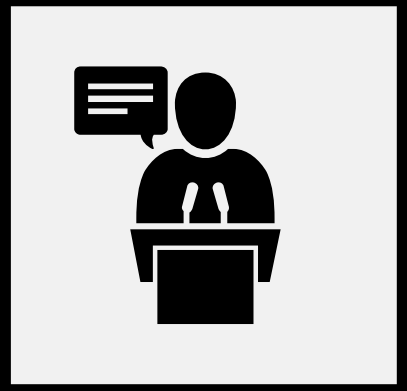
GitOps auf echtem Blech
oder: moderne Verwaltung von Desktop Systemen

Verwaltung von Ubuntu Desktops für Softwareentwickler mit GitOps, IaC und CI/CD

Patrick Banholzer, ITT/ME, Open Source Automation Days, 2021/10/05

Mercedes-Benz
The best or nothing.

Personal information



Patrick Banholzer

Current Position:

Mercedes-Benz AG
System Architect / DevOps Engineer
Since 2019
Responsible for ~2500 Ubuntu desktops at Mercedes-Benz R&D

Experience:

Deep knowledge in everything that is IT infrastructure
15+ years of professional Linux experience
Motto: automate everything!

Agenda

Challenges in a big (🪟) corporate network

Requirements of different dev teams

Journey from Old-IT to New-IT

- What we did it in the past
- How we evolved by using DevOps and FOSS

Where are we today?

Why Saltstack? What about Puppet / Ansible ...?

Whats Next?

Challenges:
in a (big) Windows Network

Workplace

Email, Calendar, Collaboration, MS Office

Authentication

Kerberos, LDAP, SSSD,
MS ActiveDirectory

Corporate Network

Proxy, Firewalls, DNS, VPN, ...

Processes

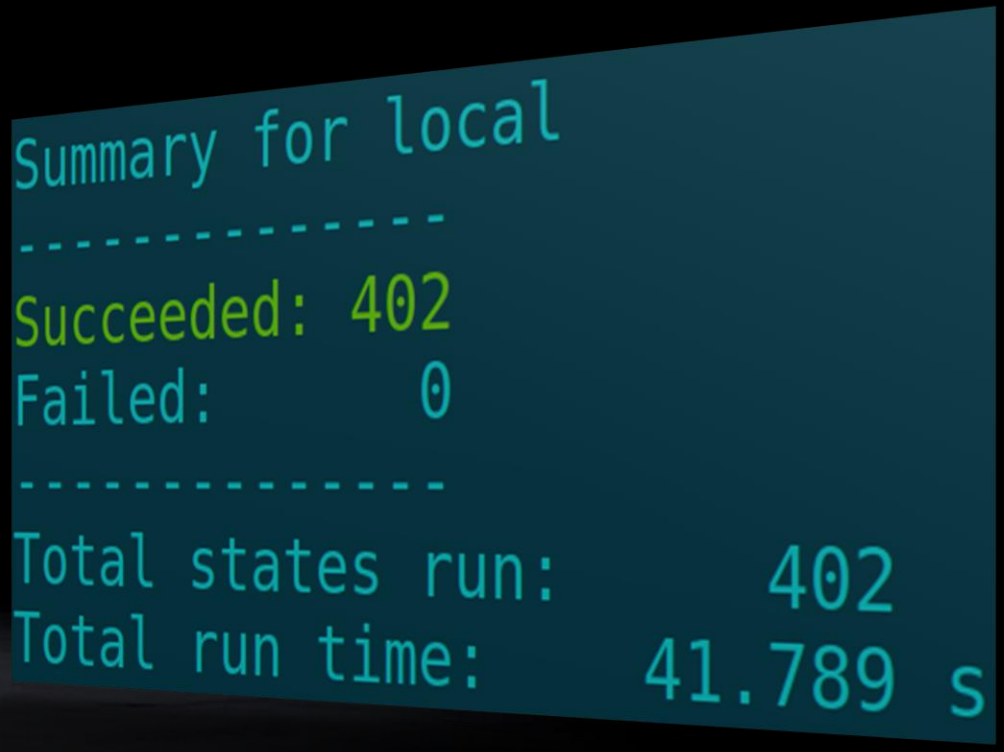
Asset Accounting, Asset Management

Security

x509, TPM2, LUKS, Permissions

Challenges:
we definitely need config management

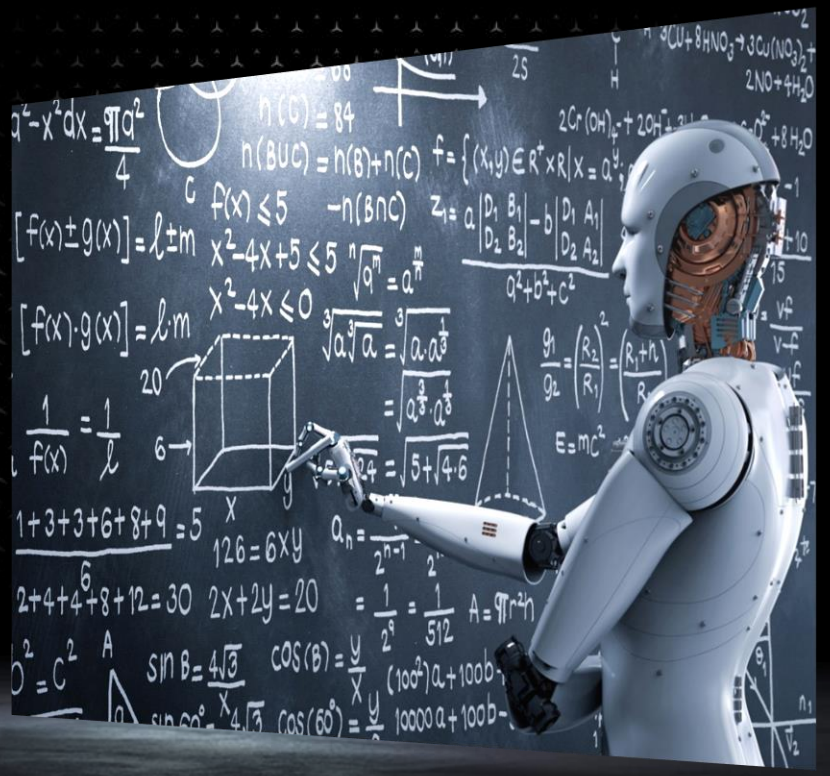
As even systems that are not customized
for special project setups apply
more than 400 salt-states.



