
GLOBAL SCRUM GATHERING
London
8-10 OCTOBER 2018
ScrumAlliance
The Fantastic Four Continuous Delivery Coding patterns
Luca Minudel
HUBLOT
Delivering software
Faster & Safer at the same time
@LukaDotNet

SMHARTER

LUCA MINUDEL

LUCA.MINUDEL@SMHARTER.COM

HTTPS://WWW.LINKEDIN.COM/IN/LUCAMINUDEL/

@LUKADOTNET

SCUDERIA FERRARI

ThoughtWorks®

HSBC

LexisNexis®

LLOYDS BANKING GROUP

- high pressure, high speed
- constant change
- highly regulated environment
- mission/life critical high availability software
- unmovable deadlines
- inescapable reality checks every race weekend
- bugs broadcasted live worldwide to a large global audience

The Challenge: Faster and safer at the same time ?!?

Our constraints turned into learning

- Inescapable F1 calendar teach you time is finite and limited
  Some features development may span 2 races
  Lesson => Maximise learning at every deploy for every feature

Our constraints turned into learning

- Inescapable F1 calendar teach you time is finite and limited
  Some features development may span 2 races
  Lesson => Maximise learning at every deploy for every feature

- Automatic remediation plans (e.g. rollback) are invaluable
  You cannot rollback after a backward incompatible change
  Lesson => Avoid irreversible changes to allow 1-click rollbacks

Our crisis turned into learning

- Version Control System bug preventing branching

Lesson => you can develop SW faster and enjoy more flexibility without branching

Lesson => extensive continuous design and refactoring is possible (no merge hell) without branching

