DevOps

1. Legacy

Origin Story

- What has happened before?
- Web, OpenSource, Agile and Cloud.

How the Web change the way we see IT

IT as a tool

Help to perform work

- → a separated department
- General Resources
- IT

Cost Center

Introduction of methodologies to migitate the cost.

- ITIL
- V-Model
- Waterfall
- GANTT
- ...

Outsourcing / Offshoring

HERE COMES A NEW

CHALLENGER

Native Web Companies

Strategic Differentiator

- Tech used to perform
- For example: PayPal vs Banks

Open-source (End of 1990s)

Richard Stallman

Organizations

<img src="https://upload.wikimedia.org/wikipedia/commons/thumb/4/42/Opensource.svg/2560px-Opensource.svg.png" alt="Open Source Initiative" style="width:40%;" > <img src="https://upload.wikimedia.org/wikipedia/commons/1/1e/FSF-Logo_part.svg" alt="Free Software Foundation" style="width:40%;" >

Definition

1. Free Redistribution
2. Source Code
3. Derivated Works
4. Integrity of The Author's Source Code
5. No Discrimination Against Persons or Groups
6. No Discrimination Against Fields of Endeavor
7. Distribution of License
8. License Must Not Be Specific to a Product
9. License Must Not Restrict Other Software
10. License Must Be Technology-Neutral

Benefits

- Security
- Affordability
- Transparency
- Perpetuity
- Interoperability
- Flexibility

An answer to cost center

Another way to colaborate

- People from different companies, cultures, background can work on the same product.

Agile Manifesto

- 2001

Agile software development values

- **Individuals and Interactions** over processes and tools
- **Working Software over comprehensive** documentation
- **Customer Collaboration** over contract negotiation
- **Responding to Change** over following a plan

Adaptive vs. predictive

Iterative, incremental, and evolutionary

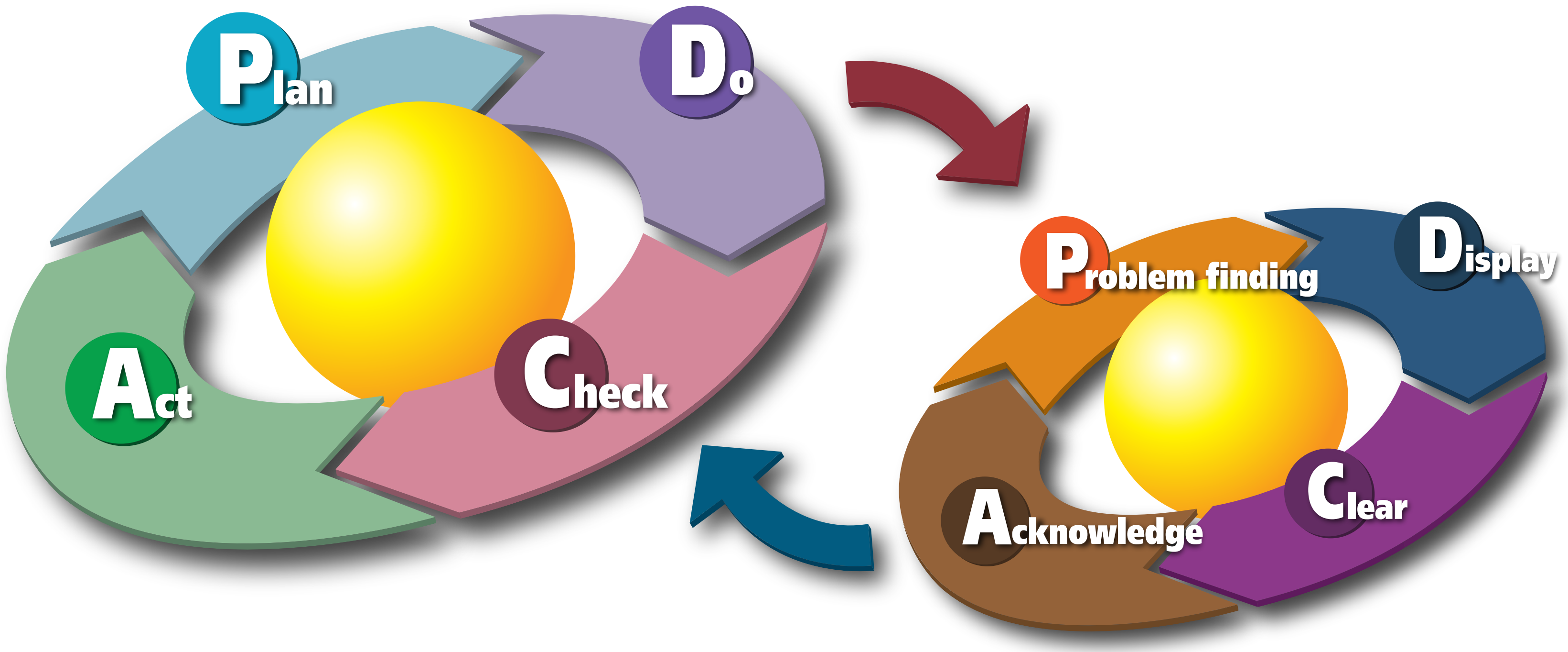
- small increments
  - depending of software (~2w)
- Opposition with tickets, QA systems
- Make Outsourcing difficult

Software Development Frameworks

- Scrum
- Kanban
- XP
- ...

Kaizen

改善 is the Sino-Japanese word for "improvement"



{{% note %}}

This is also known as the Shewhart cycle, Deming cycle, or PDCA.

{{% /note %}}

Software Craftmanship

A fifth value for the Agile Manifesto:

- **"Craftsmanship over Crap"**
- He later changed his proposal to "Craftsmanship over Execution".

