Consolidate.

Simplify.

Leverage.

Pragmatic Methods for Developing Software Product Lines

Charles W. Krueger, PhD
CEO, BigLever Software

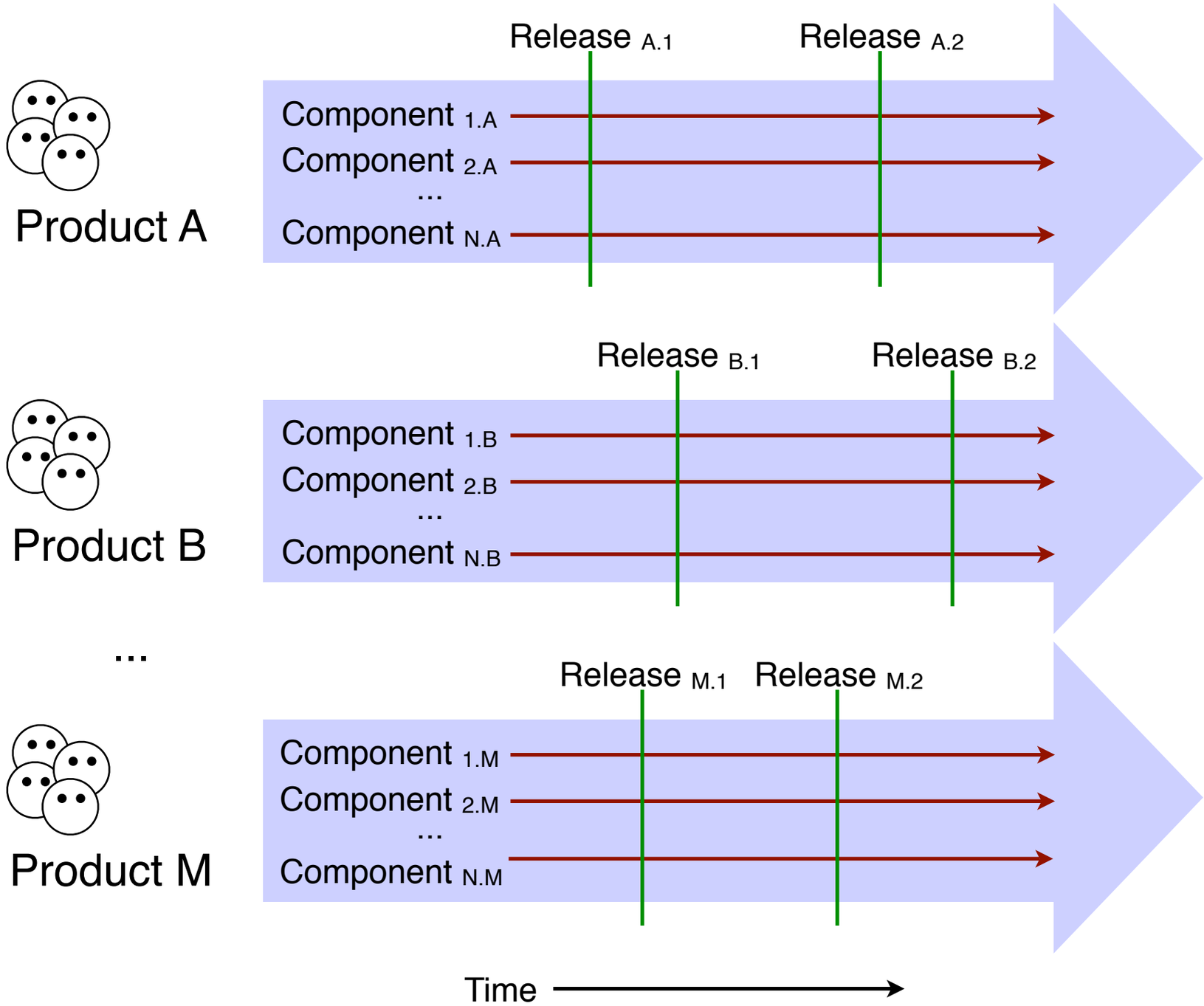
A Mismatch in the Industry

- Customer demand requires most companies to create a product line of differentiated products – with variations in features and functions – rather than a single product
- Most software development tools, methods and techniques focus on individual products

Product versus Product Line Mismatch

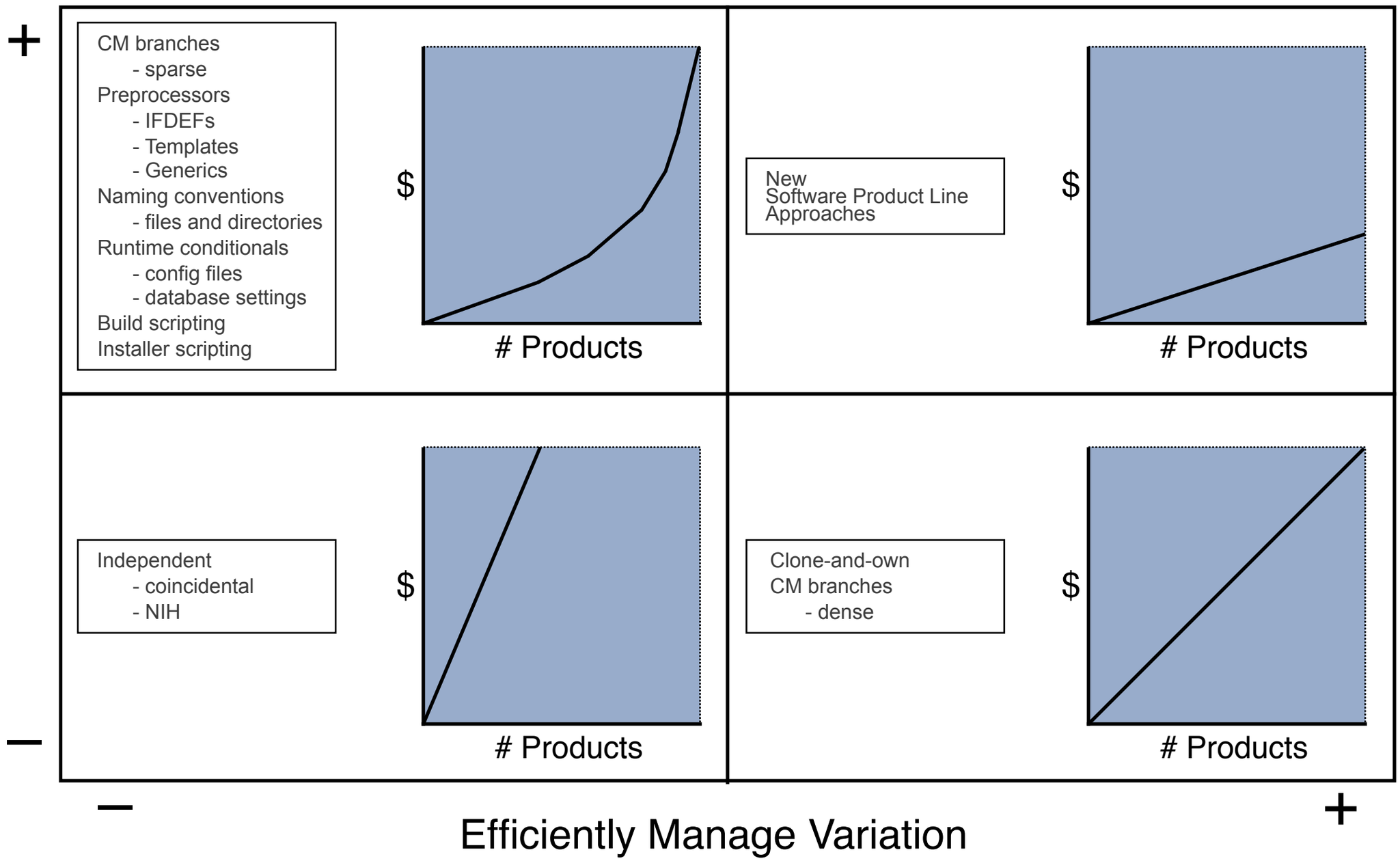
- Product-centric approaches fail to simultaneously...
  - Capitalize on commonality across products
  - Efficiently manage the variations among products
- Which leads to...
  - Duplication of effort
  - Labor intensive complexity
- Case studies show this can account for 50% to 90% of the development effort for a large software product line