Github repo link: https://goo.gl/N6cKn8

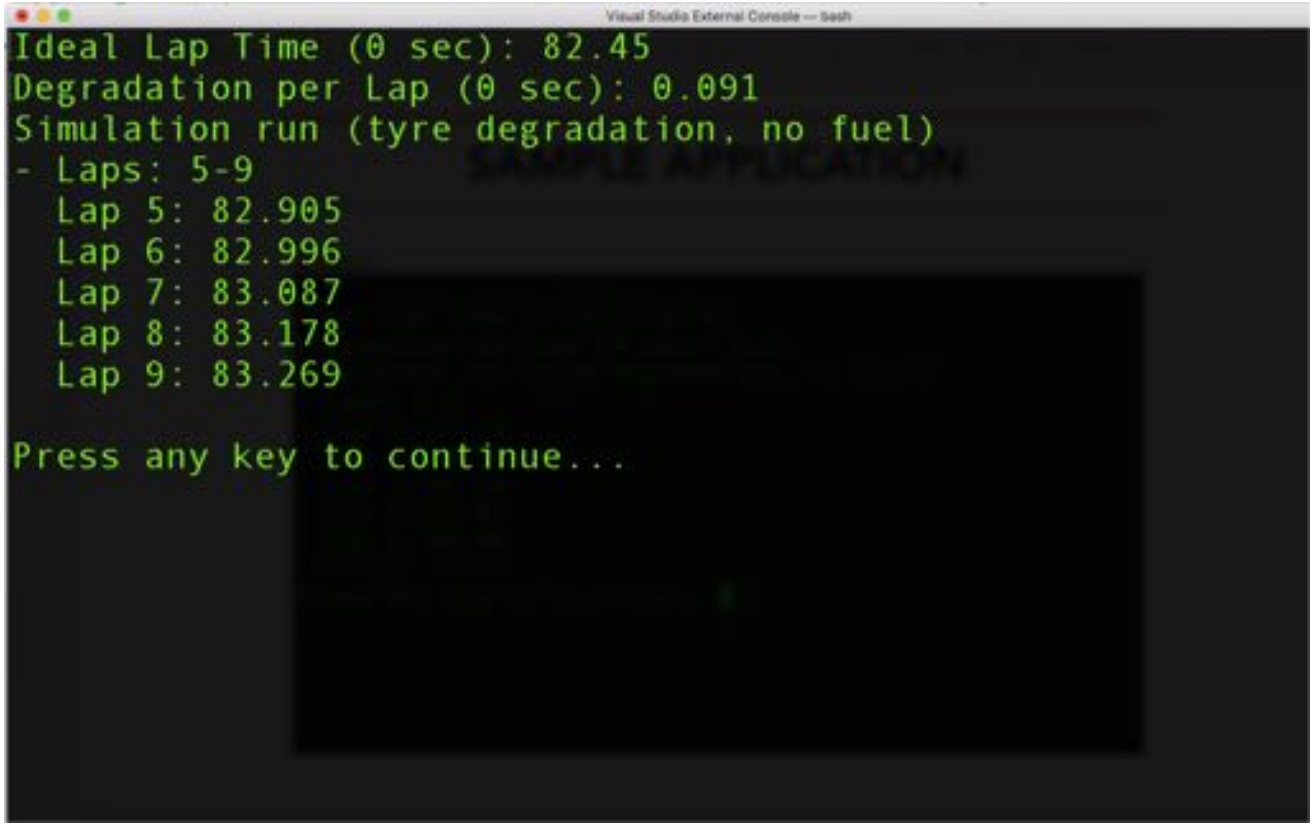
Four pieces of the solution

Lap time simulation & the initial code version 1

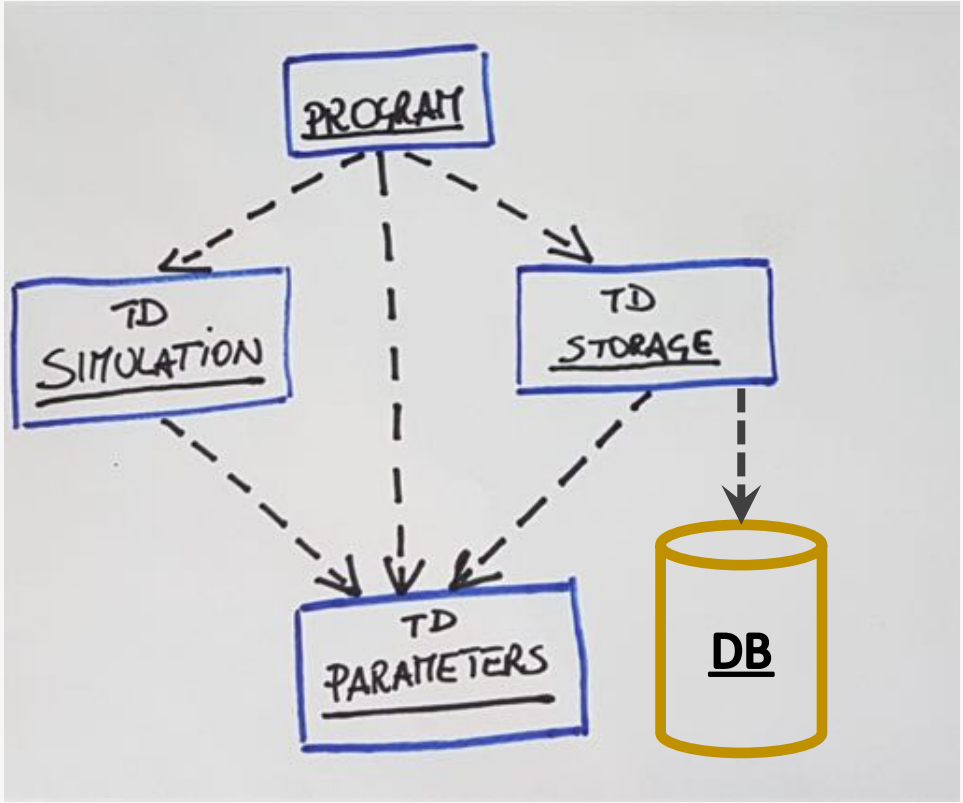
- Ideal lap time

- Tyre degradation

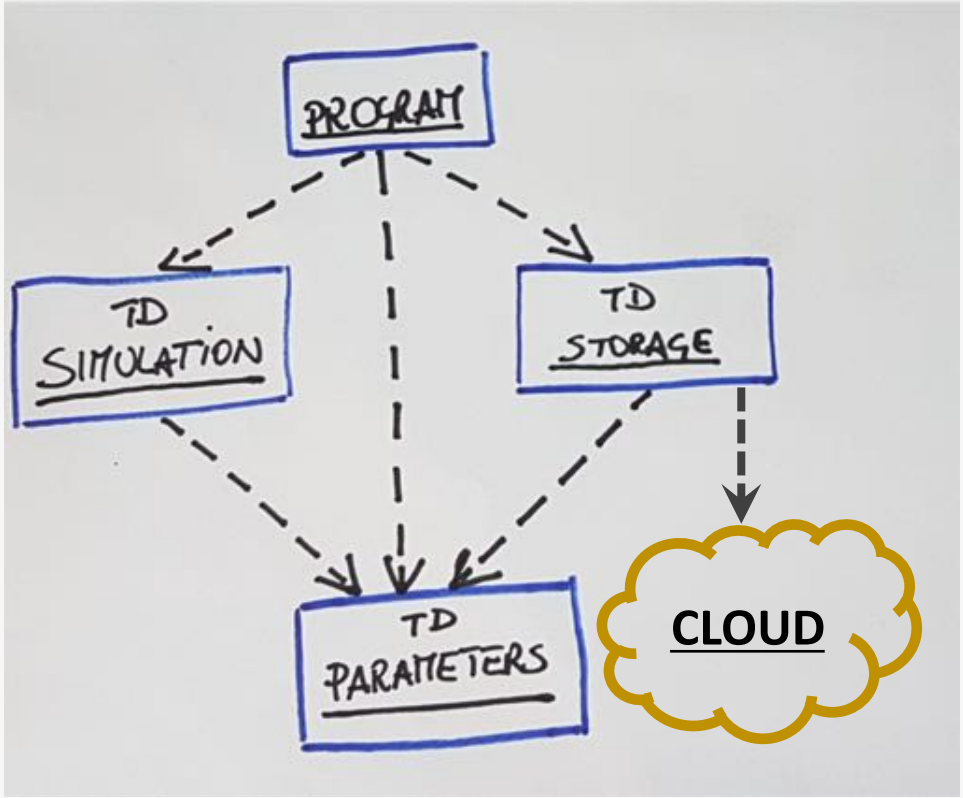
SAMPLE APPLICATION



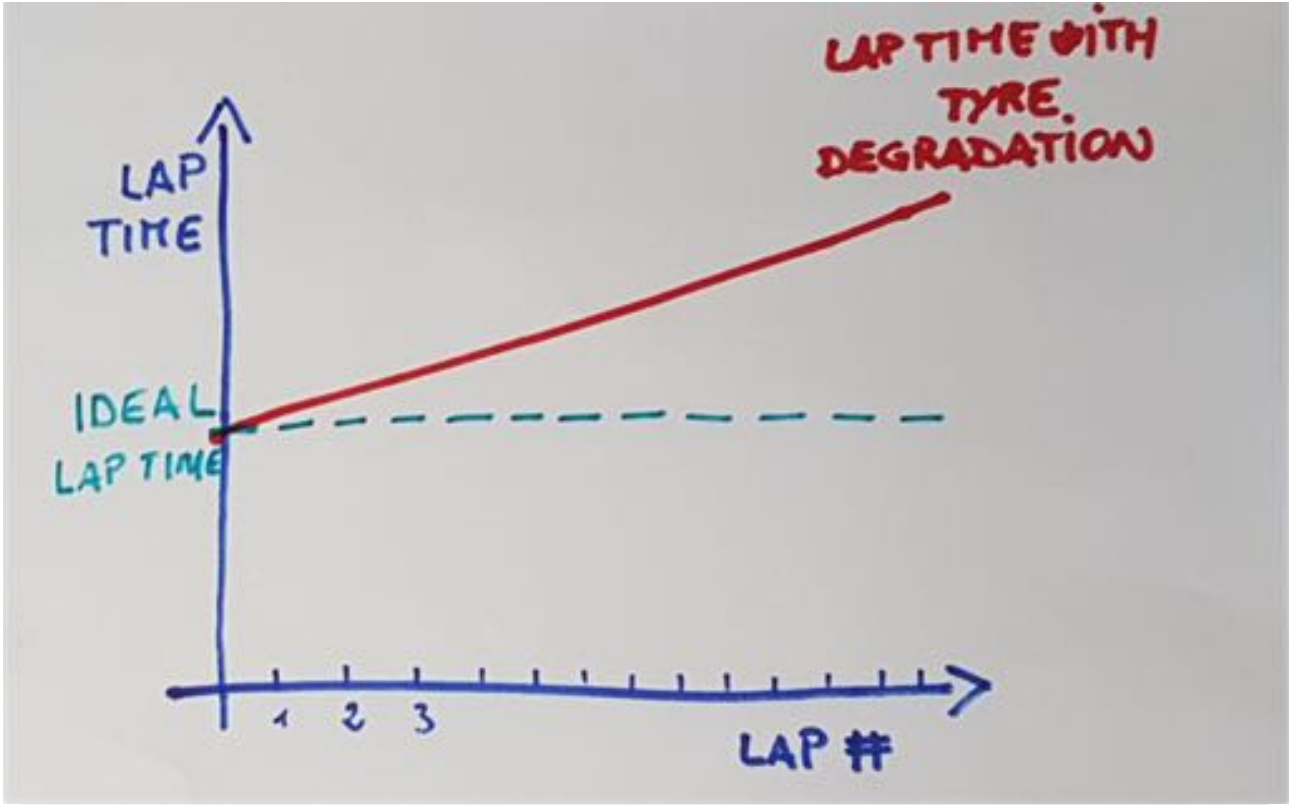
Lap time simulation & the initial code version 1