Manifesto for Software Craftmanship

As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft. Through this work we have come to value:

- Not only working software, but also well-crafted software
- Not only responding to change, but also steadily adding value
- Not only individuals and interactions, but also a community of professionals
- Not only customer collaboration, but also productive partnerships

Cloud

- AWS 2006
- OpenStack ~2010

Characteristics

- Flexibility
- Cost "reductions"
- Device and location independence
- Maintenance
- Multitenancy
- Performance
- Productivity
- Availability
- Scalability and elasticity
- Security

2. Culture

Patrick Debois, 2007

- When working for Belgium government, is frustrated by bad communications between developpers and operators.

2008

- Agile 2008 Toronto: Andrew Shafer « Agile Infrastructure »
- Patrick Debois is the only one present.
- They created the Agile Systems Administration Group.

2009

- During *velocity O’Reilly*, John Allspaw et Paul Hammond: "10+ Deploys a Day: Dev and Ops Cooperation at Flickr"
- Patrick Debois can't be there.
- "DevOps" is known worldwide

Silos

PLATFORM WORKS FINE
DEV PROBLEM NOW
quickmeme.com

CAMS

- Culture
- Automation
- Measurement
- Sharing

Culture

CAMS culture

Automation

Do not repeat same mistakes (again!)

Measurement

What can we improve?

Did we succeed?

Sharing

An End rather than a Means

> DevOps is about eliminating technical, process, and cultural barriers between idea and execution - using software

Velocity

What DevOps isn't

- X tool
- A job
- A team

Agile

12 Values

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Deliver working software frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers

5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the primary measure of progress
8. Sustainable development, able to maintain a constant pace

9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

What Agile is not

- Cowboy coding.
  - Definition of "done" and explicit value to customer is required on every sprint.
- Rigor without planning.
  - Continual planning through the project, instead of up front.
- Excuse for a lack of a roadmap.
  - Recognizing change != pivoting every sprint.
- Development without specifications.
  - Right-sized specifications to reflect *why* and *how*.

Scrum

3 Roles

Product owner

- Customer Representative
    - not a manager
    - not a project leader
- Responsible for what the team is building and why.
- Keeps the backlog up-to-date and in priority order.

Development team

- Self-organizing.
- Responsible for engineering + quality.

Scrum master

- Facilitator
- Coach
- Resolving impediments and other blocking issues.

Artifacts

User Stories

from the perspective of an end user

> As a *role* I can *capability*, so that *receive benefit*

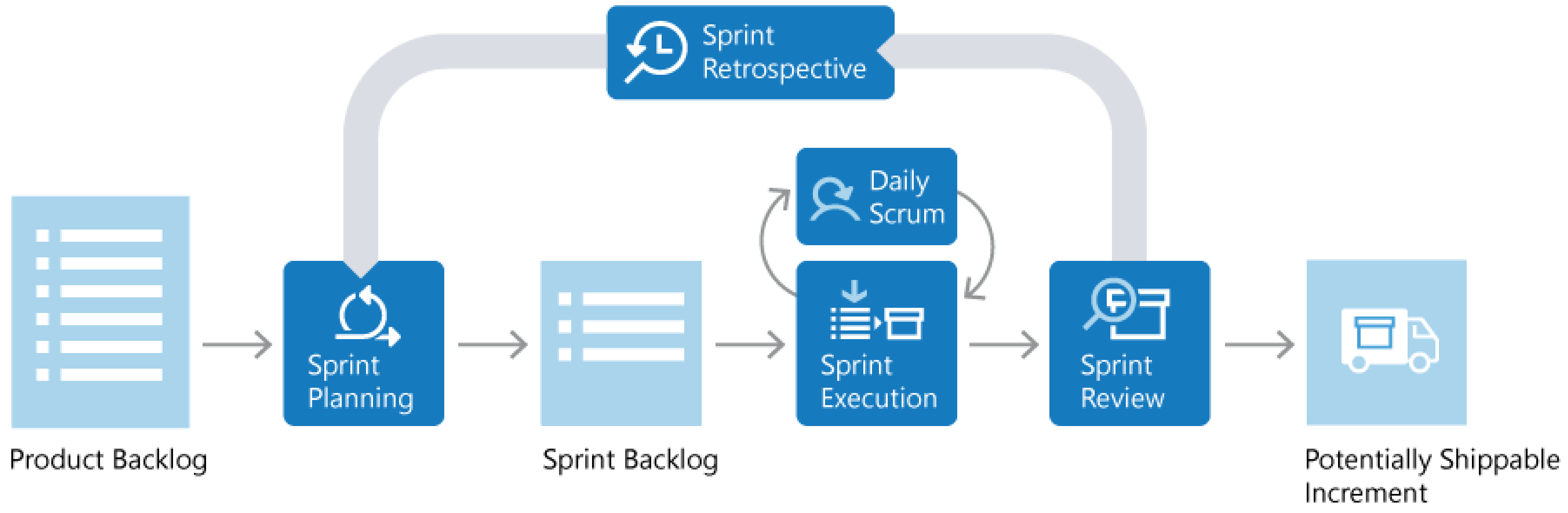
Product Backlog

Ordered list of US that should be done

Workflow

- Sprint planning
- Daily scrum
- Sprint review
- Sprint retrospective
- Backlog refinement (Grooming)

Workflow (cont.)



3. Rolling DevOps

Step 1: Lean Software Development

Principles

1. Eliminate waste
2. Amplify learning
3. Decide as late as possible
4. Deliver as fast as possible
5. Empower the team
6. Build integrity in
7. Optimize the whole

Eliminate waste…

1. Partially done work
2. Extra features
3. Relearning
4. Task switching
5. Waiting
6. Handoffs
7. Defects
8. Management activities

...and pain

1. Building the wrong feature or product
2. Mismanaging the backlog
3. Rework
4. Unnecessarily complex solutions
5. Extraneous cognitive load
6. Psychological distress
7. Waiting/multitasking
8. Knowledge loss
9. Ineffective communication.