Product-centric Commonality and Variation

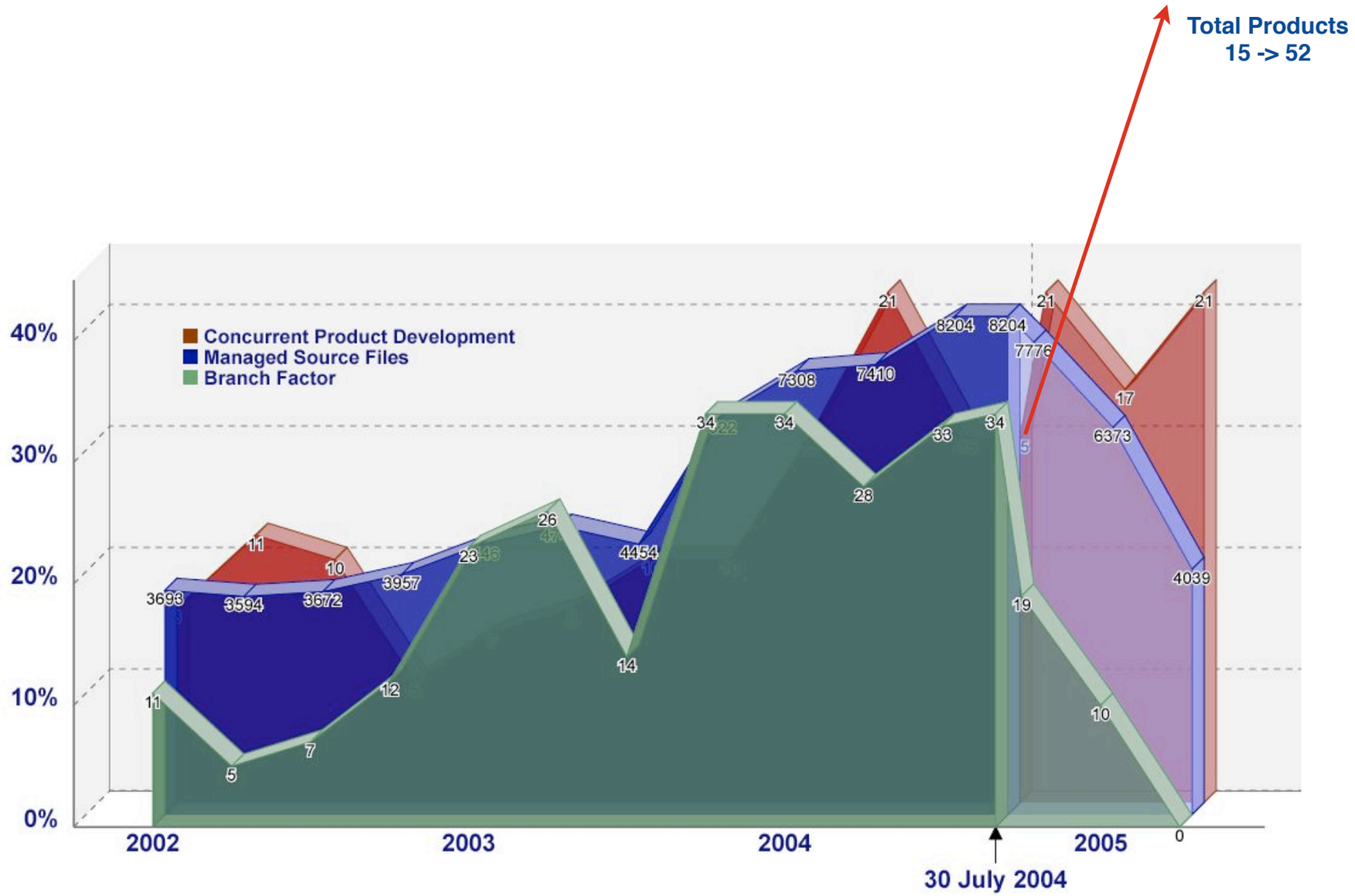


- Clone-and-own
- CM branches
  - sparse
  - dense
- Preprocessors
  - IFDEFs
  - Templates
  - Generics
- Naming conventions
  - files and directories
- Build scripting
- Installer scripting
- Runtime conditionals
  - config files
  - database settings
- Independent
- ...

Conventional versus New SPL



Metrics from Engenio
(LSI Logic Storage Group)

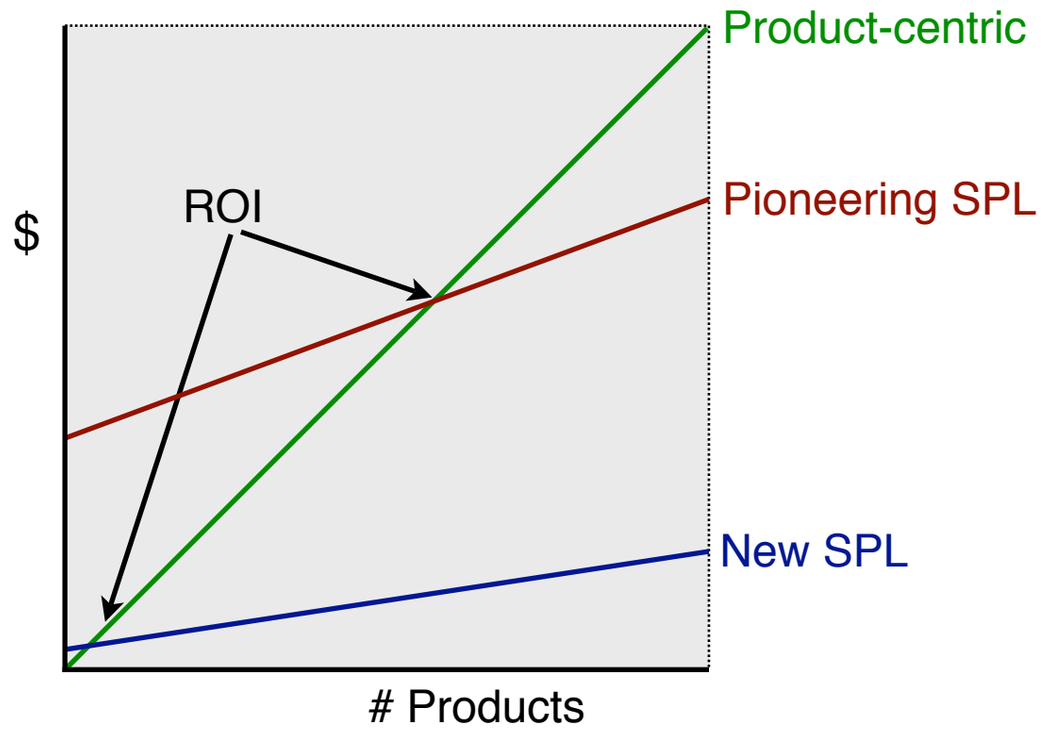


Evolution of SPL Methods

- Conventional product-centric variation management
  - 60's thru 70's
- Pioneering "accidental" software product line approaches
  - 80's thru 90's
- Intentional software product lines approaches
  - 2000+