Lap time simulation & the initial code version 1



LAP TIME SIMULATION



Lap time simulation & the initial code version 1

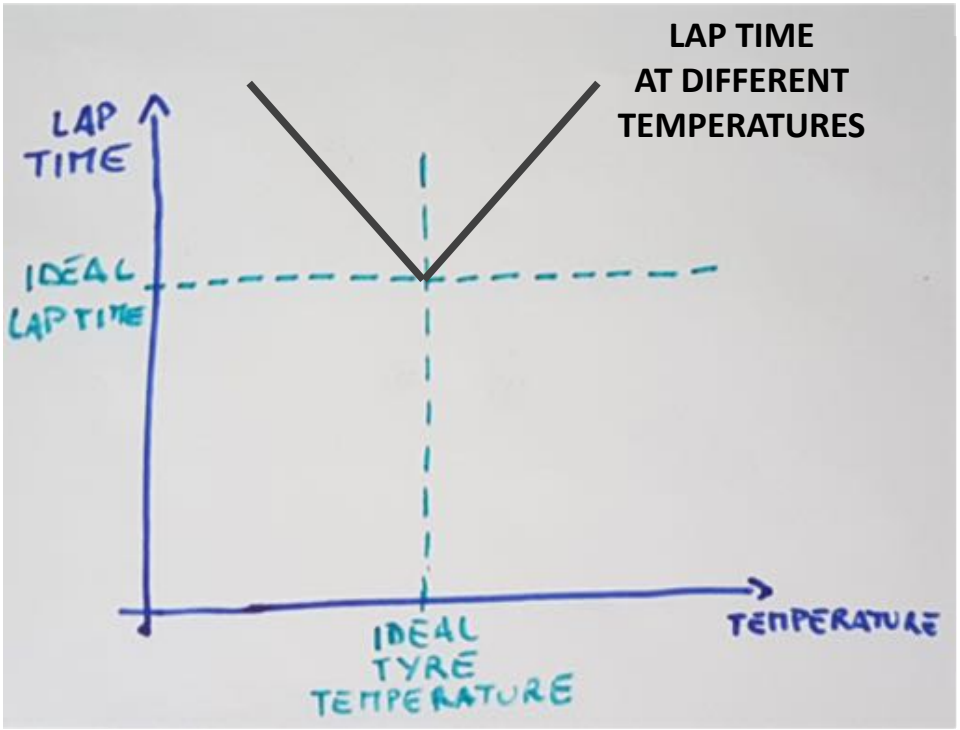
CODE VERSION	BACKWARD COMPATIBLE	ROLLBACK VERSION	DB VERSION
V1			DB-V10

Code version 2a

- Starting to introduce in the simulation the impact on Lap time of the Ideal tyre operating temperature

Remember: no VCS branches !

TYRE OPERATING TEMPERATURE



Code version 2a

⇒ PATTERN 1: Trunk based development

⇒ PATTERN 2: Latent-code

⇒ Branch by abstraction (instead of branching)

CODE VERSION	BACKWARD COMPATIBLE	ROLLBACK VERSION	DB VERSION
V1			DB-V10
V2A	**Y**	**V1**	**DB-V10**

Trunk-based development:
A branching model where developers collaborate on code in a single branch called trunk/master, don't create other long-lived development branches by employing documented techniques. They therefore avoid merge hell while practicing continuous design & refactoring, do not break the build, and live happily ever after.

- TrunkBasedDevelopment.com

Latent-Code:
Code that is committed to the trunk/master of the source-code repository, it is tested by automatic tests,
it is included in the binaries released into production,
while it remains otherwise unused and inaccessible to the users.

- Luca Minudel

Branch by Abstraction:

- *See www.BranchByAbstraction.com*

Race weekend: code version 2b, feature unfinished

- Start learning from production
  without releasing the unfinished feature

  Making the most of the limited
  number of test and race events

Race weekend: code version 2b, feature unfinished

⇒ PATTERN 1: Trunk based development

⇒ PATTERN 2: Latent-to-live-code

CODE VERSION	BACKWARD COMPATIBLE	ROLLBACK VERSION	DB VERSION
V1			DB-V10
V2A	Y	V1	DB-V10
V2B	**Y**	**V2A**	**DB-V10**

Latent-To-Live-Code:
Latent code gradually transitioned into live code that is partially exercised and used by existing features running in production, and by their automated tests.

It provides rapid and reliable feedback on unfinished features still under development, while keeping the software and the code-base always in a potentially releasable state.