Amplify learning

- Learning by coding
- Testing before documentation
- Short iterations

Decide as late as possible

Uncertainty → Delaying

- An agile software development approach can move the building of options earlier for customers, thus delaying certain crucial decisions until customers have realized their needs better

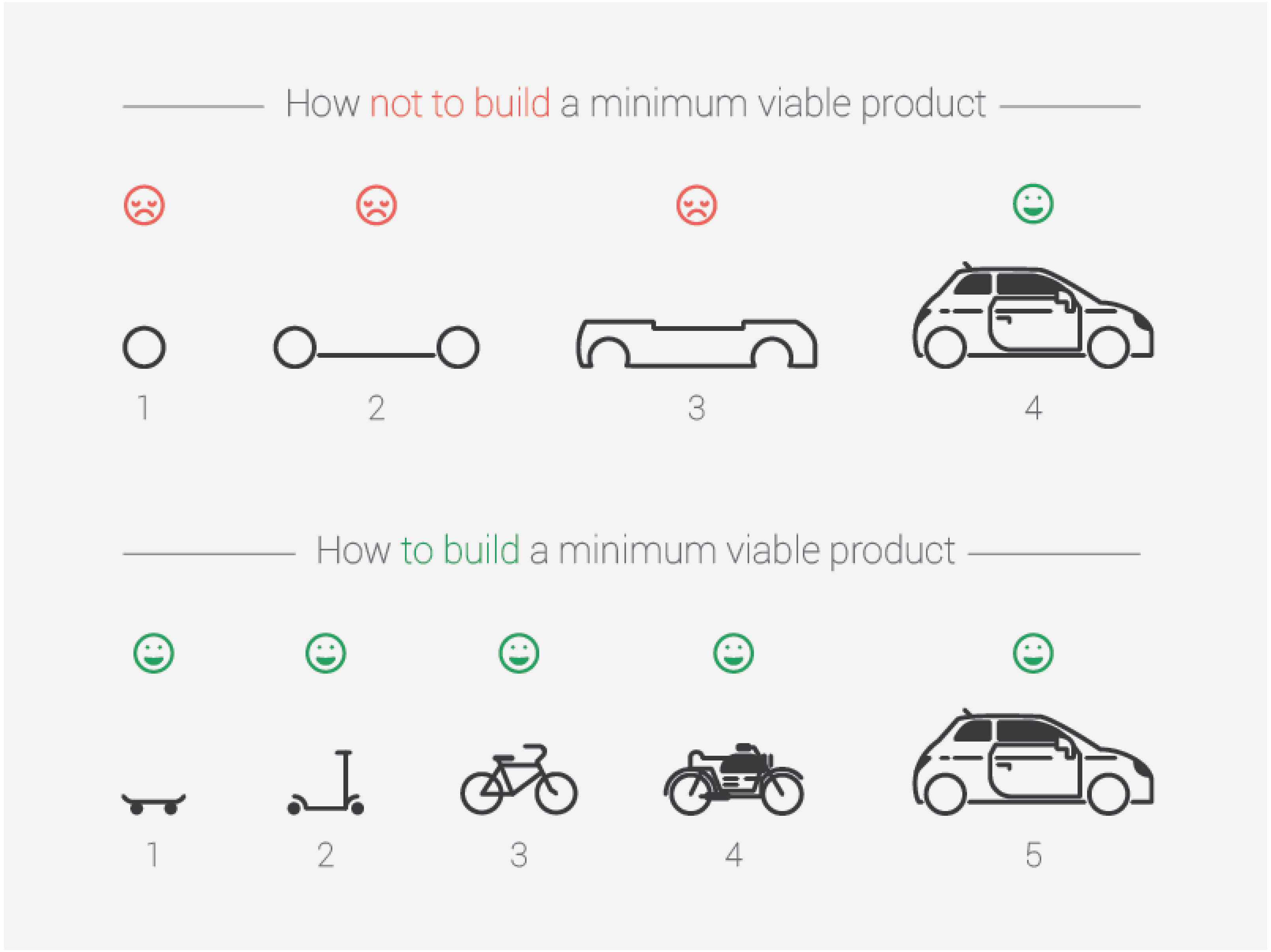
Deliver as fast as possible

- Just In Time

Empower the team

People != resources

Build integrity in


How not to build a minimum viable product
1
2
3
4
How to build a minimum viable product
1
2
3
4
5

Optimize the Whole

> the whole is always more than the sum of its parts

Customers > Company > Team > Self

Step 2: Automation

Why

Time

- Manual tasks: ~ linear time
- Automated tasks: invest + profit

Quality

- Manual tasks: variable
- Automated tasks: uniform

On-boarding

- Manual tasks: human process
- Automated tasks: Industry standard

What

- Provisioning
- Commissioning
- Configuration
- Deployment

How

- Infrastructure-as-Code
- Virtualization
- Orchestration: Ansible, Salt, Terraform, Cloudformation