Pioneering versus New Generation SPL

- Pioneering case studies are now 10-20 years old
- New generation utilizes best practices learned
  - methods
  - tools
  - techniques
- Orders of magnitude easier



Agenda:
Three Categories of New Methods

- Software Mass Customization
  - Application Engineering considered harmful
- Minimally Invasive Transitions
  - Work like a surgeon, not like a coroner
- Harnessing Combinatorics
  - As a practical limitation, the number of possible products in your product line should be less than the number of atoms in the universe

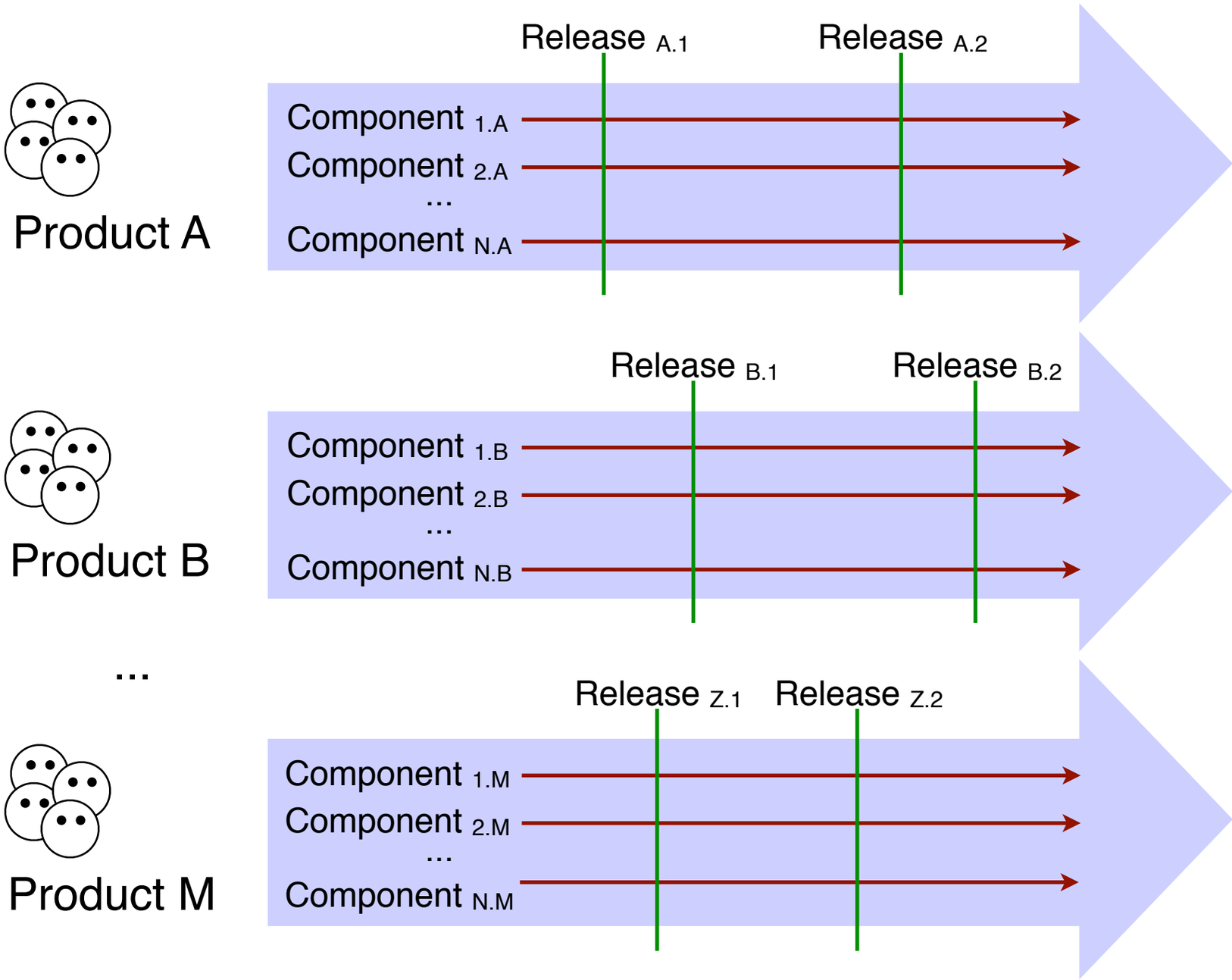
Software Mass Customization

Why Software Mass Customization?

- Observation: Manual application engineering and production plans are harmful to software product line economics
  - Leads to labor intensive duplication, divergence, merging and coordination
  - Has many of the product-centric characteristics of clone-and-own
  - Glue code is undisciplined one-off development
  - Analogous to building cars from blueprints and the auto parts store