- Luca Minudel

Code version 2c, hiding unfinished UI and DB changes

- Adding the new temperature parameter
  - to the UI
  - to the DB/STORAGE (this breaks backward compatibility)

Remember, no VCS branches,
and backward compatibility
required to allow
1-click rollbacks

Code version 2c, hiding unfinished UI and DB changes

⇒ PATTERN 3: Feature toggles (instead of cherry picking)

CODE VERSION	BACKWARD COMPATIBLE	ROLLBACK VERSION	DB VERSION
V1			DB-V10
V2A	Y	V1	DB-V10
V2B	Y	V2A	DB-V10
V2C	**Y (N)**	**V2B (NONE)**	**DB-V10 (V11)**

Feature toggles:
A technique that uses configurable toggles that allow to disable or enable an unfinished feature so that the feature can be integrated, tested, and be included in the production binaries before it is completed.

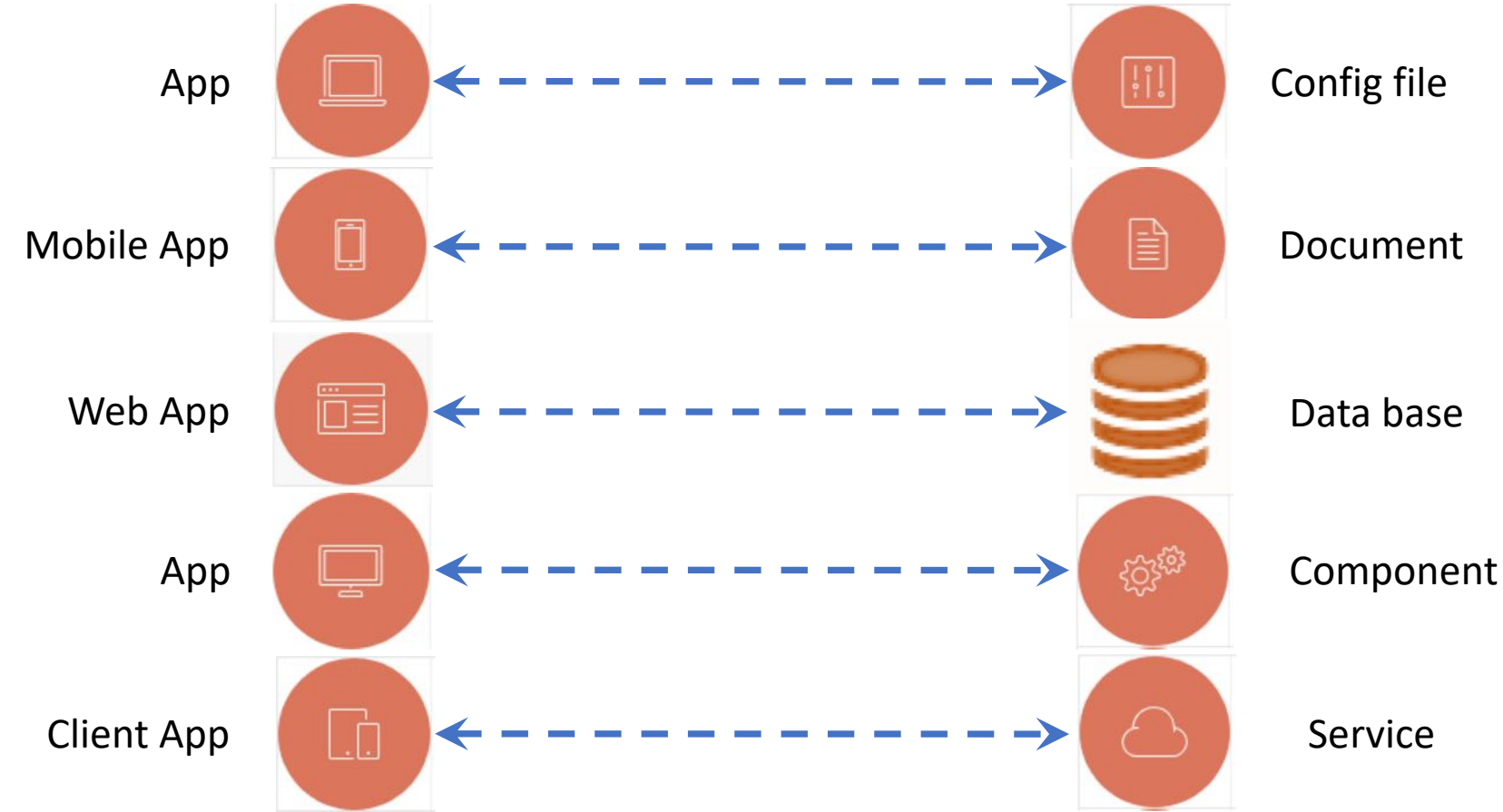
It enables rapid feedback about the feature under development, and it is an alternative to maintaining multiple source-code branches without incurring into the costs and risks of branching and merging.