Step 3: CI

Why

- Test smallest increment possible
- increase bus factor

What

- release
- test
- build

How

Environment spawning

Examples

- Jenkins
- Gitlab CI
- CircleCI
- Travis

Impact on my application

Github flow

Branch → Pull Request → Validation → Merge

12 factors

1. Codebase

   One codebase tracked in revision control, many deploys
2. Dependencies

   Explicitly declare and isolate dependencies
3. Config

   Store config in the environment
4. Backing services

   Treat backing services as attached resources

5. Build, release, run

   Strictly separate build and run stages

6. Processes

   Execute the app as one or more stateless processes

7. Port binding

   Export services via port binding

8. Concurrency

   Scale out via the process model

9. Disposability

   Maximize robustness with fast startup and graceful shutdown

10. Dev/prod parity

    Keep development, staging, and production as similar as possible

11. Logs

    Treat logs as event streams

12. Admin processes

    Run admin/management tasks as one-off processes

Step 4: CD

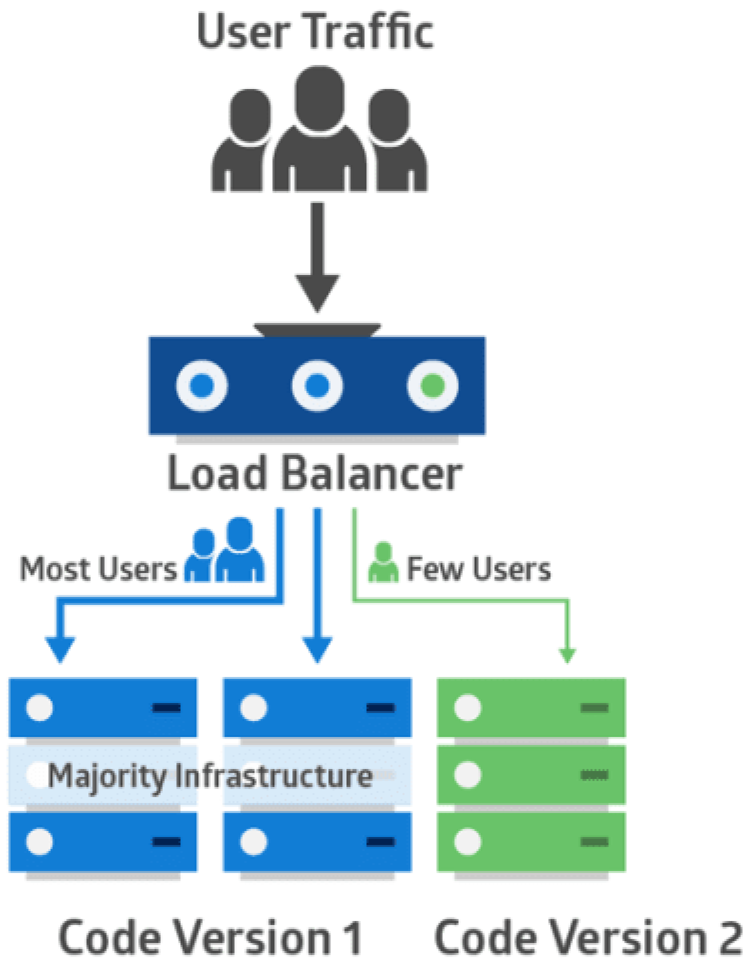
breaking the wall of deployment

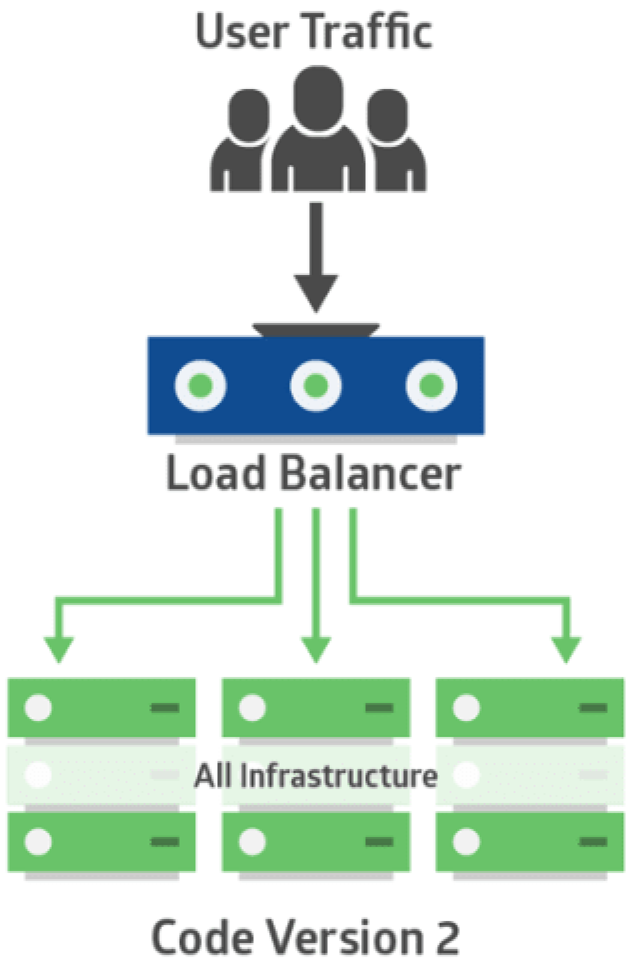
LSCU 107737 9
JP 4300
MAX.GW 67200LB 30480KG
TARE 8510LB 3860KG
MAX.CW 58690LB 26620KG
CU.CAP 2389CU.FT 67.7CU.M