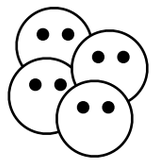
Sans Application Engineering

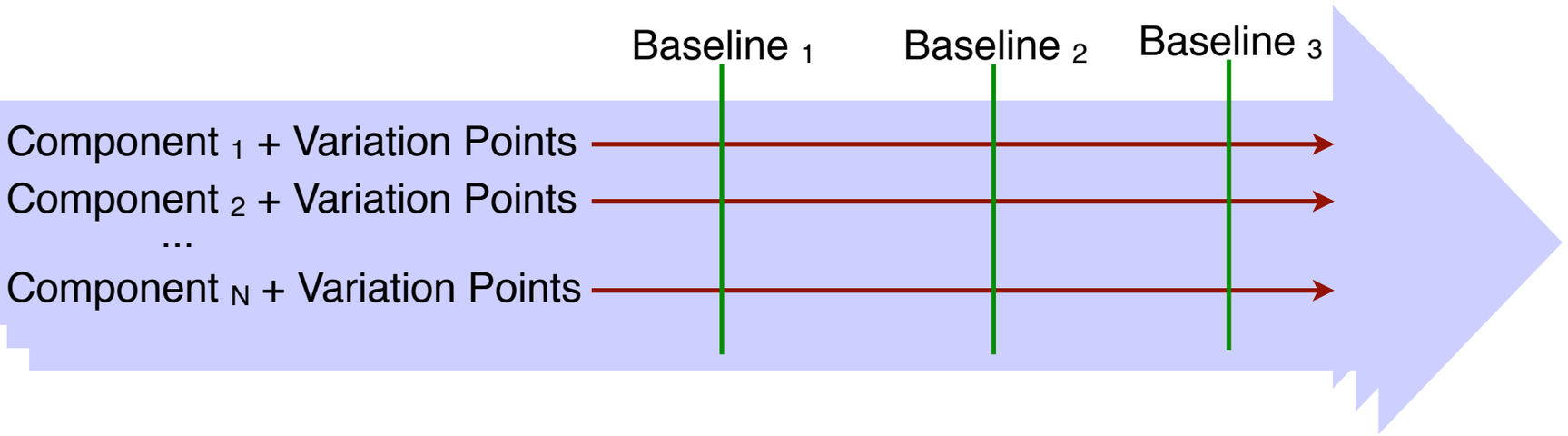
From Product to Product Line Thinking

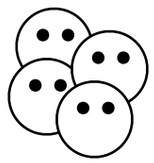


From Product to Product Line Thinking

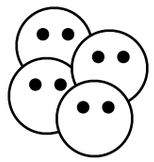
Step 1







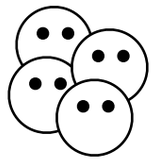
...

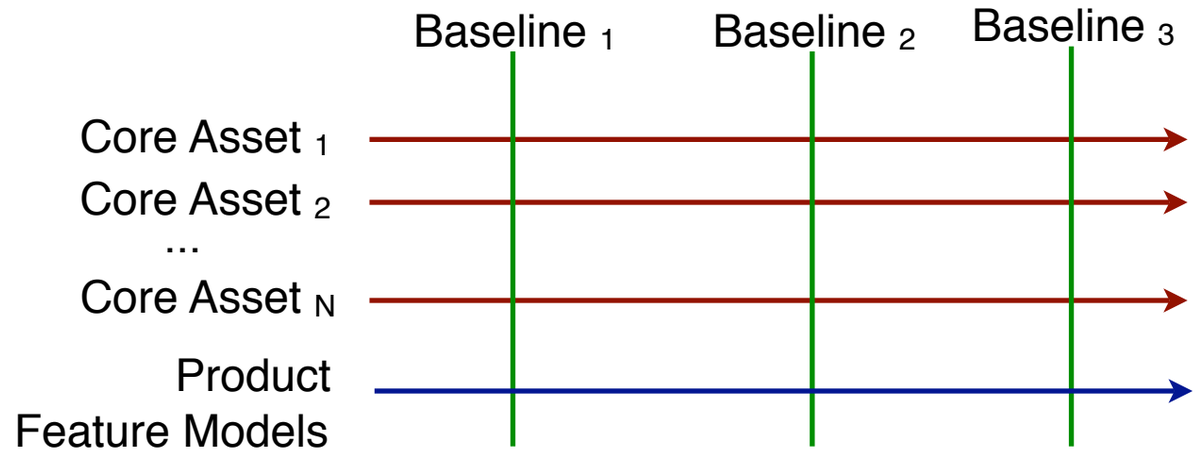


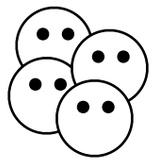
- Consolidate products into one set of shared components
- Encapsulate *Variation Points*
- Component + Variation Points = *Core Asset*