Requirements and wishes of sw developers

- Everyone wants sudo (they really need it in the most cases)
  - keep users from fighting the config-management
  - less experienced users tend to break their systems
  - → quite hard job for operations team to keep up with

- Window-Managers (KDE / Gnome / XFCE / i3 / awesome?)

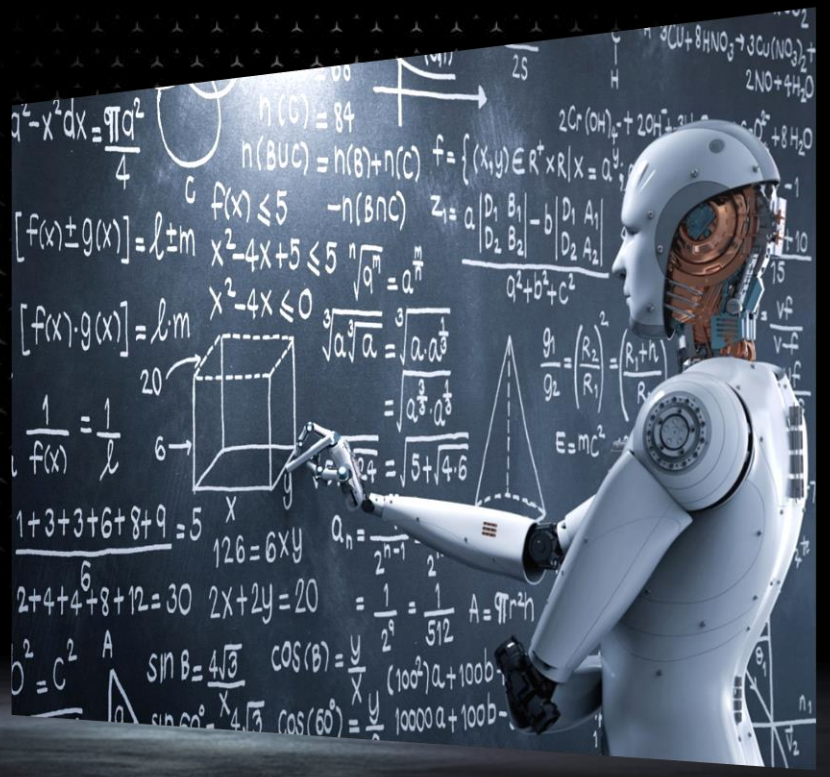


Requirements and wishes of sw developers

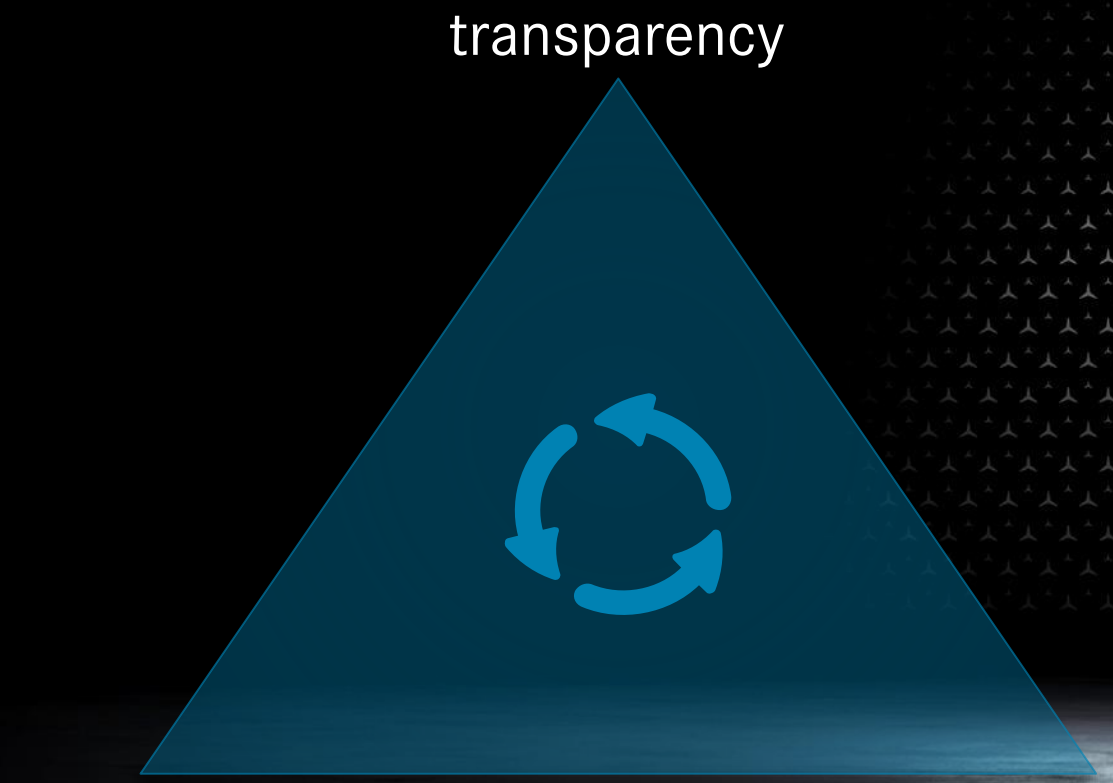
- Nvidia Drivers and CUDA – each project wants its own setup

- Local vs. remote / VDI
  - Access huge data needs system directly connected to clusters
  - VDI is not that comfortable (blurry, laggy, …)

- Custom “in Car” systems for autonomous driving projects
  - stability / traceability / conformity / security
  - More than five different high performance systems in a trunk