Code version 3a, creating a bridge between versions

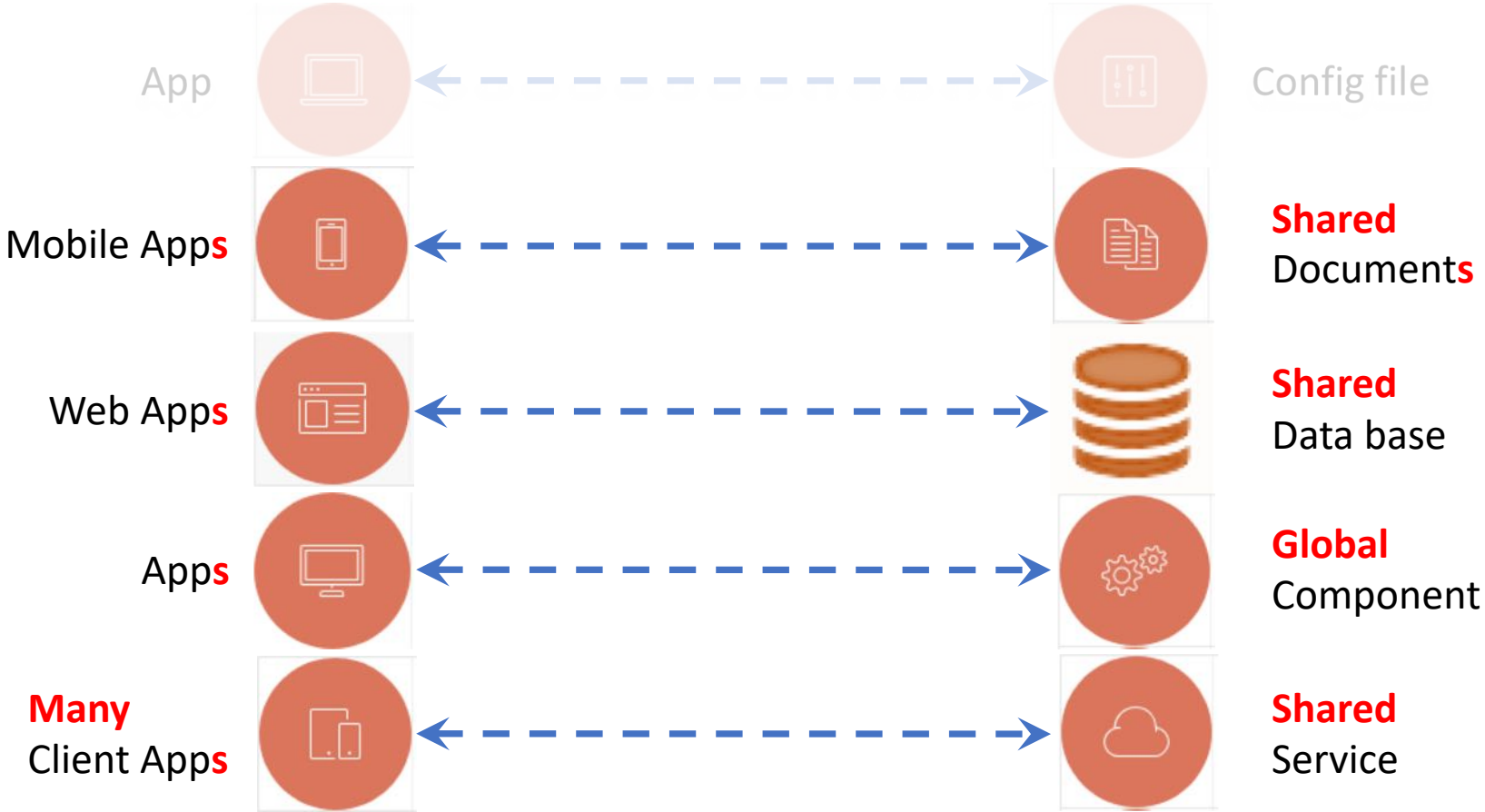
- Two-phase release of the new temperature parameter
  - in the UI
  - in the DB/STORAGE (this breaks backward compatibility)

Preserving the ability to rollback to the previous stable version in case of bugs

