

Replatform in a year or die

The tale of a turnaround

myob

@KenScambler







myob

ωλορ

I. The crisis

INVOICE

Customer: Terms: Tax InclusiveCurrency: 1 AUD = 0.61003 EUR
 Ship to:

Invoice No.:

Online payments

Date:

Customer PO No.:

Ship	Backorder	Item Number	Description	Location	Price (AUD)	Price (EUR)	Disc%	Total (AUD)	Total (EUR)	Job	Tax
3	0	010	Blades	Location1	\$163.92717	100.00		\$491.78	300.00		N-T

Salesperson: Subtotal (EUR): Comment: Freight (EUR): GST Ship Via: Tax (EUR): Promised Date: Total Amount (EUR): = 491.78 AUDJournal Memo: Applied to Date (EUR):

Referral Source: Invoice Delivery Status: Balance Due (EUR):

Money in

\$280
2 invoices issued

\$280
2 invoices overdue

Money out

\$372
Owing to suppliers

\$679
PAYE & KiwiSaver owing

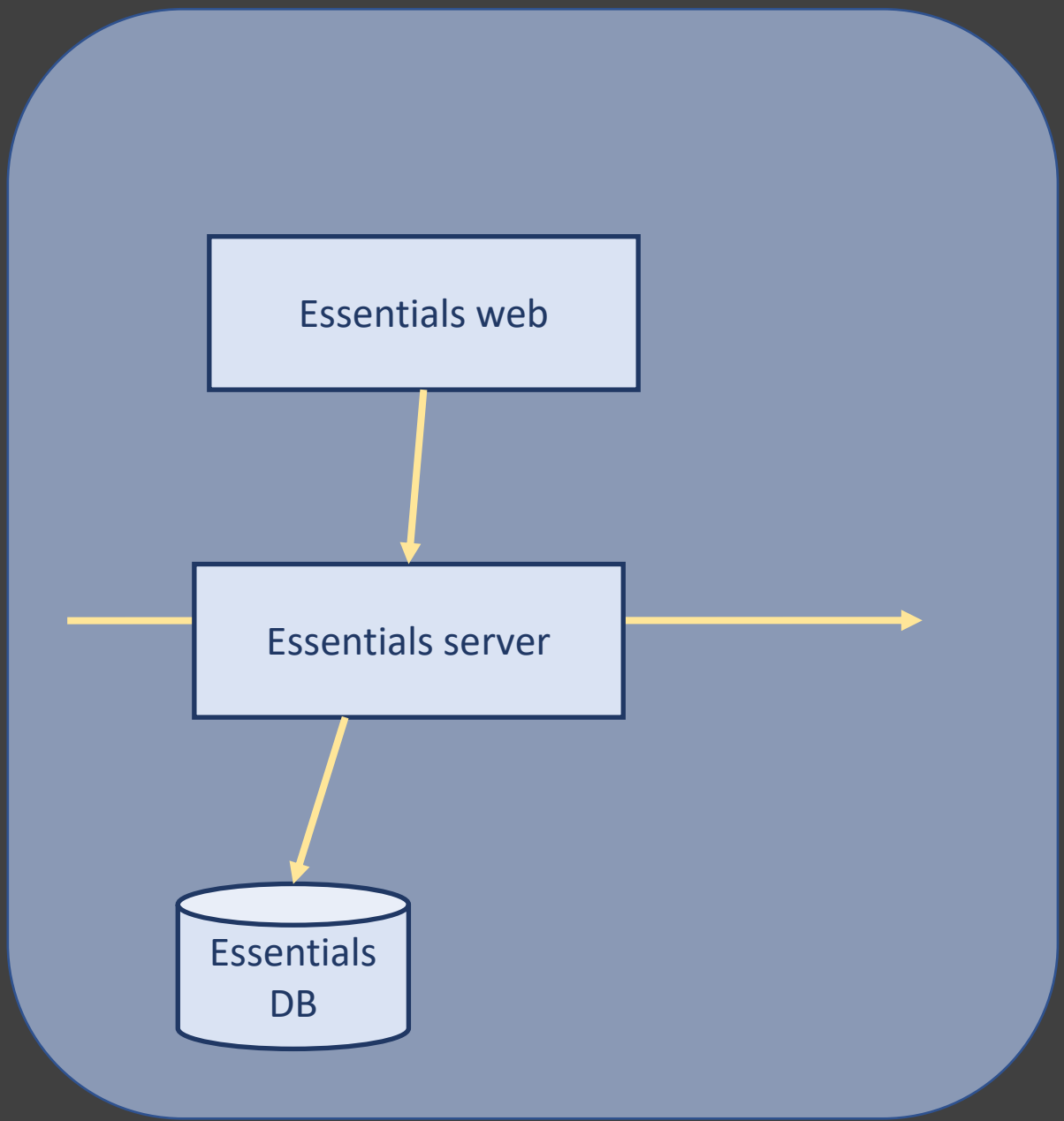
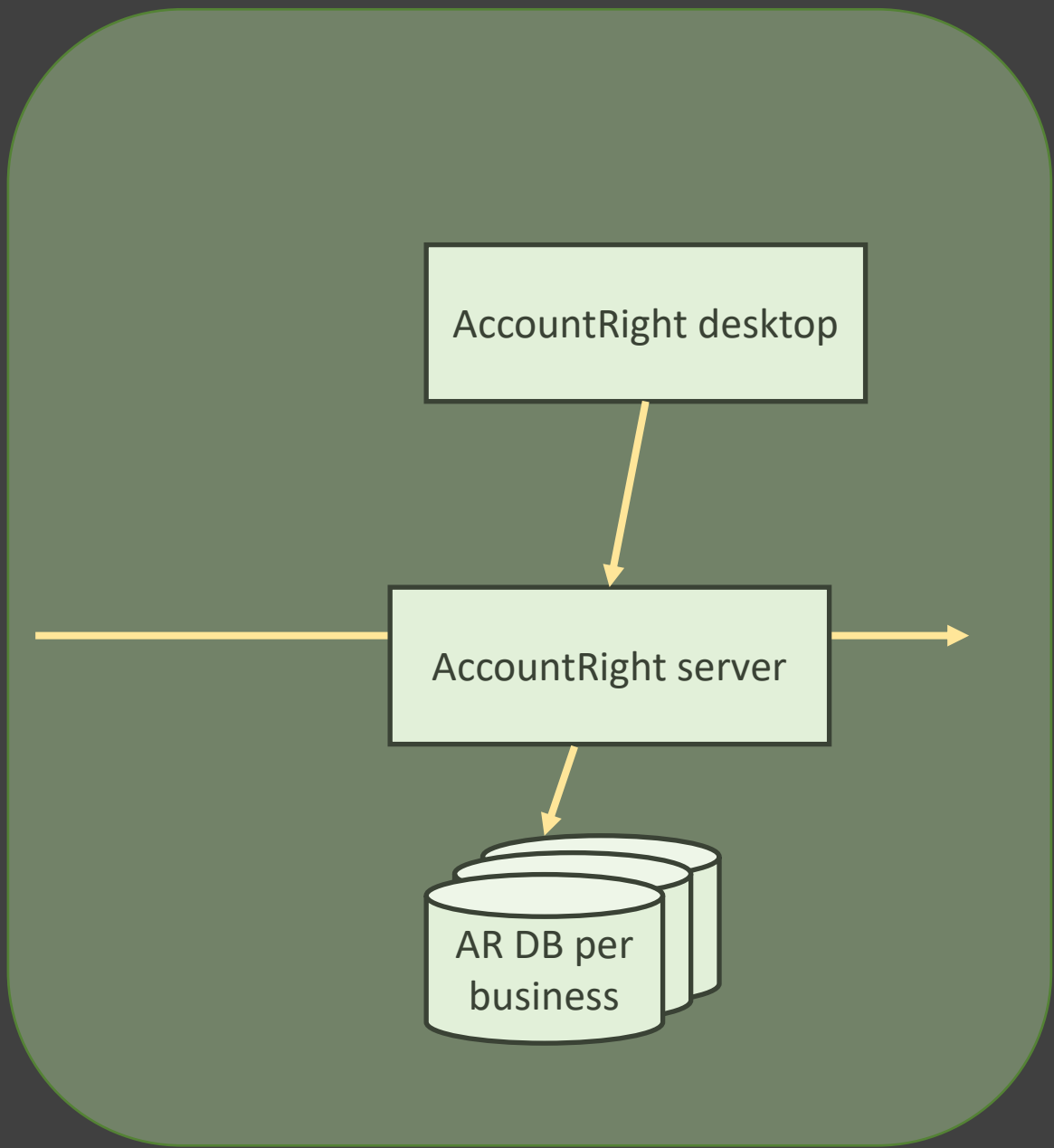
Banking