Three factors for successfull config-management



The Journey

OLD IT

Intransparent
Closed
Static
Slow

NEW IT

Transparent
Open
Flexible
Fast

Old World of OS Configuration Management

Monolithic and rigid approach

No Sharing

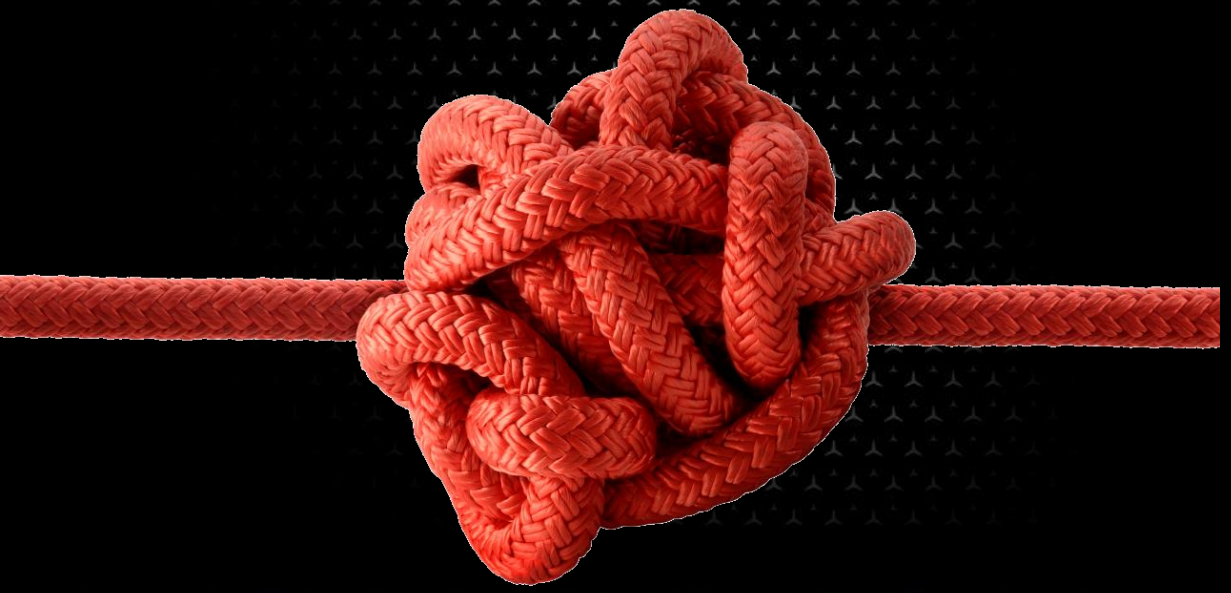
No (granular) Access Control

No Workflow

Vendor Lock

No Open Source

Problems with Old World Approach



- No tenants
- No API, no interfaces
- No automatic testing
- No quality checks / any admin can do anything
- No sharing of code & configs
- No continuous integration of multiple contributors
- No client introspection / no state definition

Only a bunch of scripts that do things.

Riscs and Weakness we had

POOR WORKING MODEL

- Ticket → manual execution
- Low level of automation
- New code brings back previously solved bugs
- Missing workflow enforcement
- Missing enforcement of knowledge sharing

RESULTED IN

- Nobody knows what the others do
- Quality varies with admin experience and knowledge exchange
- No audit trail
- No status information of clients
- No history about clients

The DevOps Approach:
Adapt Best Practices from agile Software Development

IaC / GitOps

everything lives in git

Tested and approved

by process design

Automation and Integration

via APIs and open interfaces

Continuous Integration and Delivery

commit & test & deliver fast and reliable

Gains of Infastructure as Code

Open, cooperative and stateful

Co-operation: Share your code!

Workflow: Know what happens!

Time to Delivery: Be fast!

Quality: Apply Gates!

Lifecycle of a device

- Order and Deliver with internal shopsystem
- Provision and set parameters with Foreman
- Configure and control with Saltstack
- Operate and deploy with code and git
- Updated via Ubuntu repository snapshots *(might be Katello & Pulp in future)*

Setup of the Infrastructure

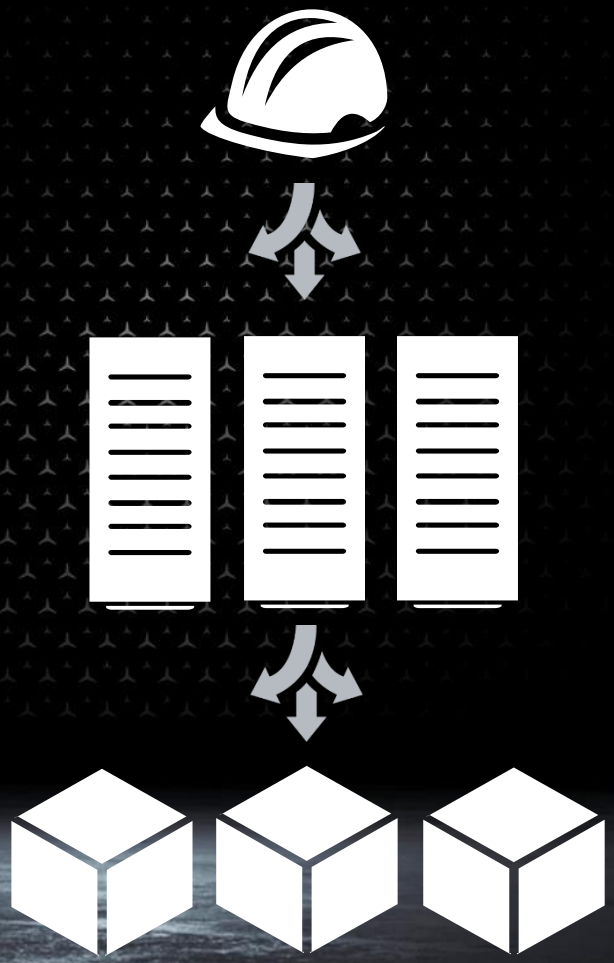
One Foreman that is connected to many Salt-masters running in containers on multiple servers

→ One API to talk to (Foreman)