Container

Same artifact all along the workflow

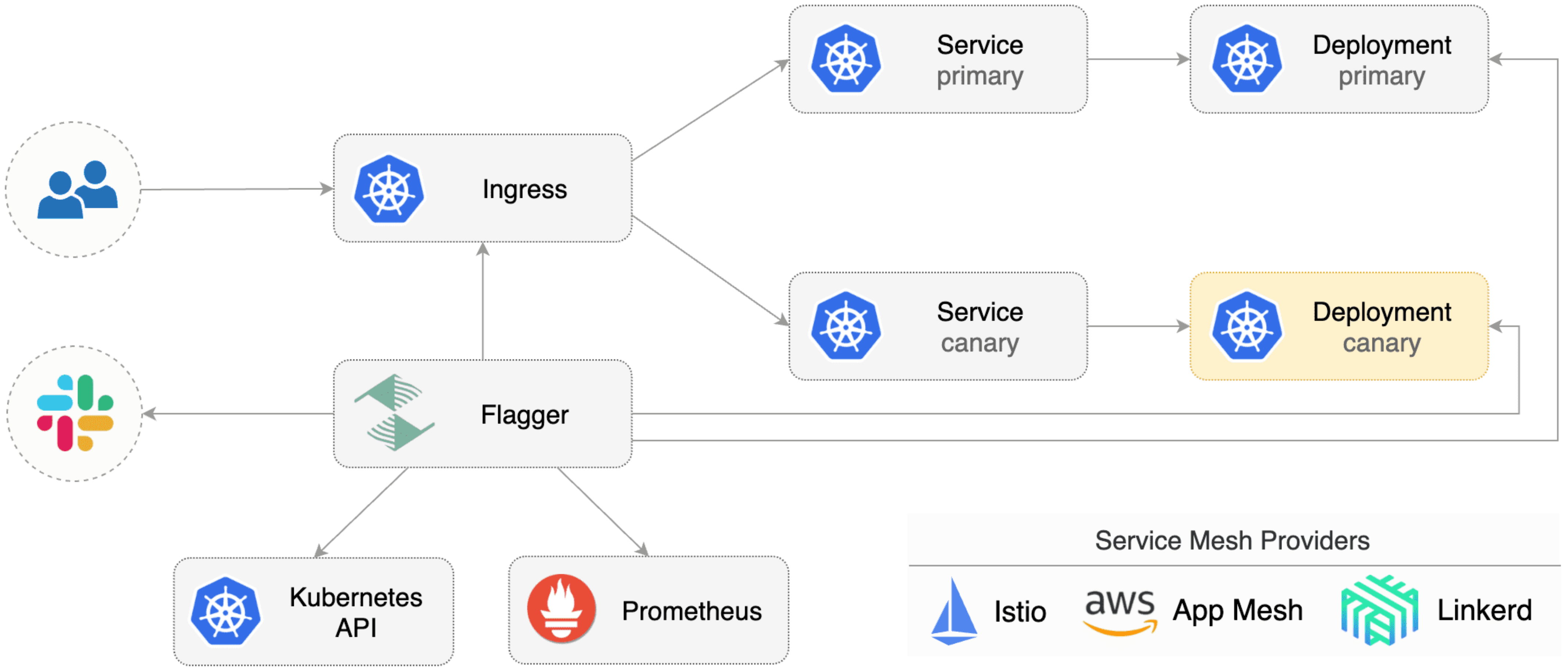
Blue / Green Deployment



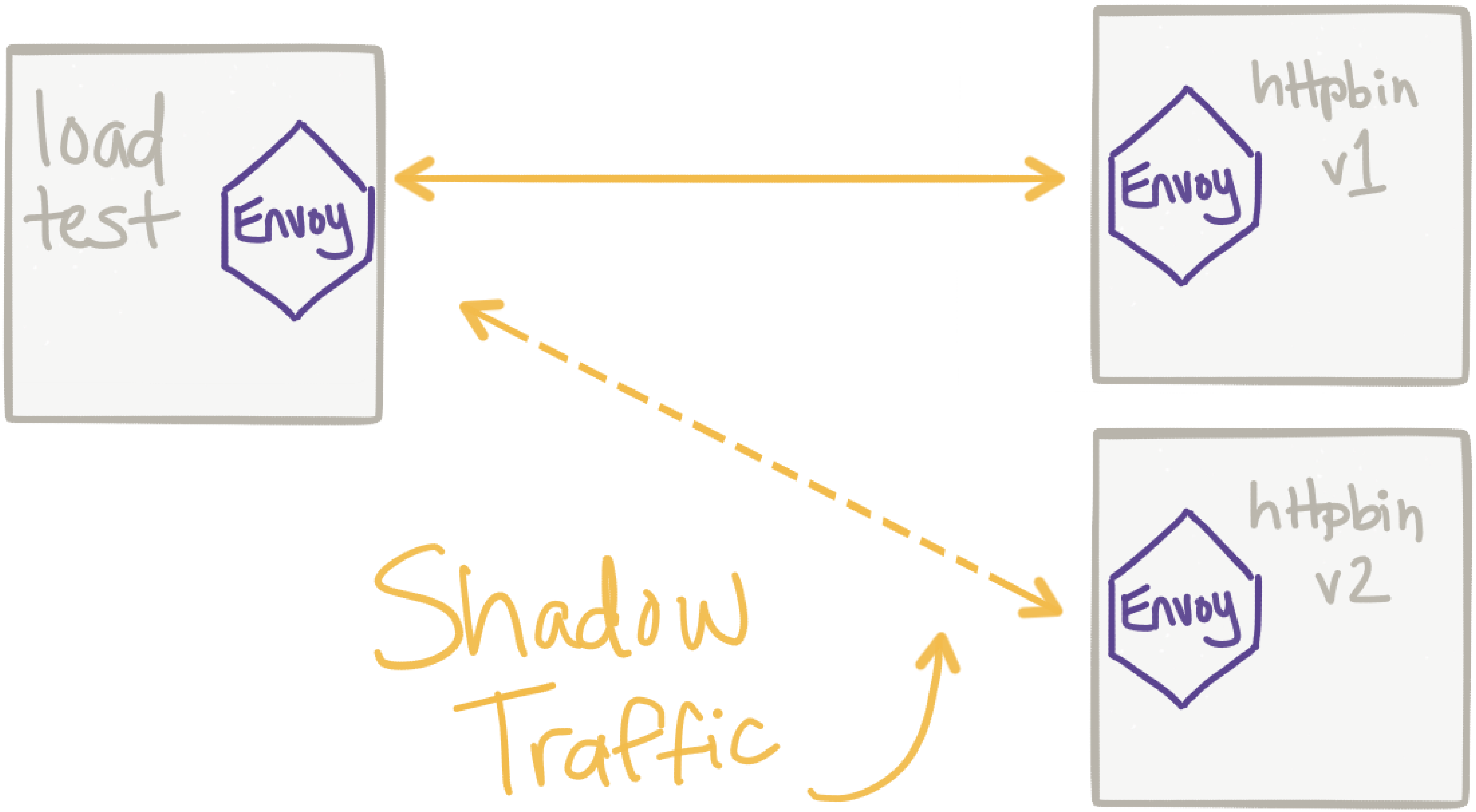


Step 5: All in

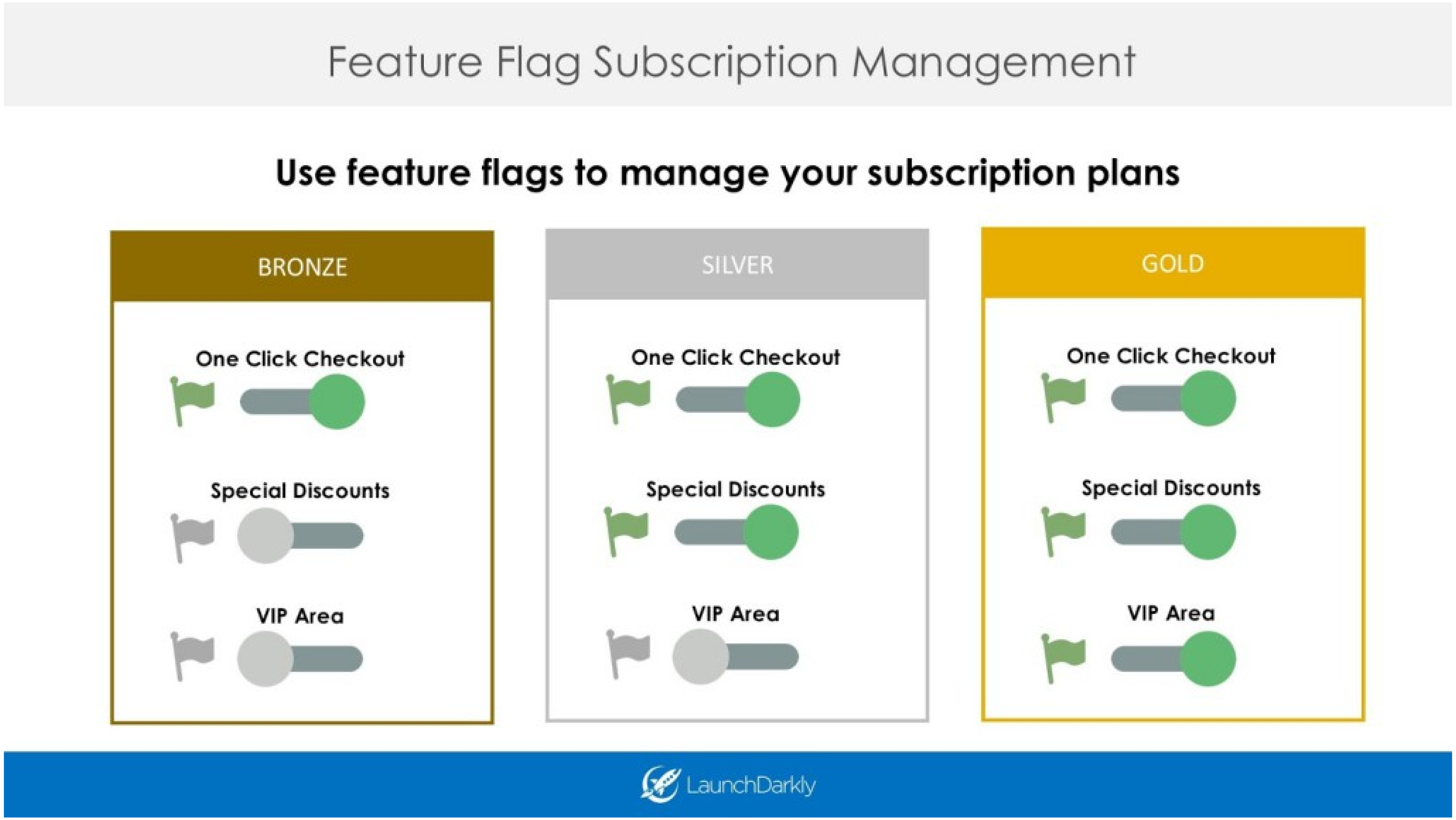
Canary



Shadowing


load test
Envoy
httpbin v1
Envoy
httpbin v2
Envoy
Shadow Traffic

Feature Flags



Implementing changes

Factors

- Features
- Security
- Performance
- Reliability
- Usability

Leverages

ex:

- refactoring
- monitoring
- 3rd party software
- ...

Velocity factors

- People (skills etc)
- Maintainability
- Tests
- Automation
- ...

Flow

Each iteration choose to:

- change **{factor}** with **{leverage}** at the speed of **{{velocity factor}}**.
- invest in velocity factors

4. Microsoft DevOps Journey

Before vs After

Before	After
4-6 month milestones	3-week sprints
Horizontal teams	Vertical teams
Personal offices	Team rooms and remote work
Long planning cycles	Continual Planning and learning

Before vs After (cont.)

Before	After
PM, Dev, Test	PM, Design, and Engineering
Yearly customer engagement	Continual customer engagement
Feature branches	Everyone in main
20+ person teams 8-12 person teams	
Secret roadmap	Publicly shared roadmap

Before vs After (cont.)