Backward incompatible changes require 2 moving parts

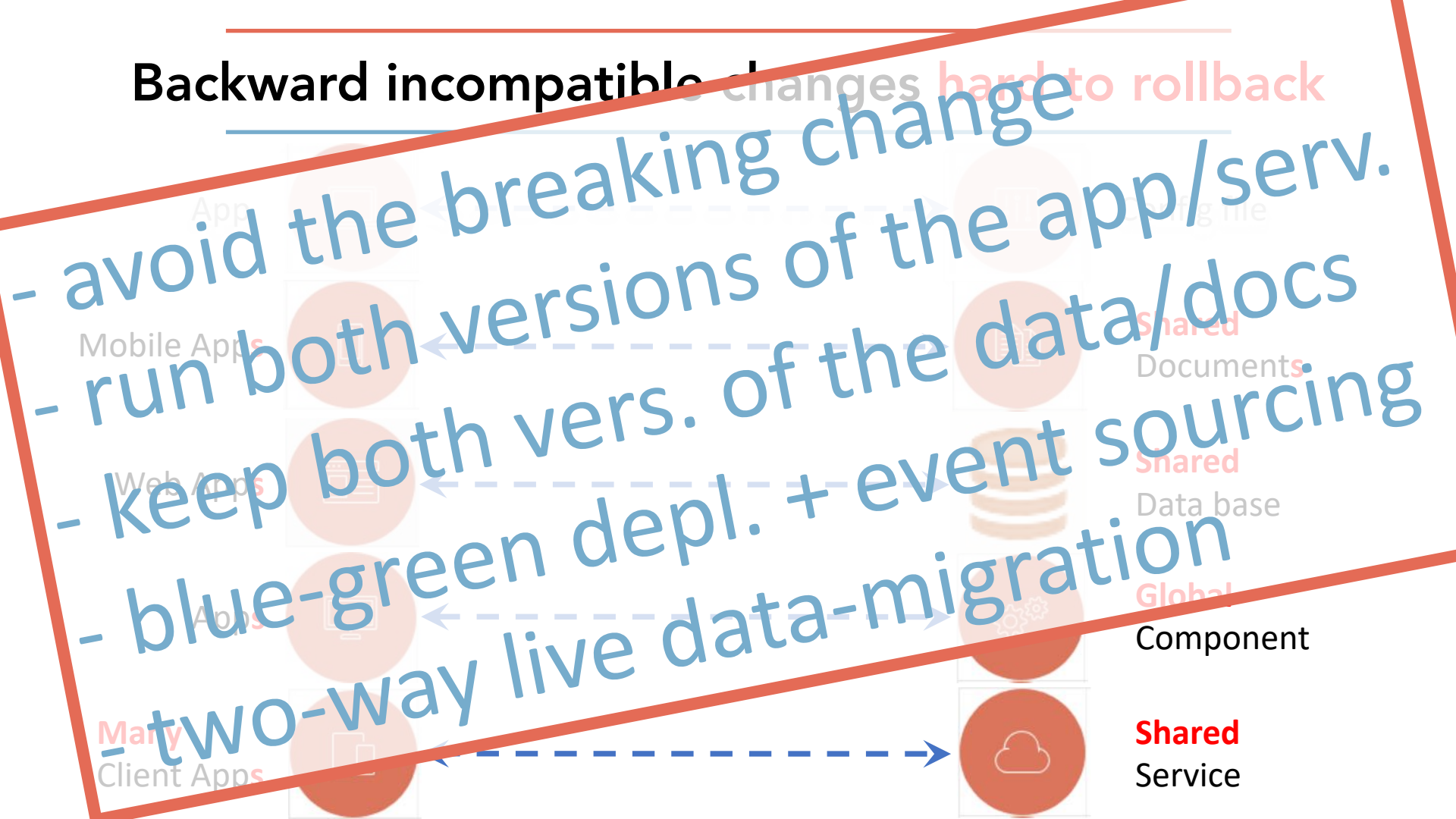


Backward incompatible changes hard to rollback



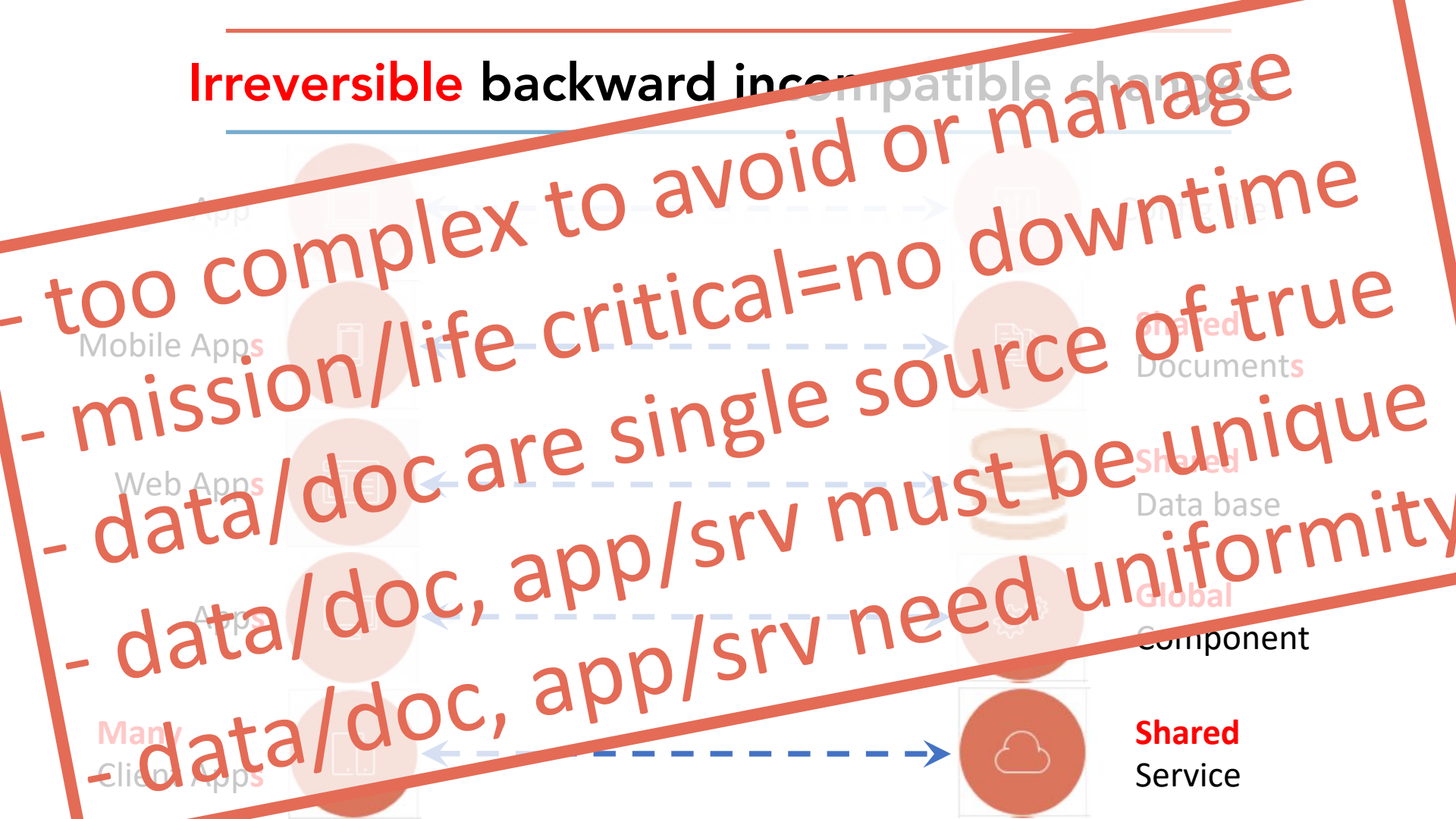
Backward incompatible changes hard to rollback

- avoid the breaking change
- run both versions of the app/serv.
- keep both vers. of the data/docs
- blue-green depl. + event sourcing
- two-way live data-migration