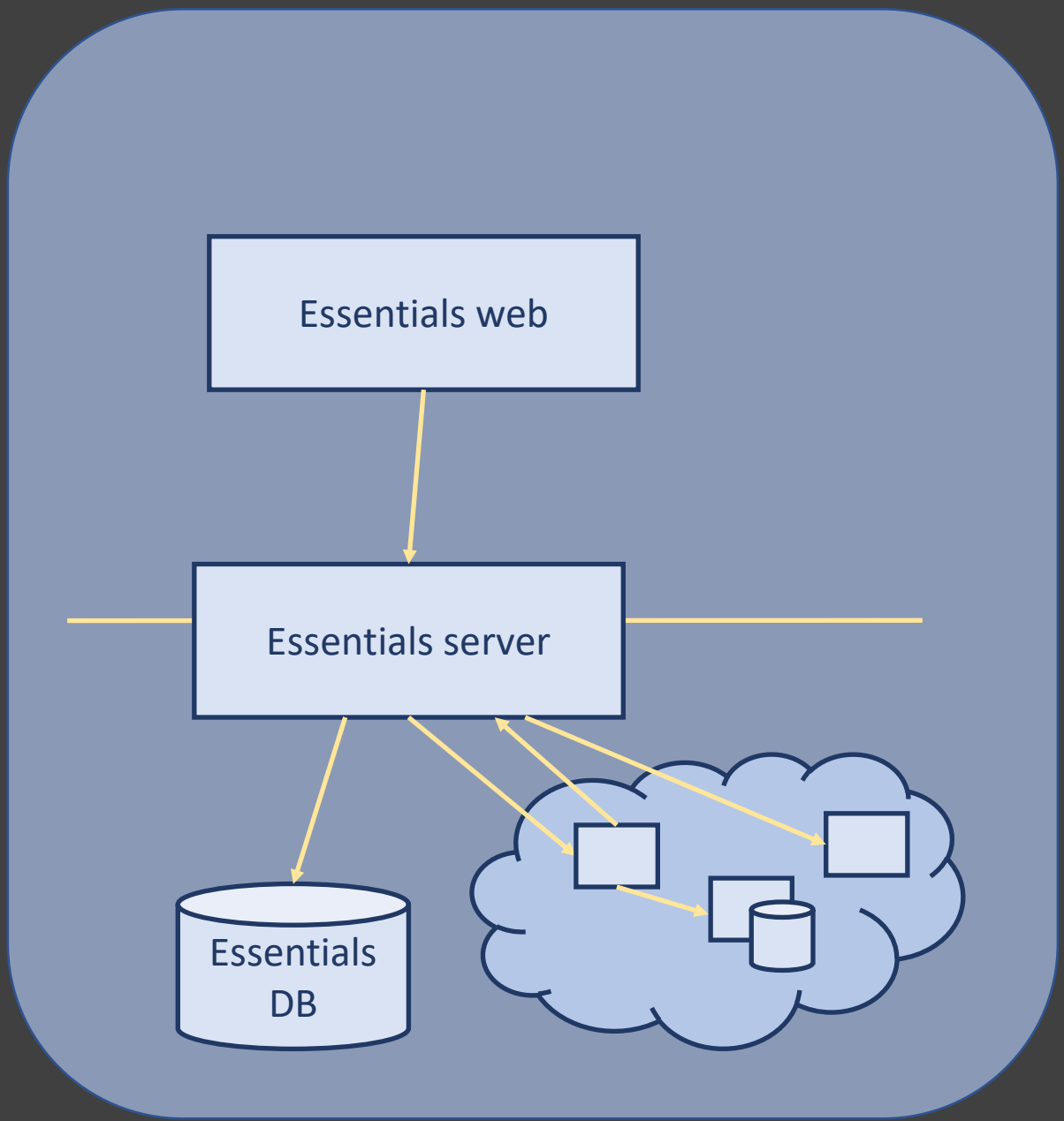