From Product to Product Line Thinking

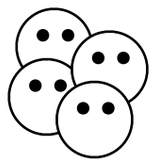
Step 2





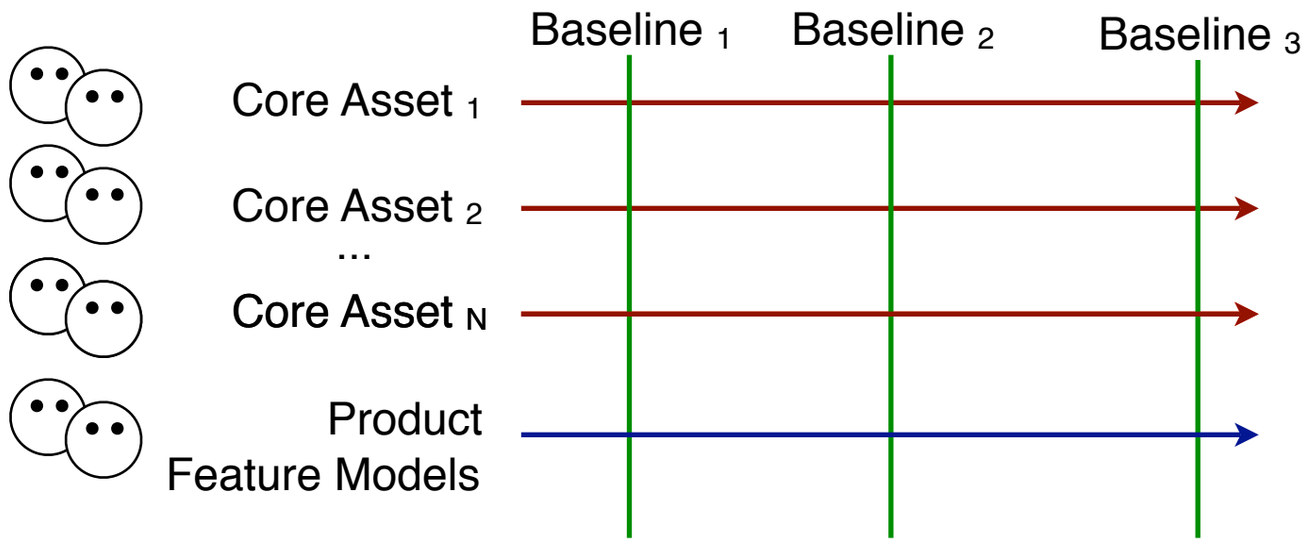


- Capture product configuration information in *product feature models*



From Product to Product Line Thinking

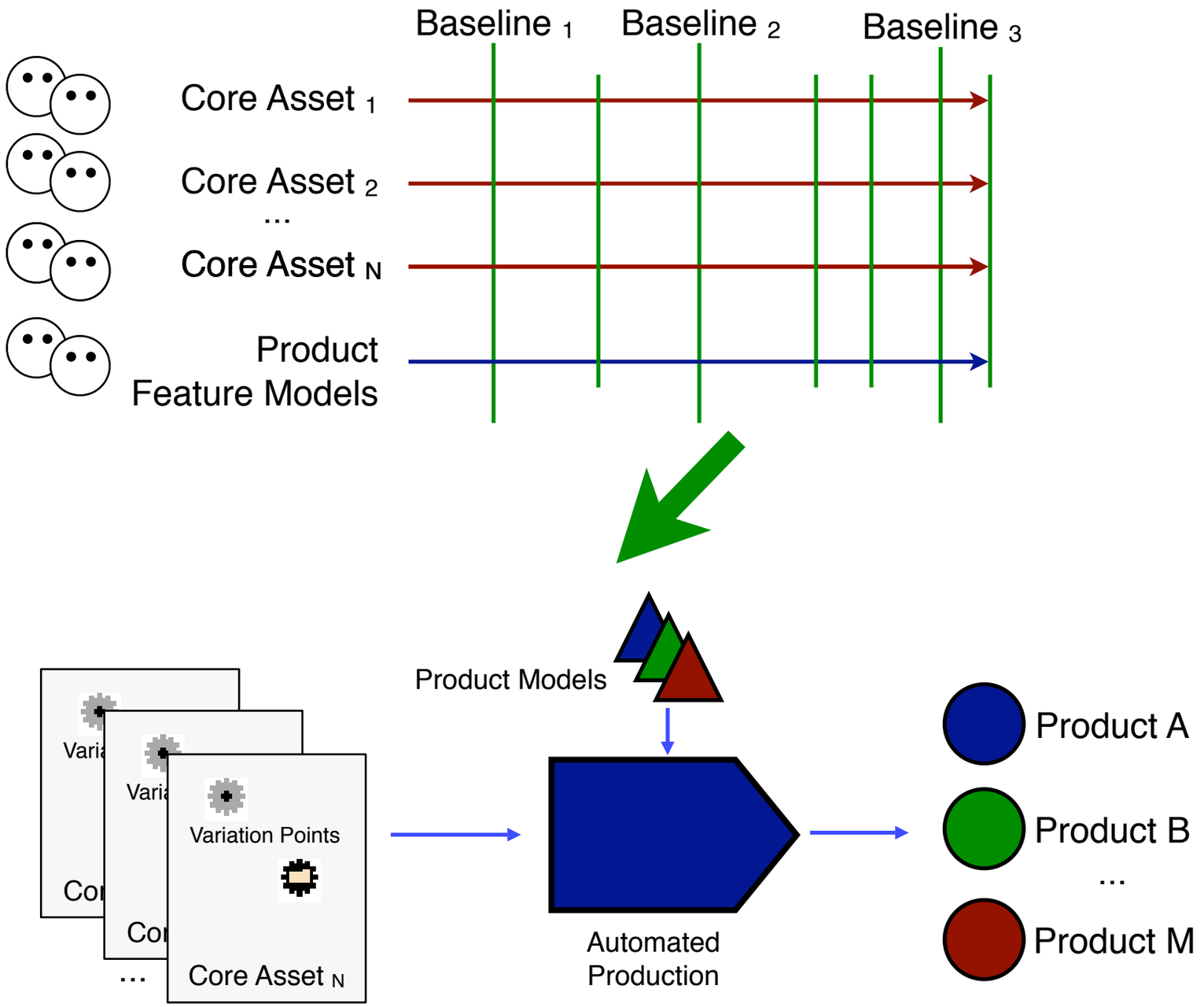
Step 3



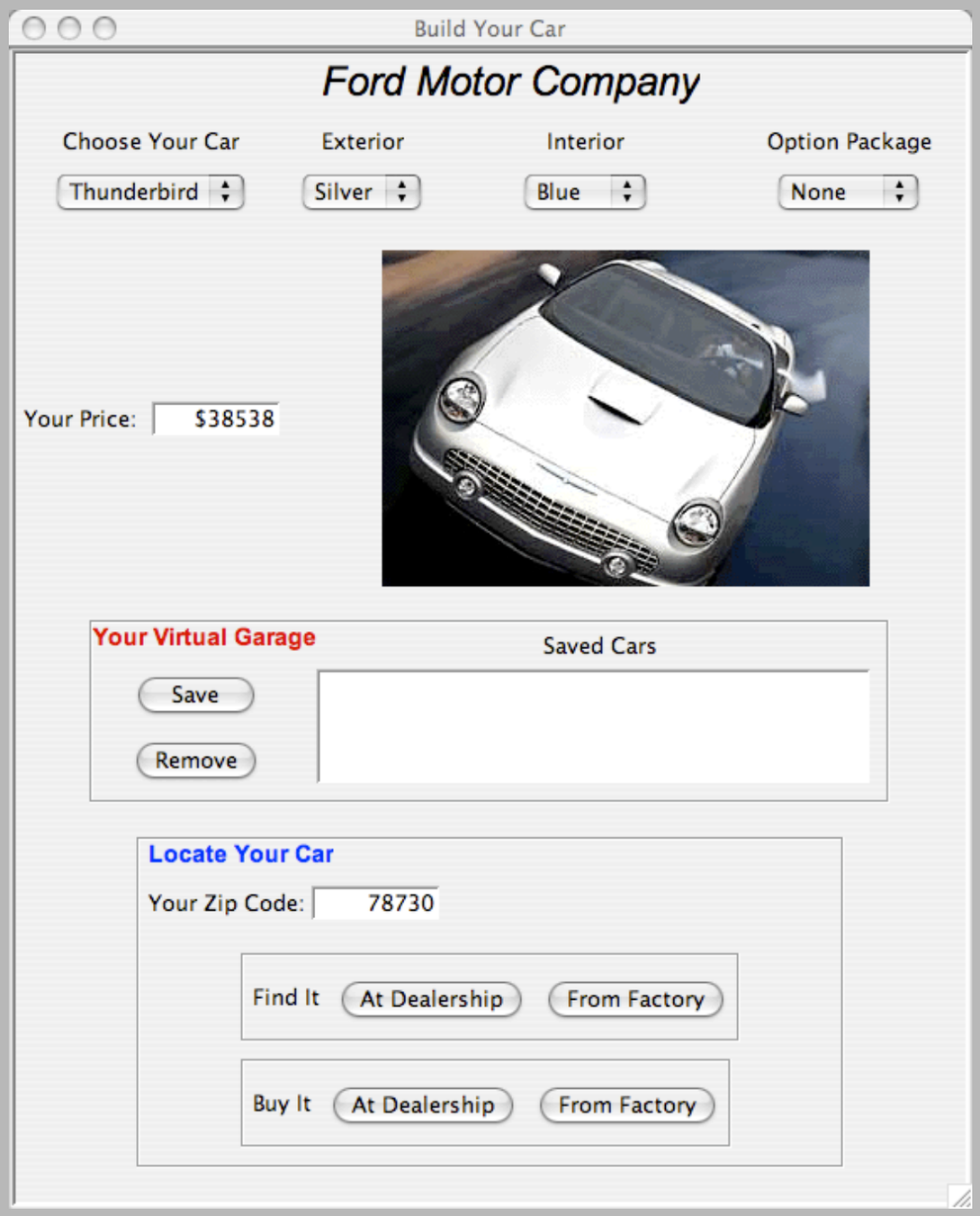
- Allocate developers by their core asset expertise
- Establish singular focus on product line features

From Product to Product Line Thinking

Step 4



Demonstration of Concepts...