Irreversible backward incompatible changes

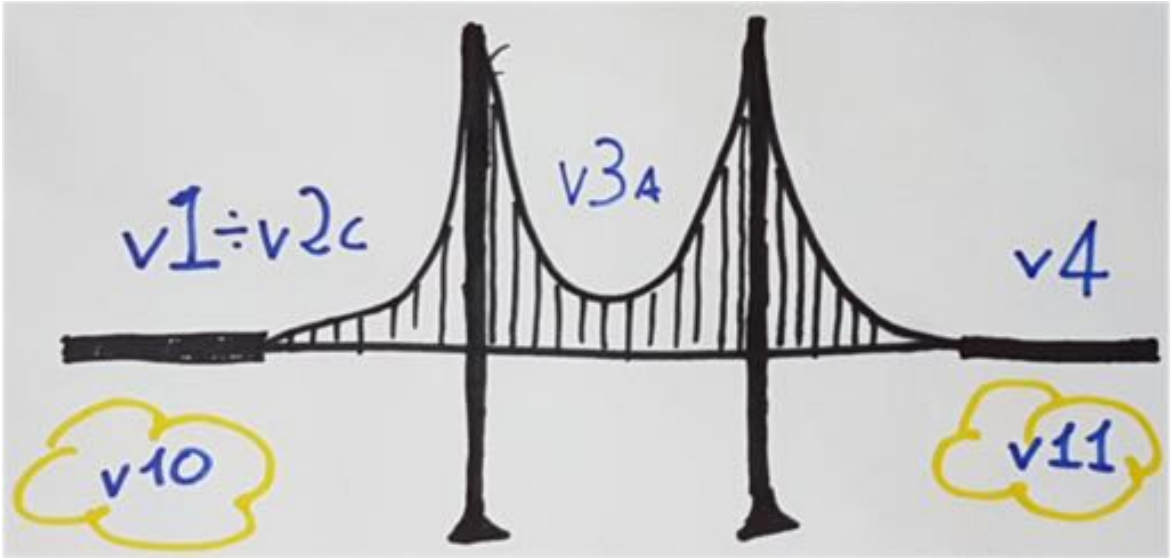
- too complex to avoid or manage
- mission/life critical=no downtime
- data/doc are single source of true
- data/doc, app/srv must be unique
- data/doc, app/srv need uniformity



FORWARD COMPATIBLE INTERIM VERSIONS AKA TWO-PHASE RELEASE

FORWARD COMPATIBLE INTERIM VERSIONS AKA TWO-PHASE RELEASE

FORWARD COMPATIBLE INTERIM VERSIONS AKA TWO-PHASE RELEASE



Code version 3a, creating a bridge between versions

⇒ Forward-compatible-interim-versions (AKA Two-phase release)

CODE VERSION	BACKWARD COMPATIBLE	ROLLBACK VERSION	DB VERSION
V1			DB-V10
V2A	Y	V1	DB-V10
V2B	Y	V2A	DB-V10
V2C	Y (N)	V2B (NONE)	DB-V10 (V11)
V3A	**Y**	**V2C**	**DB-V10 + V11**

Forward compatible interim version (AKA Two-phase release):
A special version of a software application that is backward compatible with the old version and at the same time forward compatible with the new version that breaks the backward compatibility.

This interim version is specifically created to allow an automatic remediation plan (such as a zero-downtime automated rollback) even in the case of a release that breaks backward compatibility.

- Luca Minudel

Forward compatible interim versions (AKA Two-phase release):
The first phase of the release includes the most complex changes that can potentially lead to a show-stopper bug, hard to detect. And it runs in production until there is sufficient confidence it does not introduce a show-stopper bug.

The second phase contains only the simplest part of the change that is extremely unlikely to introduce a show-stopper bug, that is also extremely unlikely to go undetected immediately after the release at the time when remediation is still easy, fast, and cheap.

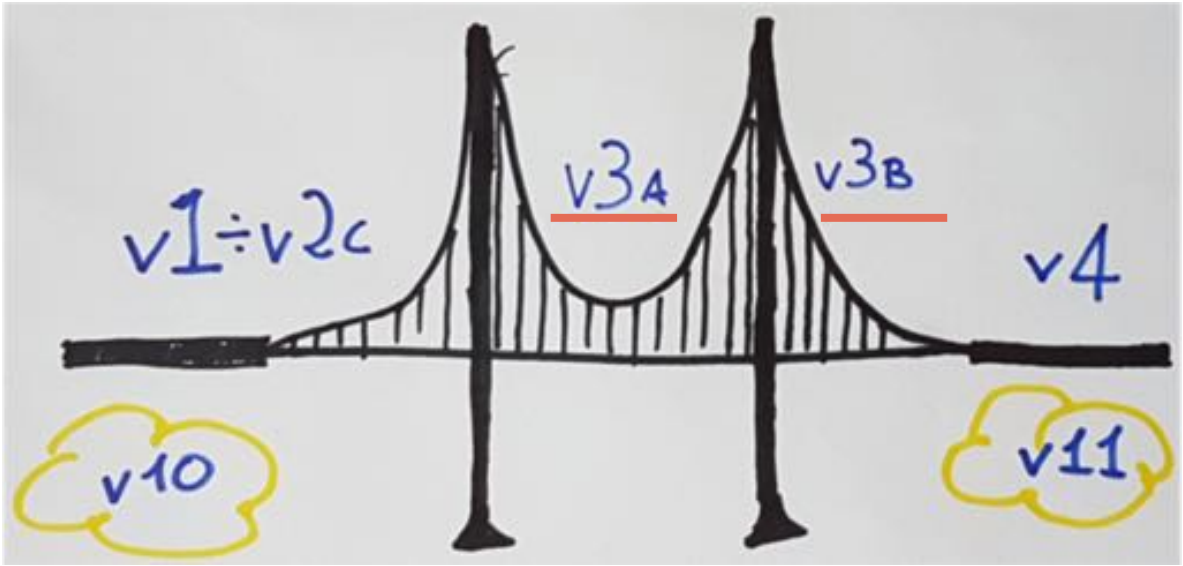
- Luca Minudel

Code version 3b, feature finished and backward comp

- Exactly the same version as 3a, just released with db v11.

Preserving the ability to rollback to the previous stable version in case of bugs

FORWARD COMPATIBLE INTERIM VERSIONS AKA TWO-PHASE RELEASE



TAMING IRREVERSIBILITY

FORWARD COMPATIBLE INTERIM VERSIONS (TWO-PHASE RELEASES)
Is a pattern for
taming the irreversibility
of backward compatibility breaking changes

Why reversibility is fundamental?