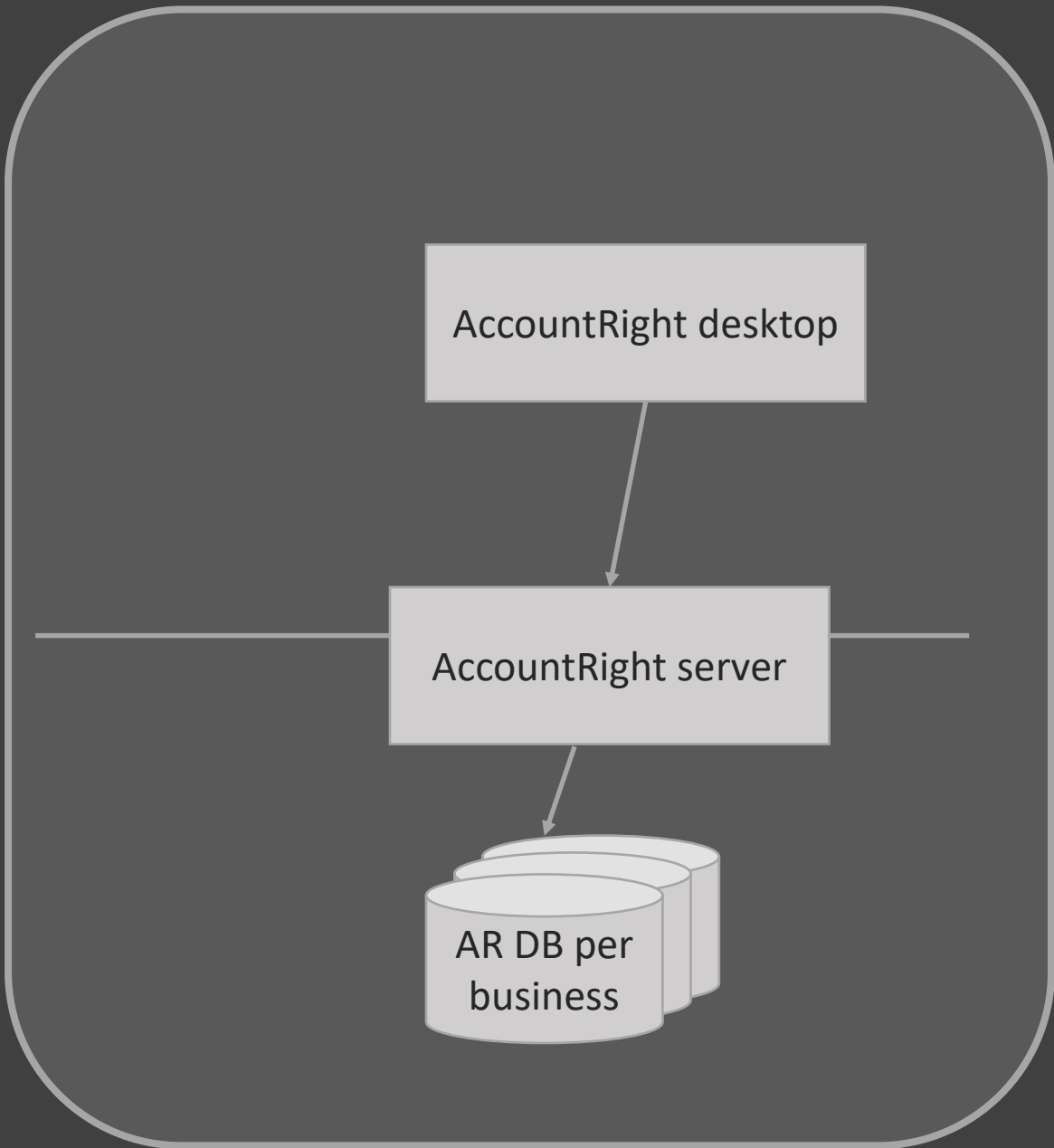
Set up bank feeds