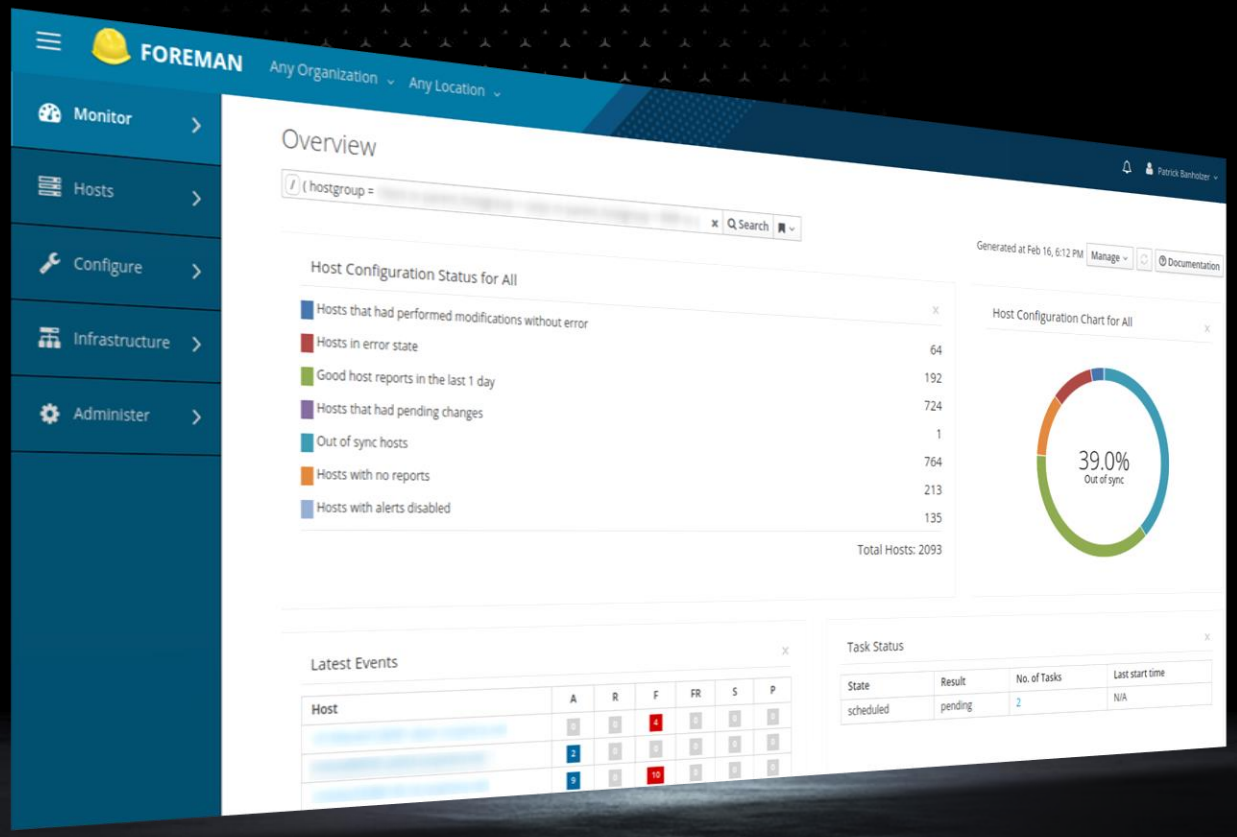
→ Servers + Containers also managed through Salt + Foreman

→ One Git repo per environment
  → Backend Systems
  → Clients
  → Cars
  → IoT / Edge Devices

→ One (or Multiple) container per environment



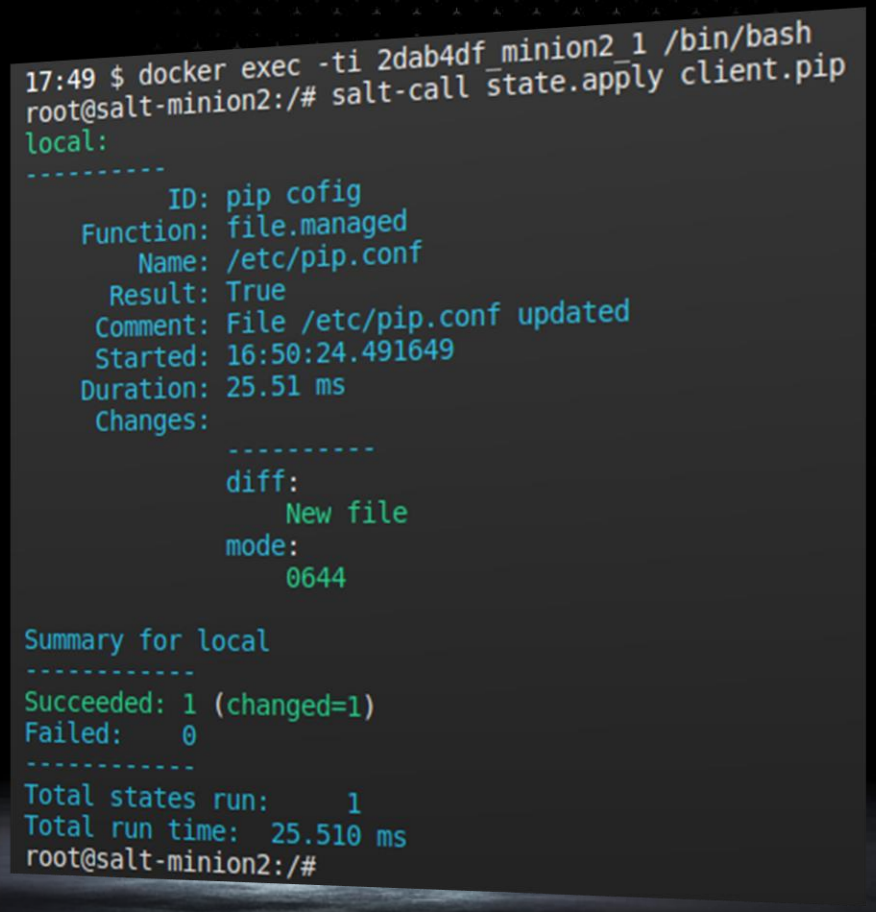
The Foreman



- Technical CI management
- Bare Metal Deployment
- Parameters, Database, API
- Reporting and Auditing

Saltstack

- Configuration management
- Has parameters from foreman
- Configured in GIT
  - repeatable, stable, transparent, traceable
- Defines a target state and applies it!