*Irreversibility is **one of the prime drivers of complexity** and so of costs & risks.*

Professor Enrico Zaninotto, 2002
Quoted by Kent Beck & Martin Fowler

TAMING THE IRREVERSIBILITY IN SOFTWARE DEVELOPMENT

FROM:

TO:

Code version 3b, feature finished and backward comp

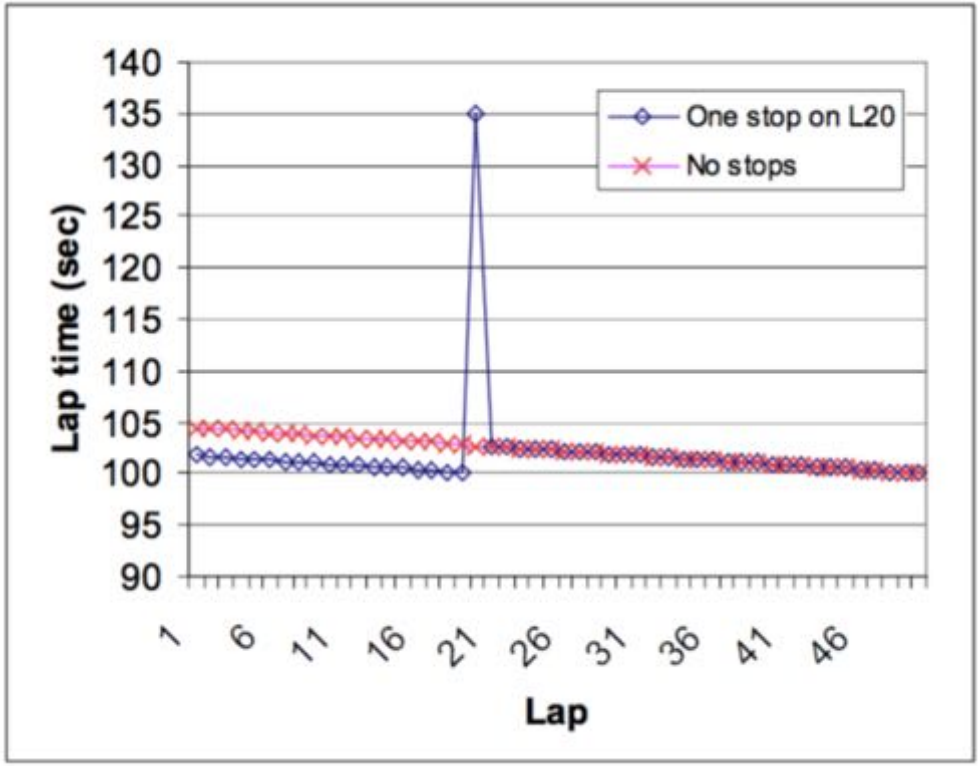
CODE VERSION	BACKWARD COMPATIBLE	ROLLBACK VERSION	DB VERSION
V1			DB-V10
V2A	Y	V1	DB-V10
V2B	Y	V2A	DB-V10
V2C	Y (N)	V2B (NONE)	DB-V10 (V11)
V3A	Y	V2C	DB-V10 + V11
V3B	**Y**	**V3A DB-V10**	**DB-V10 + V11**

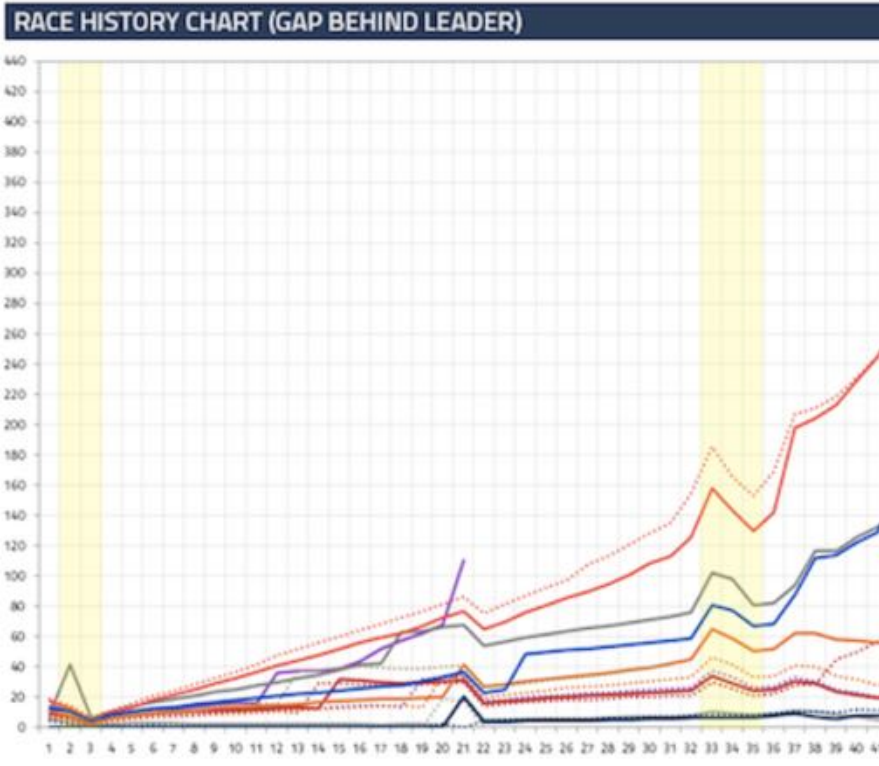
Code version 4, final code

- Same as 3a, with forward-compatible code removed

CODE VERSION	BACKWARD COMPATIBLE	ROLLBACK VERSION	DB VERSION
V1			DB-V10
V2A	Y	V1	DB-V10
V2B	Y	V2A	DB-V10
V2C	Y (N)	V2B (NONE)	DB-V10 (V11)
V3A	Y	V2C	DB-V10 + V11
V3B	Y	V3A DB-V10	DB-V10 + V11
V4	**Y**	**V3B**	**DB-V11**

Lap times graphs





SMHARTER

=> LATENT-TO-LIVE CODE

=> FORWARD COMPATIBLE INTERIM VERSIONS
AKA TWO-PHASE RELEASE

=> TRUNK BASED DEVELOPMENT

=> FEATURE TOGGLES (+BRANCH BY ABSTRACTION)

These patterns enabled us to go faster and safer at the same time without the need to tradeoff quality and safety for speed.

CONCLUSIONS

SMHARTER

EMAIL: LUCA.MINUDEL@SMHARTER.COM

HTTPS://WWW.LINKEDIN.COM/IN/LUCAMINUDEL/

@LUKADOTNET

GET IN TOUCH