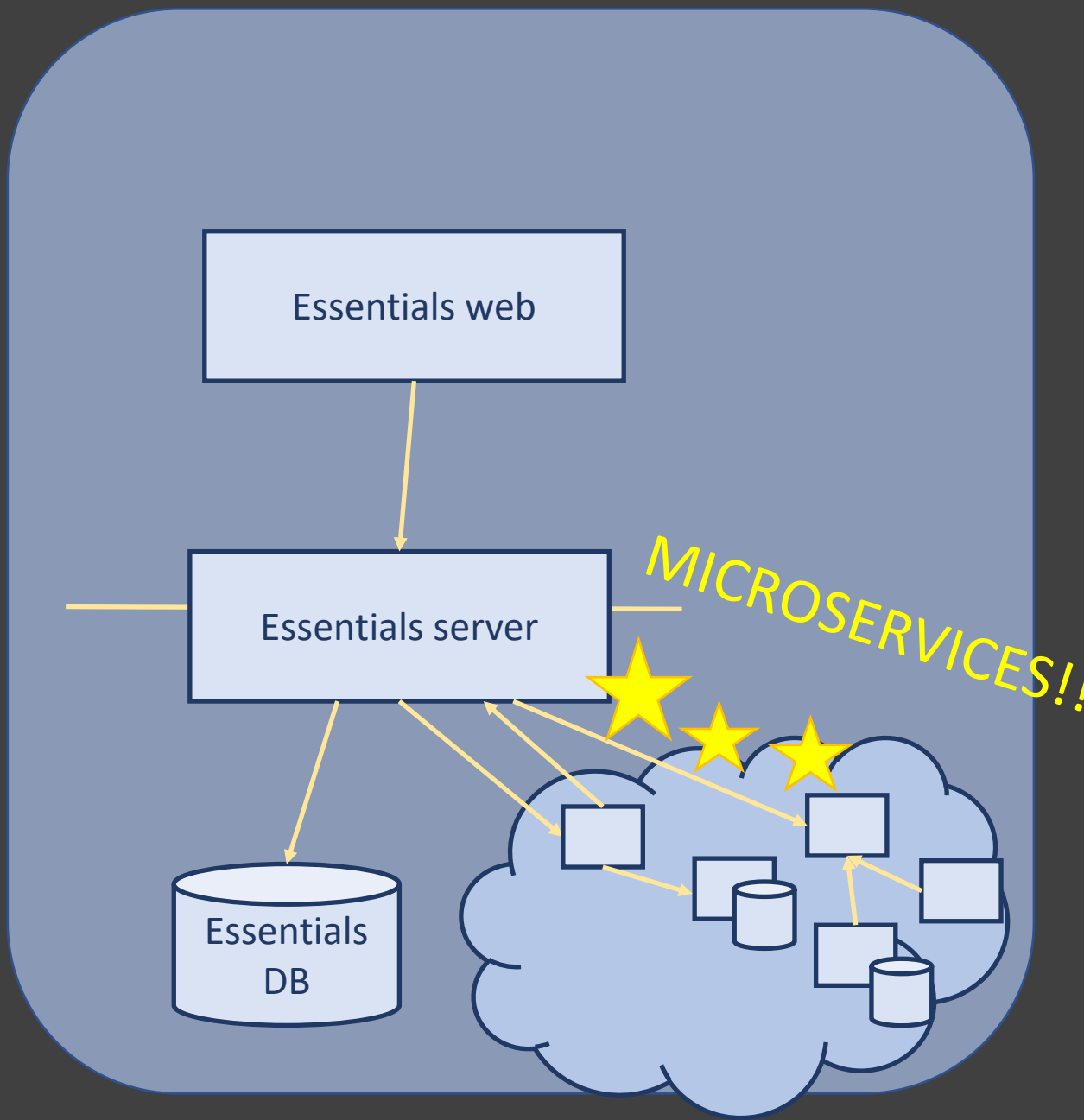
