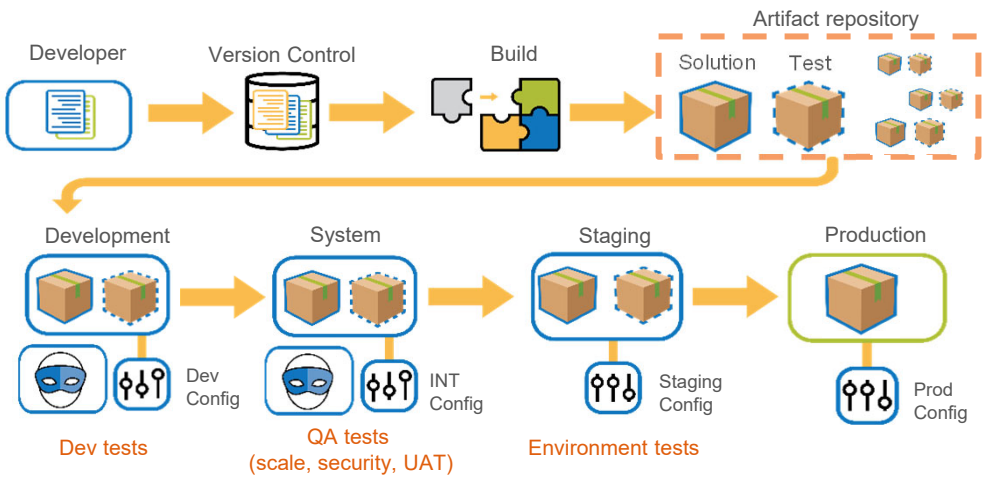
Test and deployment automation

Microservices

Infrastructure -as-code

API management

Monitoring



Virtual models/ Digital twins

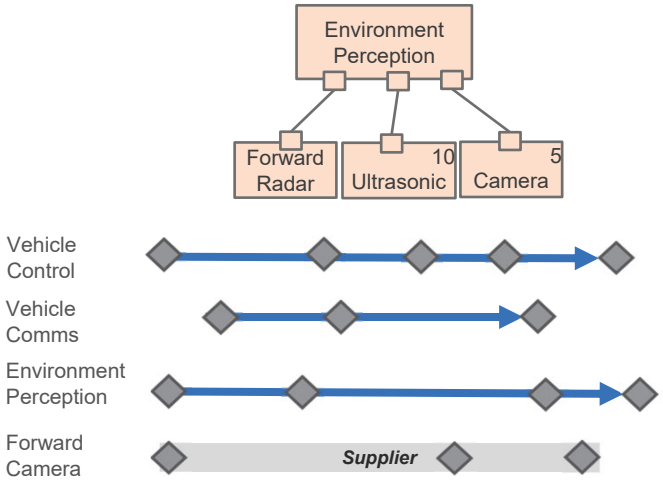
Programmable hardware

Over-the-air updates

Additive manufacturing

Constraints determine how changeable a system is

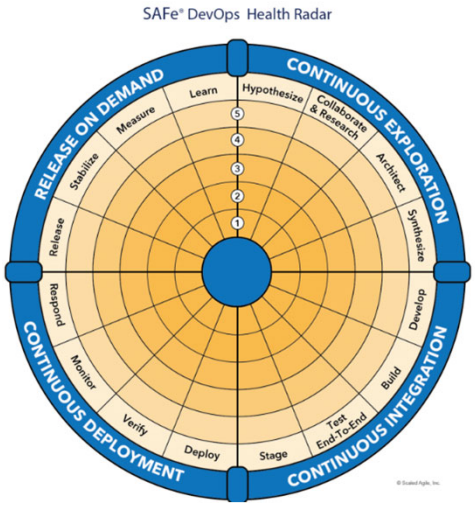
- Determine the constraints and architect them into the system
- Architect the ability to continuously deliver into the system
- Balance economic choices of lower costs (dev, manufacturing, unit costs) with the costs of delayed value and knowledge

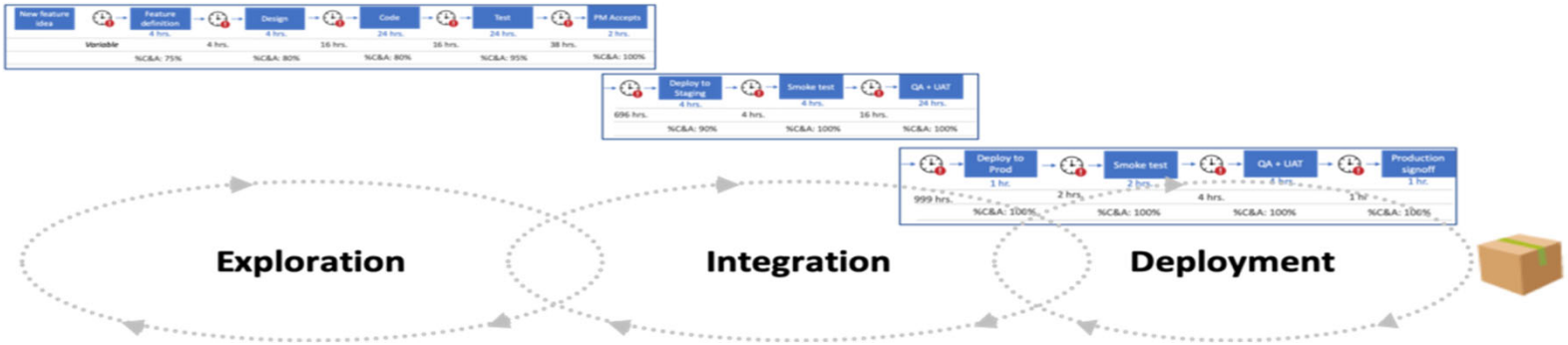


What are some of your system's constraints for continuous delivery?

Continually improve the continuous delivery pipeline

- Measure process time, lead time, and delays
- Create backlog items to build, evolve, and improve the pipeline
- Strive for continuous, but it may not be achievable





How valuable is continuous delivery?

- Discuss at your tables how your organization values continuous delivery. Consider:
  - What value is reflected in backlogs?
  - What metrics are tracked to ensure improvement?
  - Expected competence in continuous delivery best practices
  - Other ways the organization might express value

DISCUSS 5 min

SHARE 2 min

Conclusion

Nine practices for building really big systems

- Lean Solution and Systems Engineering
  1. Continually refine the fixed/variable Solution Intent
  2. Apply multiple planning horizons
  3. Architect for scale, modularity, releasability, and serviceability
  4. Continually address compliance concerns
- Coordinating Trains and Suppliers
  5. Build and integrate solution components and capabilities with ARTs and Solution Trains
  6. Apply ‘continuish’ integration
  7. Manage the supply chain with systems of systems thinking
- Continually Evolve Live Systems
  8. Build a Continuous Delivery Pipeline
  9. Evolve deployed systems

Thank you!