Comparison of Saltstack to competitors

	Connection Type	Software needed on clients	Ready to use upstream Solutions	Implemented in	Interaction with Foreman	Used at Daimler / Mercedes
Saltstack	Minion → Master – Message Queue	Salt-Minion	+ (Formulas)	Python	+ (plugin)	YES (default with SLES 15)
Ansible	Master → Target – SSH	Python	+ (Galaxy)	Python	+ (plugin)	YES
Puppet	Agent → Master – Request	Puppet Agent	++ (PuppetForge)	Ruby	++ (native)	YES

Workflow

Jira Task

Sprint planning / Emergency Change

Git Feature Branch

Implementation

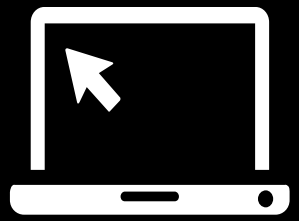
Local Tests

Commit

Git Push

refinement

Issue
Bug Report
Change Request



Automatic deployment

Merge into Master

Successfull test is mandatory

Test Pipeline

Salt Lint

Saltcheck (Unit Tests)

Run each state in docker environment

Pipeline Test

Successful test + 2 approvals

Pull Re-quest

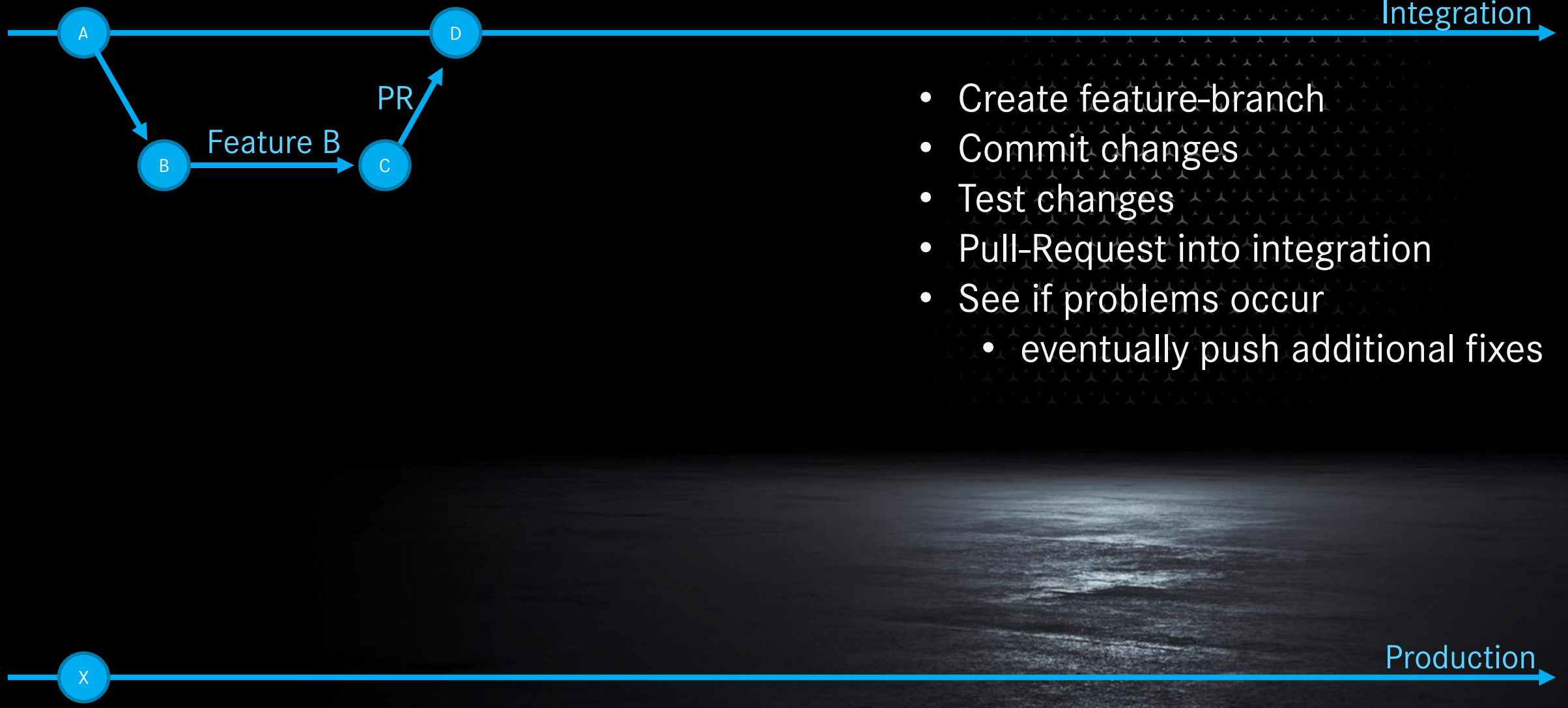
Manual tests on int. environment (including friendly-users)