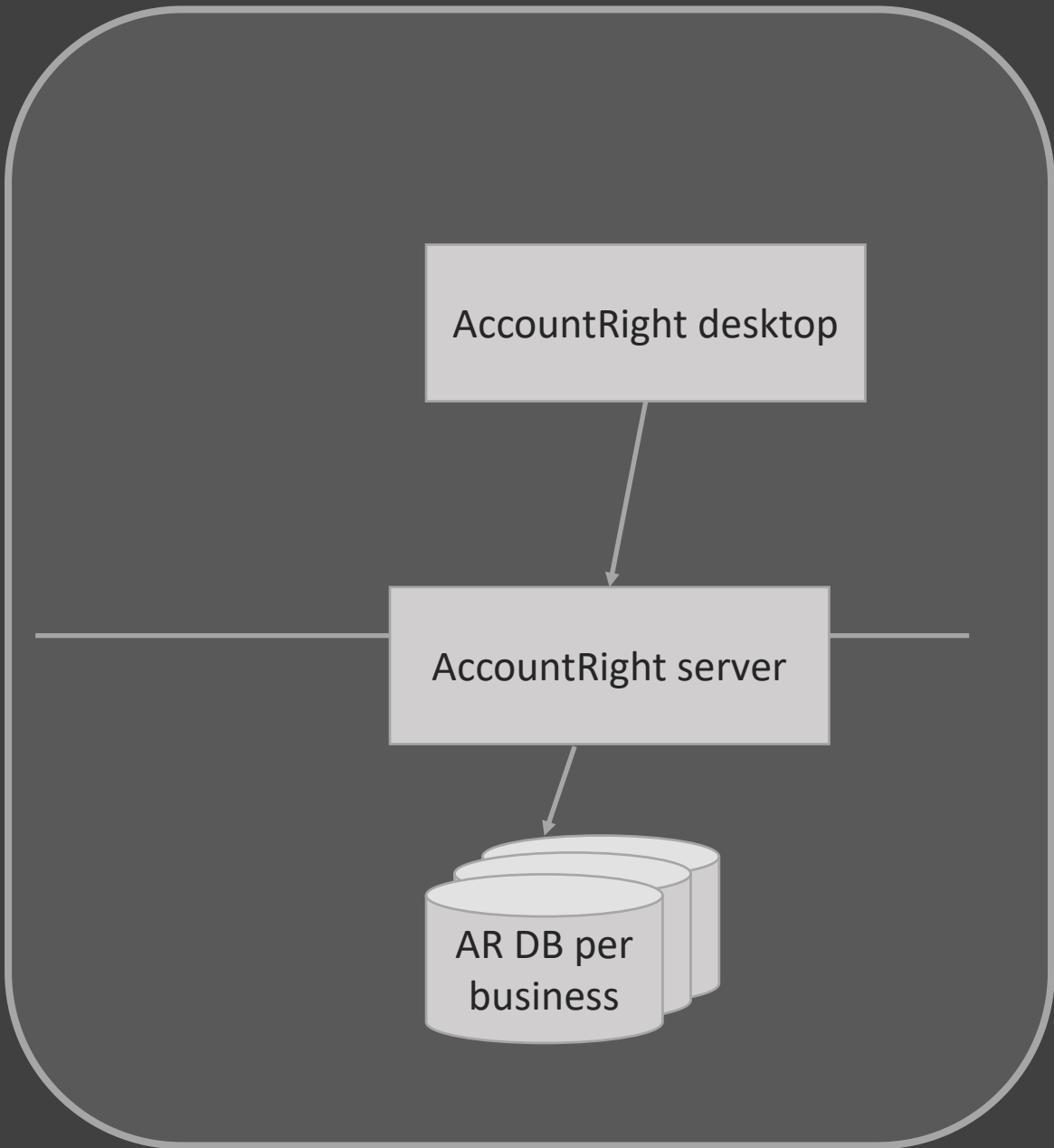
\$9,793
In the bank