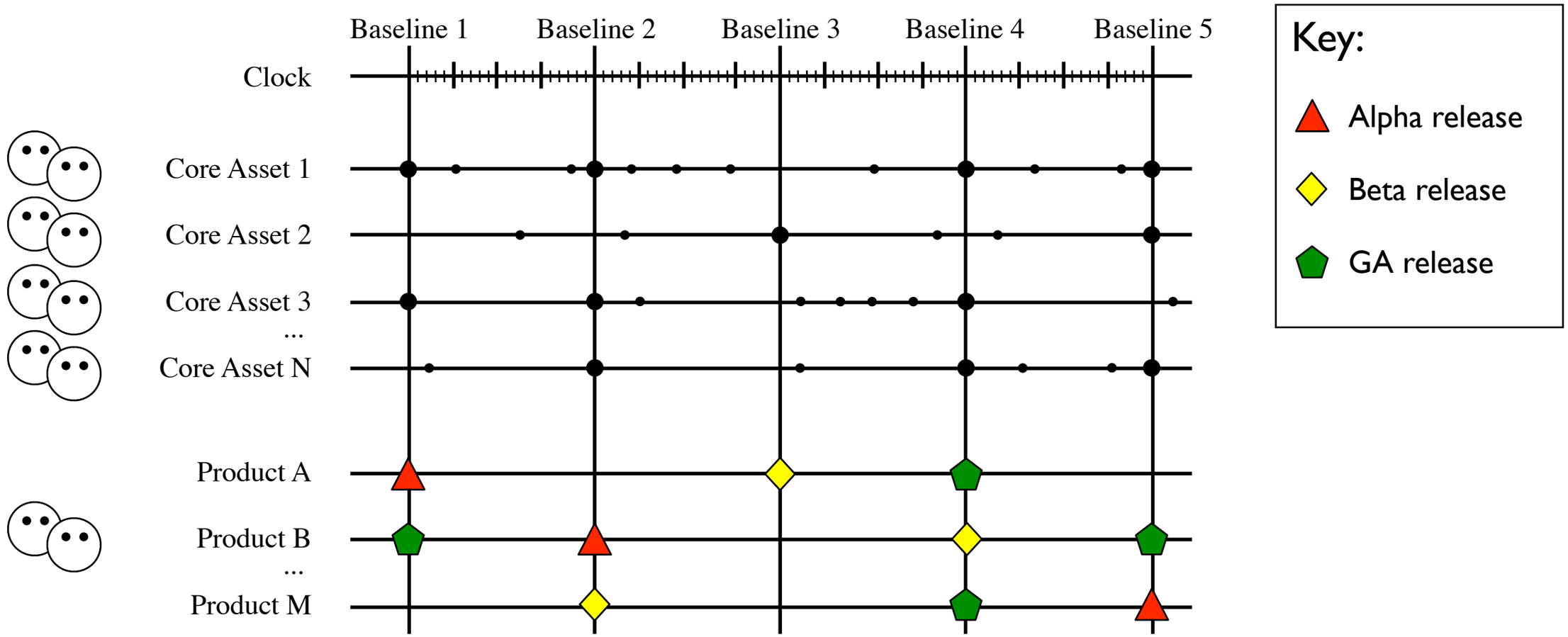
The Intangible Product

The Intangible Product

- Products only become tangible after automated production
  - Not tangible during core asset development
  - No application engineers assigned to product development
- Yet there are tangible release schedules for products
- How do you manage “feature” development in the core assets to meet product release schedules?

Synchronizing the Clocks

- Align release schedules for core assets and products



Variation Management in Time and Space

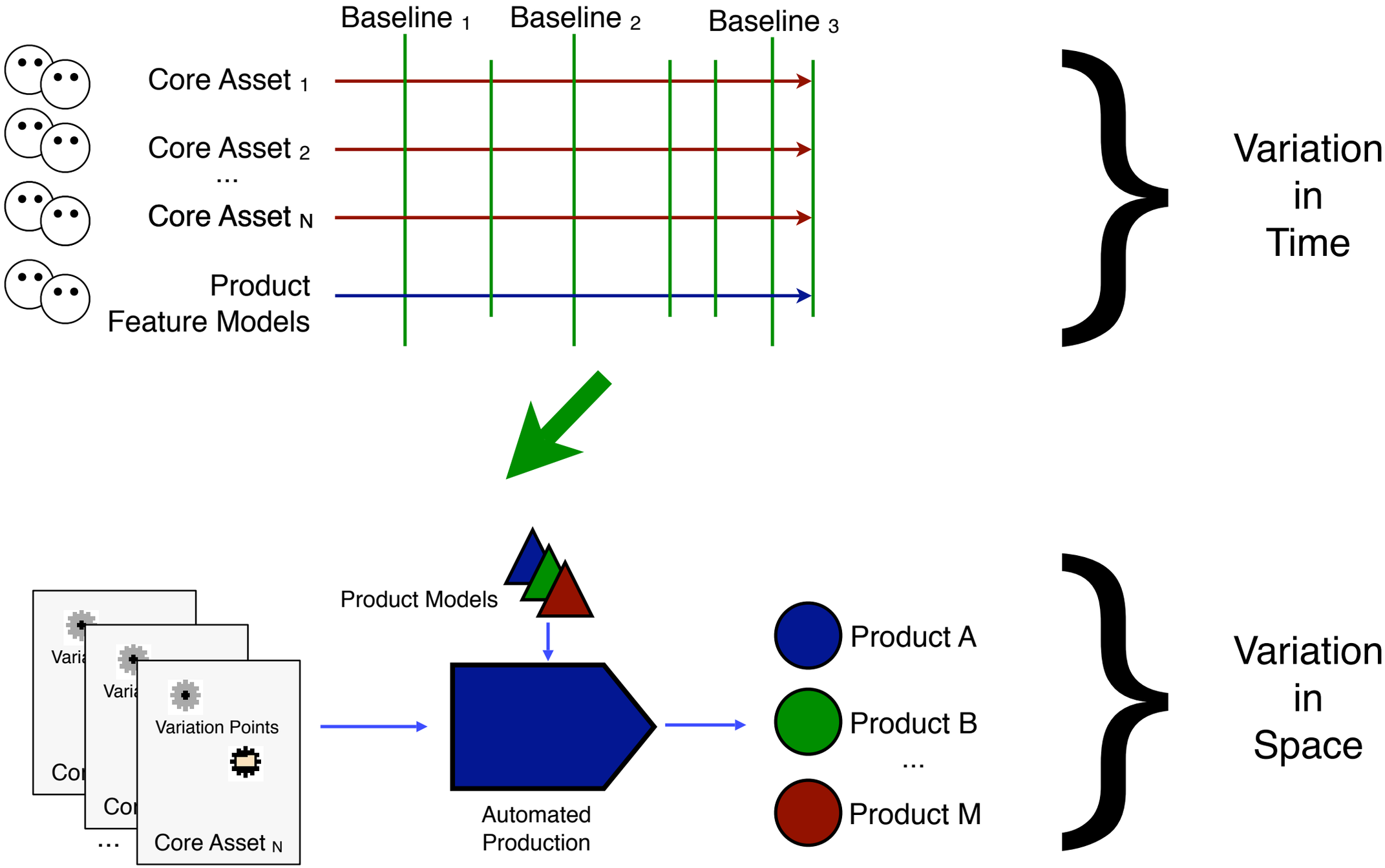
Managing the Evolution and Variations of Intangible Products

- Divide and conquer
  - Conventional configuration management
  - Core assets baseline management
  - Software product line variation management

Divide and Conquer

	Sequential Time	Parallel Time	Domain Space
File	Configuration Management		Software Mass Customization
Core Asset			
Product	Baseline Management		

Variation Management in Time and Space



Minimally Invasive Transitions

Minimize Start State to Target State

- Assumption: there is already a product line
  - Work like a surgeon. Not like a coroner.