Merge into int

Test run

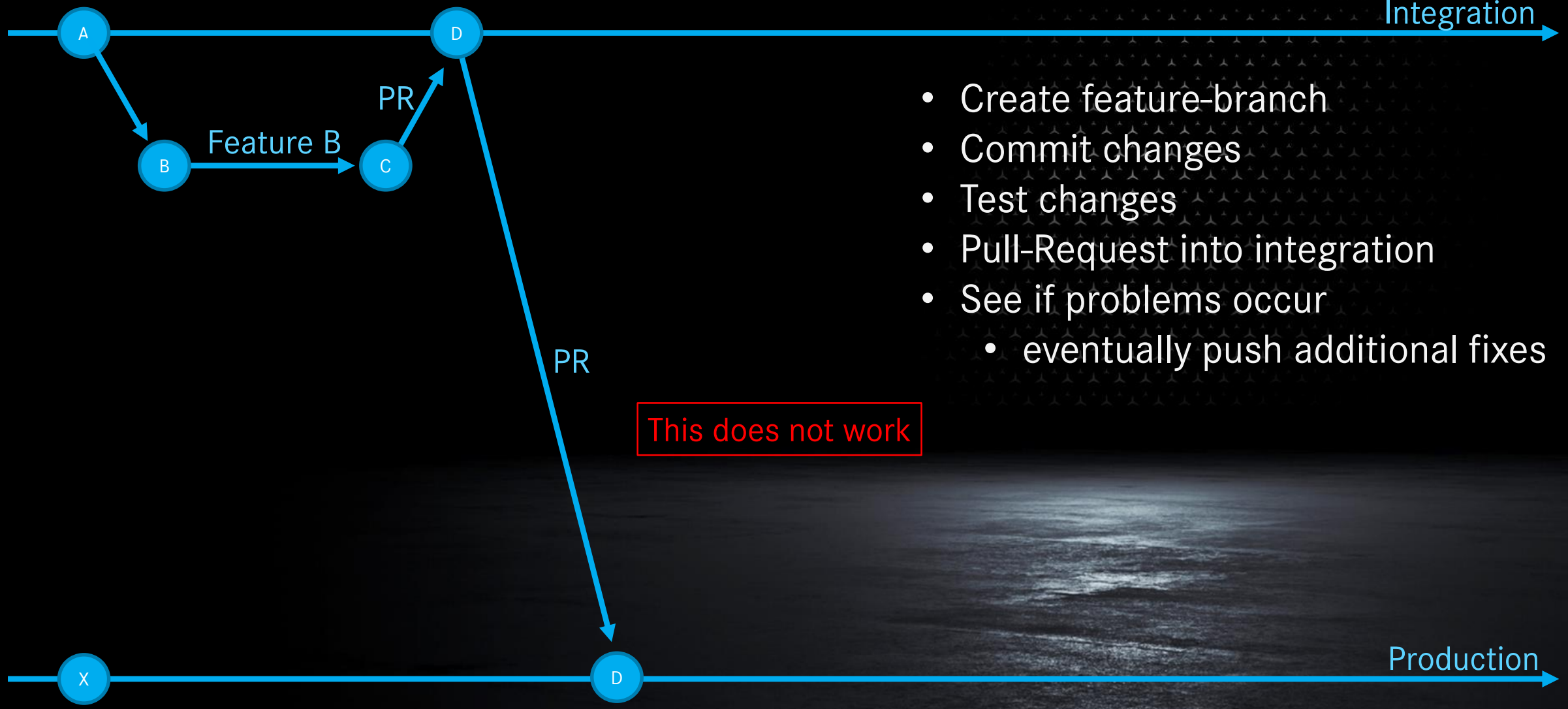
Pull Re-quest

Git Workflow



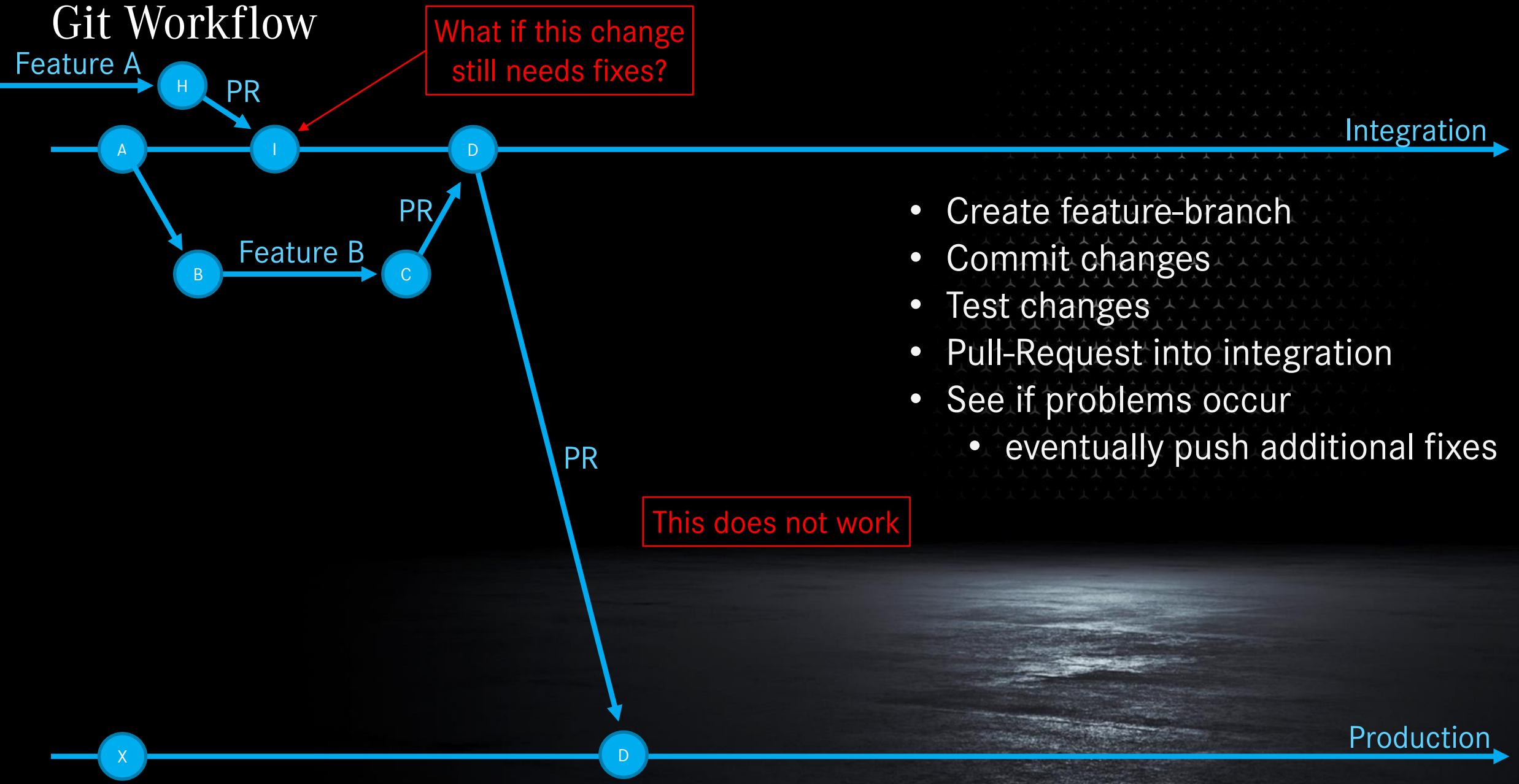
- Create feature-branch
- Commit changes
- Test changes
- Pull-Request into integration
- See if problems occur
  - eventually push additional fixes