Before	After
Bug debt	Zero debt
100 page spec documents	Specs in PPT
Private repositories	Open source/Innersource
Deep organization hierarchy	Flattened organization hierarchy
Success is a measure of install numbers	User satisfaction determines success
Features shipped once a year	Features shipped every sprint

Change in culture

- Key discovery: people value *autonomy*, *mastery* and *purpose*.
- Balance between **alignment** and **autonomy**
- Alignment from top-down to ensure teams understand their role in broader business goals, without having a negative punch in/punch out culture.
- Autonomy from bottom-up, to help individuals feel motivated at work. Too much autonomy can lead to inefficient planning.

Change focus from individuals to teams

- Teams organized in three groups.
    - Project Management.
    - Design.
    - Engineering.

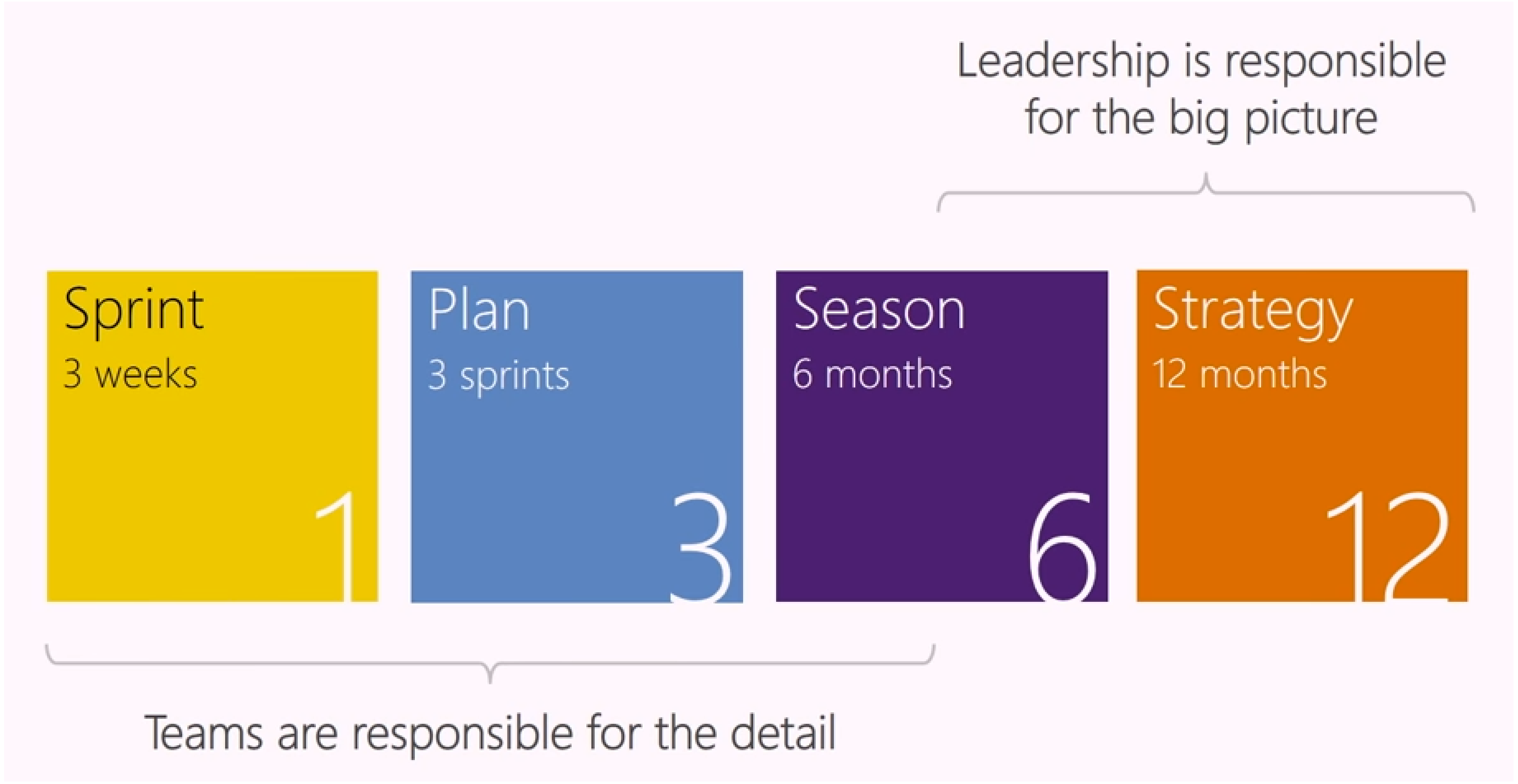
Key characteristics of teams

- Cross discipline
- 10-12 people
- Self-managing
- Clear charter and goals for 12-18 months
- Physical team rooms
- Own features in production
- Own deployment of features

Vertical vs Horizontal

- Before: teams that would cover *all* of a topic across products.
- Now: teams that own their area end-to-end *per product*.

Planning



Leadership is responsible for the big picture
Sprint
3 weeks
1
Plan
3 sprints
3
Season
6 months
6
Strategy
12 months
12
Teams are responsible for the detail