- Avoid the lure of the green field
  - Engineers always prefer to autopsy and recreate

Minimize, continued

- Reuse legacy assets for core assets
  - Treat architecture and componentization as orthogonal
  - Initially the variation points and feature models don't have to be elegant
- Don't introduce the domain engineering and application engineering dichotomy in the organization
- Start with minimalist and reactive scoping

It’s Single-system Engineering, Except…

- Software mass customization and automated production enable the core assets to become the single system
- In observing a “new generation” software product line organization before and after, it looks very conventional
- Developers don’t see much difference
  - Too much to do, not enough time, just like before
- Primarily the old product managers who see a difference
  - Managing intangible rather than tangible products

Incremental Return on Incremental Investment

- Decompose initial transition into incremental repetitions
  - One product at a time
  - One component or subsystem at a time
  - One team at a time
- Use improvements from one increment to pay for next
- Staged incremental transitions, with big wins first
  - Example: infrastructure & core assets, organization, process
- Non-disruptive transitions with accelerating production

Regression Red Flags

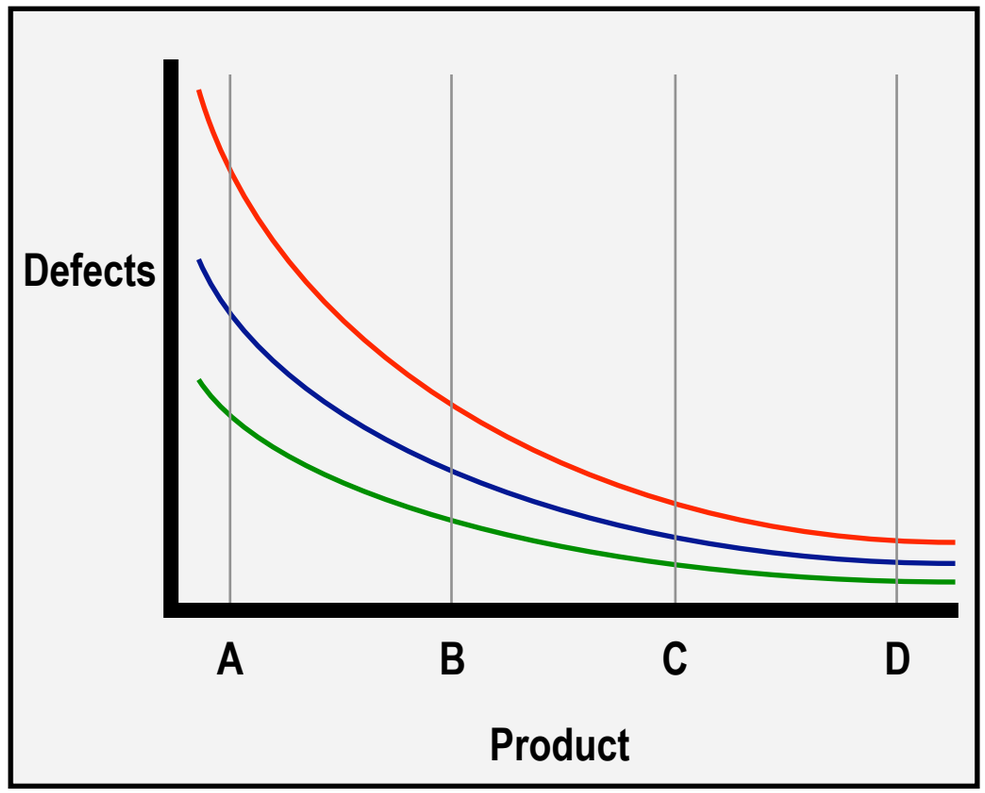
- With minimally invasive transitions, it is initially hard to remain true to product line principles
- How can you tell when you inadvertently regress to familiar old ways?
  - Tangible products rather than intangible products
  - Tangible products rather than features and benefits
    - References to products in core asset source code
    - Product names in core asset file or directory names
    - Product-specific branches
    - Product development teams cutting across core assets
    - ...

Harnessing Combinatorics

Why Worry about Combinatorics?

- 216 boolean features == 10^{65} feature combinations
  - That's the number of atoms in the universe
  - That's too many
- 33 boolean features == $8x10^{9}$ feature combinations
  - $6x10^{9}$ is plenty
- Domain engineers gleam over models with 1500 features
- Test engineers spontaneously combust over models with 1500 features

Test Metrics from Salion



Harnessing Combinatoric Complexity

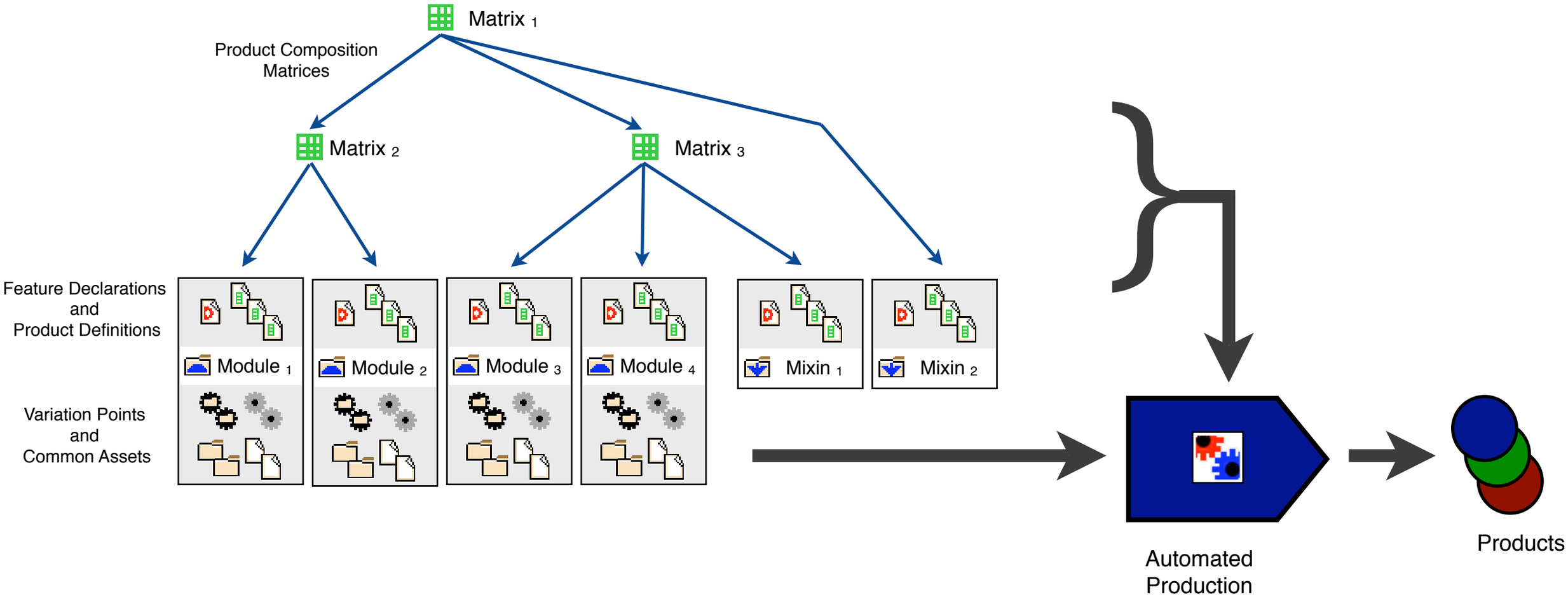
- Time-tested computer science techniques apply well
  - Abstraction
  - Modularity
  - Controlling scope
  - Controlling entropy
- Uniquely applied for software mass customization