Git Workflow



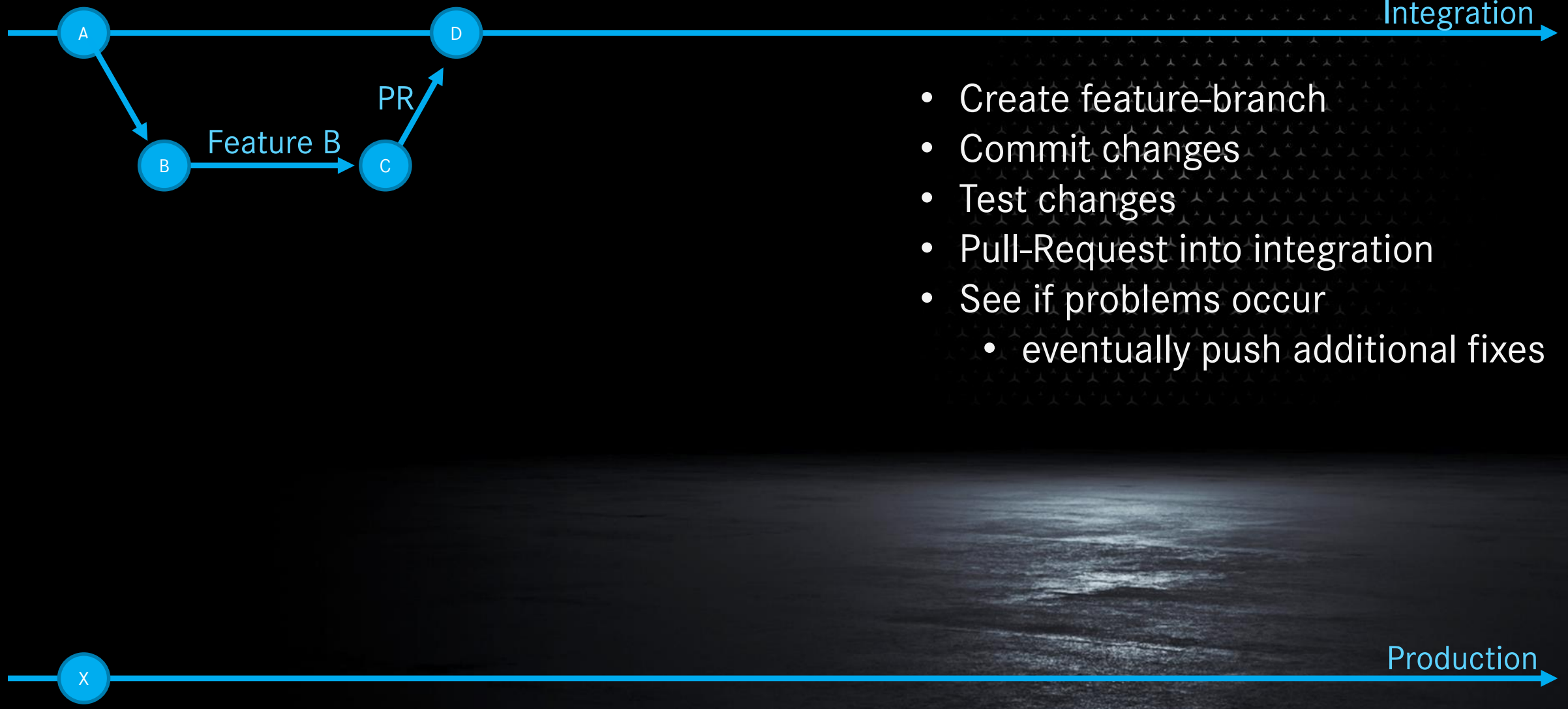
- Create feature-branch
- Commit changes
- Test changes
- Pull-Request into integration
- See if problems occur
  - eventually push additional fixes

Git Workflow



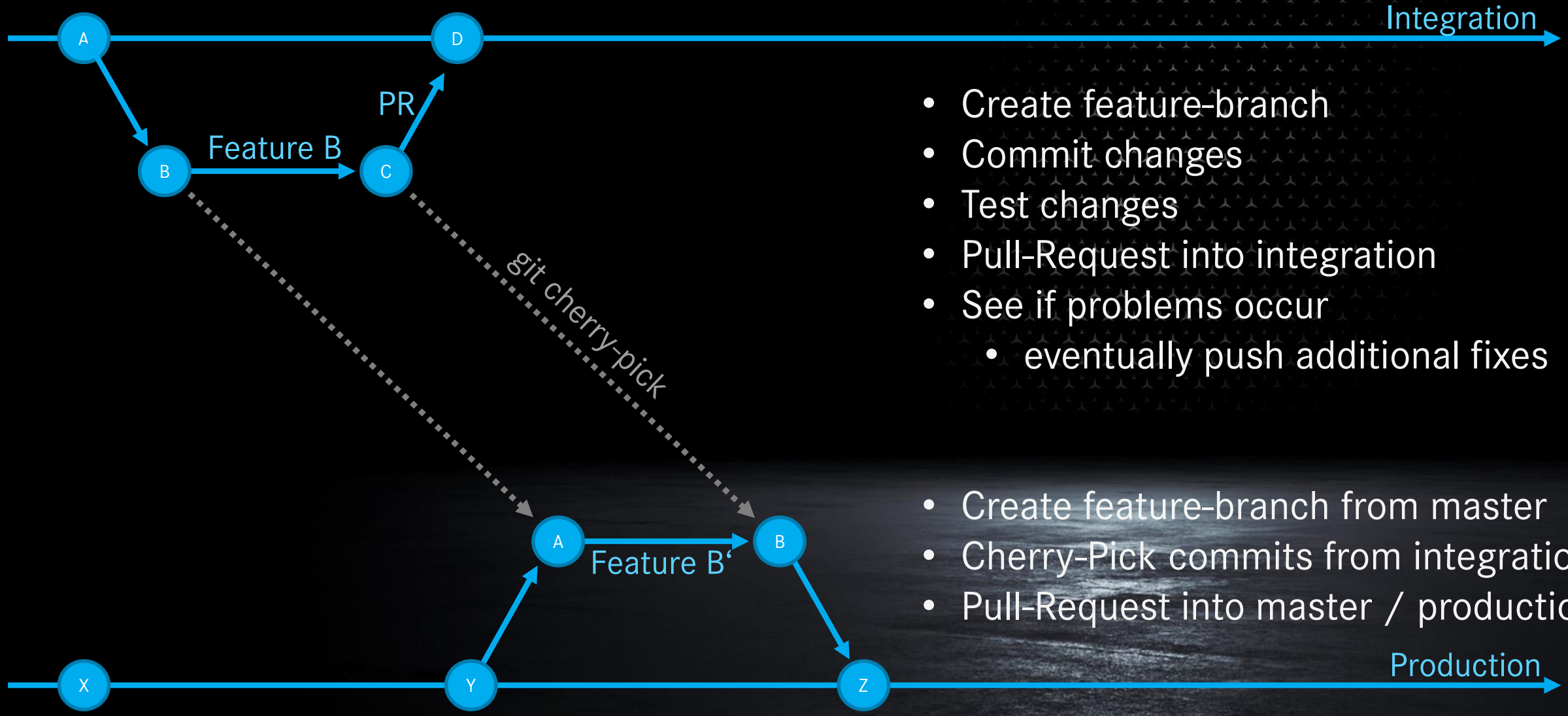
- Create feature-branch
- Commit changes
- Test changes
- Pull-Request into integration
- See if problems occur
  - eventually push additional fixes