Technical debt

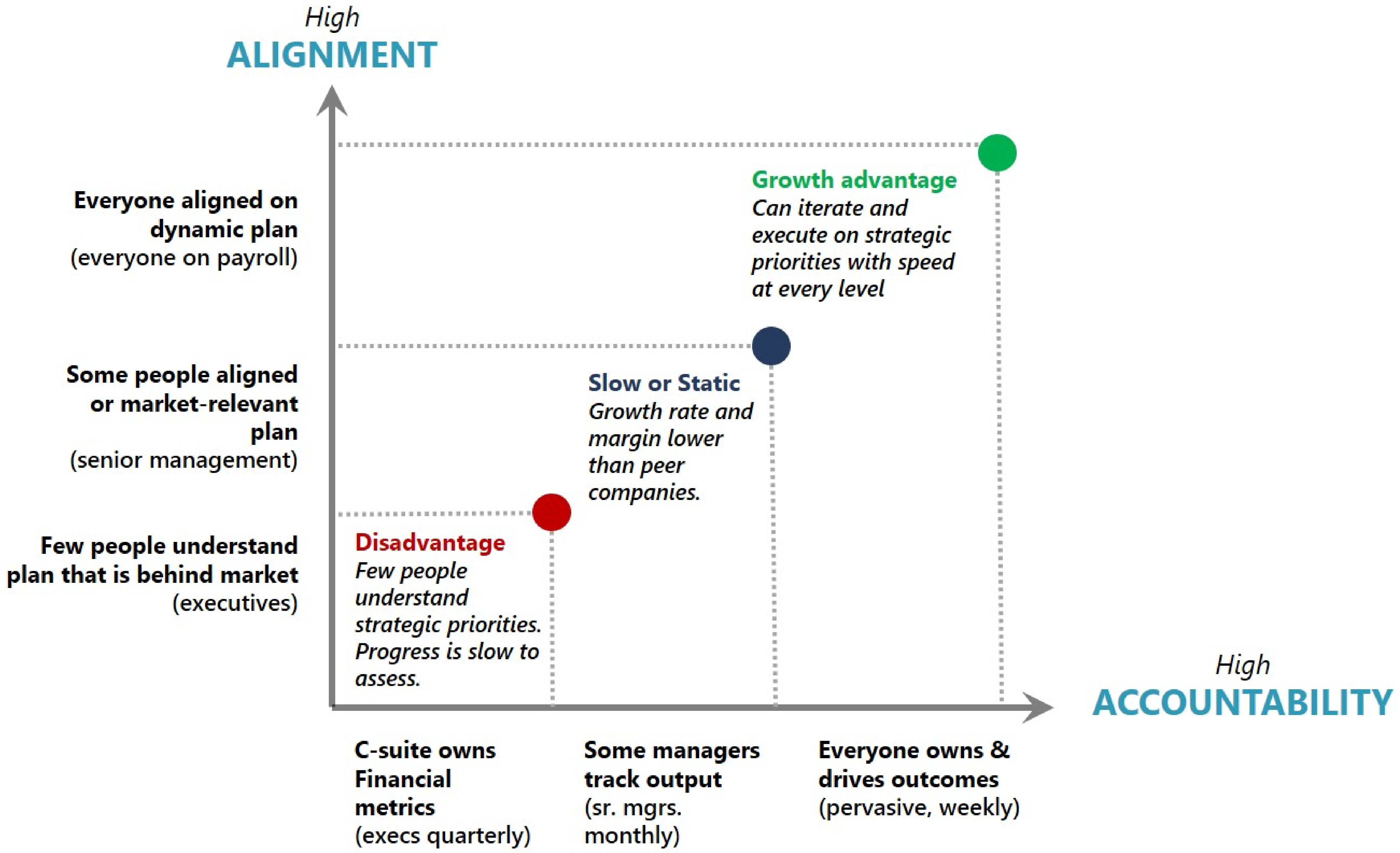
- Before: complete the code and then fix the bugs.
  - Low morale from the team: they were only working in bug fixes!
- Bug-cap `# of engineers * 5 = bug cap`
- If a team goes over their bug cap, then they stop new features until they fix their bugs.

Shield from distractions

- On every sprint, each team has two crews: Feature and Customer.
- Feature crew creates new features.
- Customer crew fixes live site issues and interruptions.

Metrics (OKR Framework)

- **Objectives:** Big ideas, not actual numbers.
- **Key Results"** Measure the progress towards the objectives.
    - Change in monthly growth rate of adoption
    - Change in performance
    - Change in time to learn
    - Change in frequency of incidents


High
ALIGNMENT
Everyone aligned on dynamic plan
(everyone on payroll)
Some people aligned or market-relevant plan
(senior management)
Few people understand plan that is behind market
(executives)
Growth advantage
Can iterate and execute on strategic priorities with speed at every level
Slow or Static
Growth rate and margin lower than peer companies.
Disadvantage
Few people understand strategic priorities. Progress is slow to assess.
High
ACCOUNTABILITY
C-suite owns Financial metrics
(execs quarterly)
Some managers track output
(sr. mgrs. monthly)
Everyone owns & drives outcomes
(pervasive, weekly)

Metrics

Some metrics do not accrue value towards the objective, and may not be helpful.

- Accuracy of original estimates
- Completed hours
- Lines of code
- Team capacity
- Team burndown
- Team velocity
- Number of bugs found
- Code coverage

OKR Framework Team Benefits

- Every team is aligned on the plan.
- Teams focused on achieving outcomes rather than completing activities.
- Every team is accountable for efforts on a regular basis.

OKR Example

- **Objective:** Be the top US provider of learning platforms to schools.
- **Key results:**
  - 45 percent of K-12 schools using our platform.
  - A 12 percent increase in student engagement, as measured through internal systems.
  - A 95 percent satisfaction rate from quarterly parent surveys.

Key takeaways from Microsoft's experience

1. Take agile seriously, but don't be overly prescriptive. Agile can become too strict, let it grow like a mindset or culture.
2. Stop celebrating activity and start celebrating results. Lines of code shouldn't outweigh functionality deployments.
3. You can't cheat shipping. Until you get into the mindset of updating services/products at the end of sprints, you won't find all of the work that needs to be done. Shipping every Sprint helps establish a rhythm and cadence.
4. Build the culture you want and you'll get the behavior you're looking for.

References

- Barpilot DevOps Culture
- How Microsoft Plans DevOps
- What is Agile?