Abstraction

Software Mass Customization Abstraction Layers

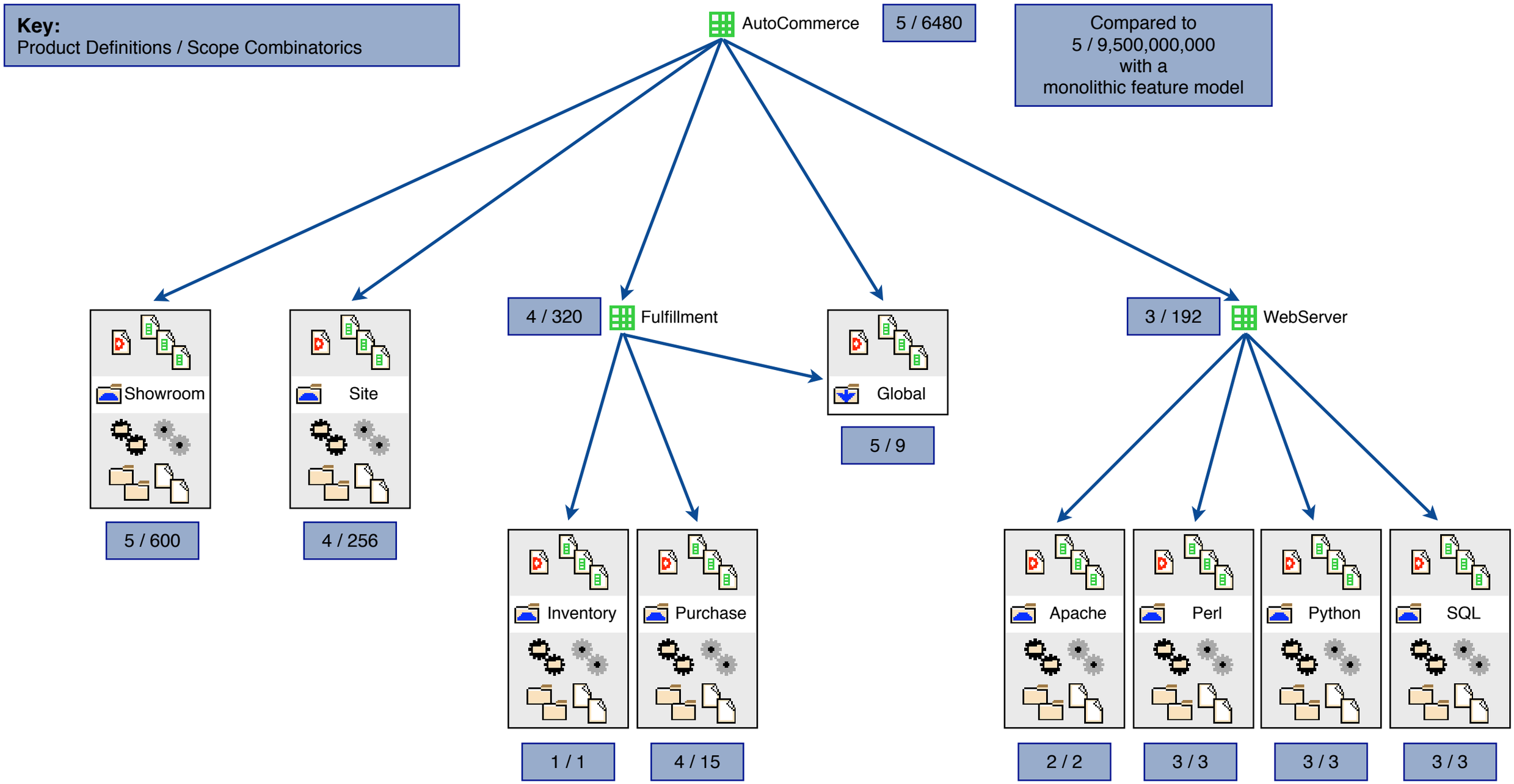
Composition	A composition of flavors of subsystems. A composition is itself a flavor. Abstraction for hierarchical product line assemblies.
Flavor	An instantiation, based on a set of decisions in the feature models. Abstraction for a point solution, with specific benefits.
Feature	The feature model. Abstraction for variability in the application domain.
Variation Point	Source-level variation in the core assets. Alternatives expressed in terms of feature values.

Composition and Hierarchical Product Lines



Demonstration of Concepts...

Key:
Product Definitions / Scope Combinatorics



Product Line Scope

Scoping

- Deciding what's in and what's out of the product line
  - Products, features and capabilities that offer customer value
- Cost / Benefit negotiation
  - The cost of adding a product, feature or capability
  - The benefit – such as revenue – of having it
- Responsibility of Product Marketing, not Engineering

Product Line Scoping

- For product lines, variation is a unique scoping challenge
- Drivers of product variation
  - Multiplicity in who, what, when, where, how, and why
  - Market turbulence
- Inconsistent with early product line literature requiring stable domains and long range planning

Proactive versus Reactive Scoping

- Proactive: Implement variation predicted on the horizon
- Reactive (agile): Implement variation needed today
- It's a spectrum, not binary

Optimizing Your Place on the Proactive–Reactive Spectrum

- Mistakes are expensive. At both ends.
  - Architecture mistakes possible if short sighted
  - Throwaway work possible if long-term predictions fail
    - Avoid the lure of the crystal ball
- Analysis and architecture can and should be more proactive than implementation
- Over-generality quickly makes test combinatorics intractable
- Software mass customization tends to optimize towards reactive end of the spectrum

Evolution

Evolution in Software Product Lines

- Refactor variation points
- Elicit emerging abstractions
- Control entropy
- Refactor across binding times

Refactoring Variation Points

- With explicit variation points, a dual-mode style of development consistently arises in practice
  - Rapid development for rapid deployment
  - Refactoring to minimize variation and factor out commonality
- Variation points with non-optimal reuse ratios are OK if they are very stable and don't undergo evolution
- Variation points that undergo frequent evolution should be optimized to factor out commonality
- Fortunately, natural human laziness encourages the correct behavior for variation point evolution

The Constant Search for Emerging Abstractions

- Salion SLOC count decreased by 30% as product and variation count increased by 500%
- Identifying one emerging abstraction allowed them to reduce the amount of varying code for a new product instance from 1600 to 150 SLOC
- Need to look both within and across source-level variation points

Controlling Product Line Entropy

- Monitor vital statistics
- Reuse ratios are an indicator of the health of product lines, sub-product lines, and modules
- Decreases in reuse ratios can be a problem indicator
  - Undisciplined development
  - Overly proactive scoping
  - Over generalization in implementation
  - Architectural, design, or implementation problems

Conclusions

- Pioneering SPL case studies are 10-20 years old
- A new generation of case studies are emerging based on new methods, tools and techniques
  - Software mass customization
  - Minimally invasive transitions
  - Bounded combinatorics
  - ...
- Making it orders of magnitude easier to gain even greater benefits of SPL approach

For More Information...

- www.biglever.com
- www.SoftwareProductLines.com
- Charles W. Krueger, PhD
  - CEO, BigLever Software
  - 512-426-2227 voice
  - ckrueger@biglever.com