Git Workflow



- Create feature-branch
- Commit changes
- Test changes
- Pull-Request into integration
- See if problems occur
  - eventually push additional fixes

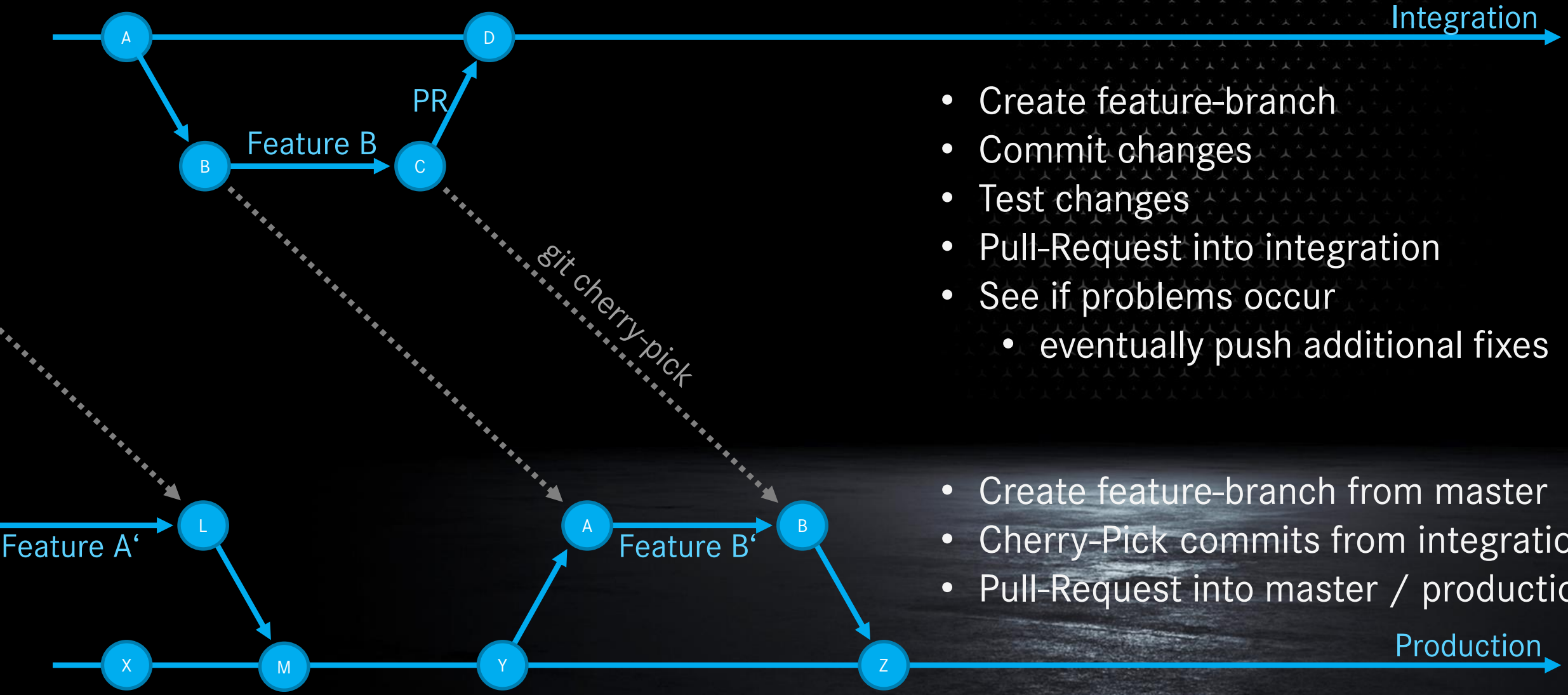
Git Workflow



- Create feature-branch
- Commit changes
- Test changes
- Pull-Request into integration
- See if problems occur
  - eventually push additional fixes

- Create feature-branch from master
- Cherry-Pick commits from integration
- Pull-Request into master / production

Git Workflow



- Create feature-branch
- Commit changes
- Test changes
- Pull-Request into integration
- See if problems occur
  - eventually push additional fixes

- Create feature-branch from master
- Cherry-Pick commits from integration
- Pull-Request into master / production

Benefits and problems with this workflow

PROS

Deploy when ready

Parallel work

No blocking changes

Easy revert possible

CONS

Commits can be missed

Gap between int and prod

PRs to master must be carefully planned

Test pipeline enables local testing

Docker / Docker-Compose help with setup

- Setup & run test-environment fast
- Tear it down even faster!
- Run again...
- Each admin pc can run tests
- Local tests compare to pipeline

Whats next?

DEVOPS CONFIG MANAGEMENT

Improve test coverage

Tests in virtual environment

Black-box tests on hardware

Tests of fresh installations
At least once a day
On each supported hardware

Improve Update-Management

DESKTOP FEATURES

Ship to Desk installations

Zero Trust Client setup

Support of E2E encrypted emails

Self-Service platform

Get rid of proxies

Your Questions!?