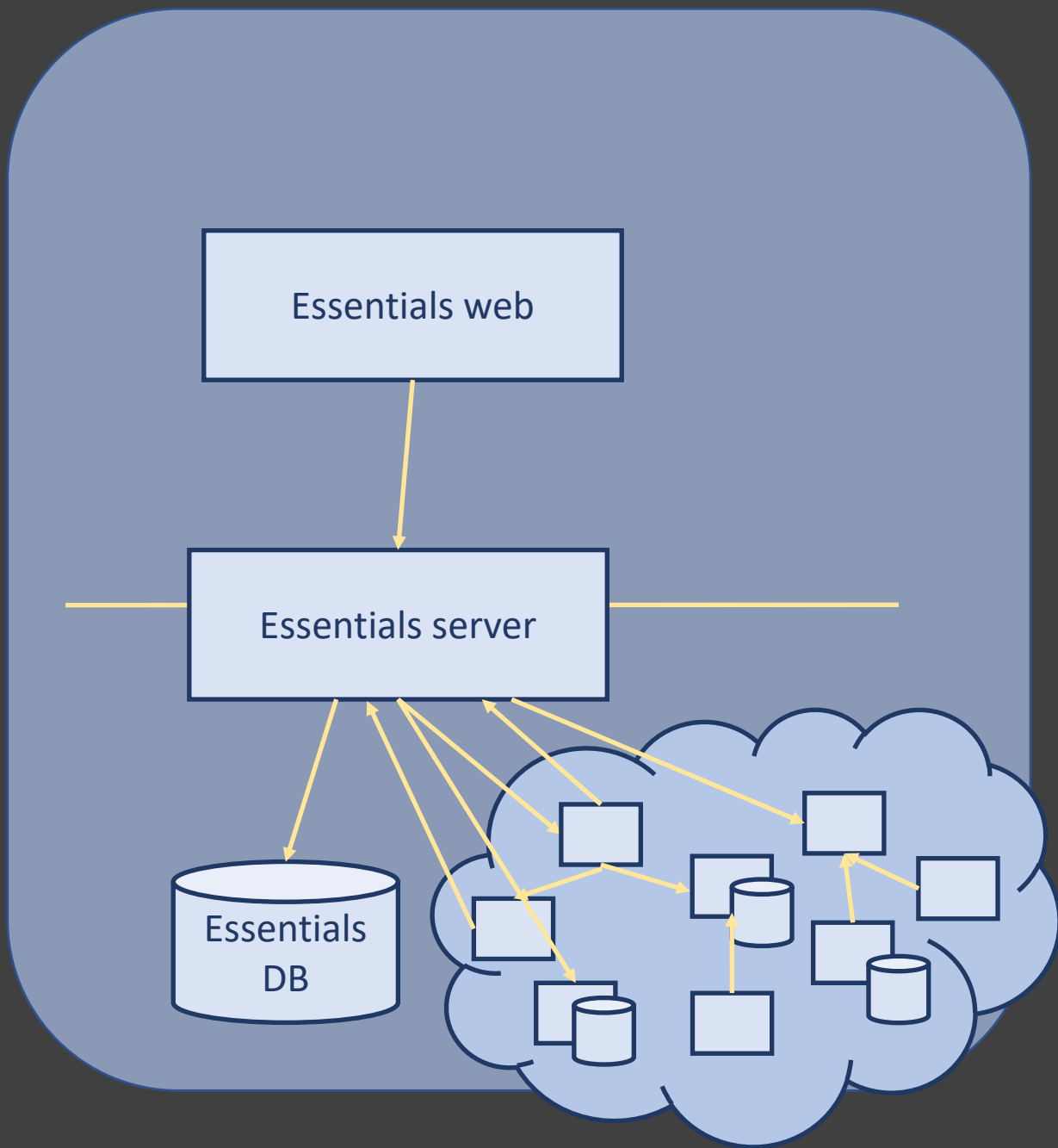
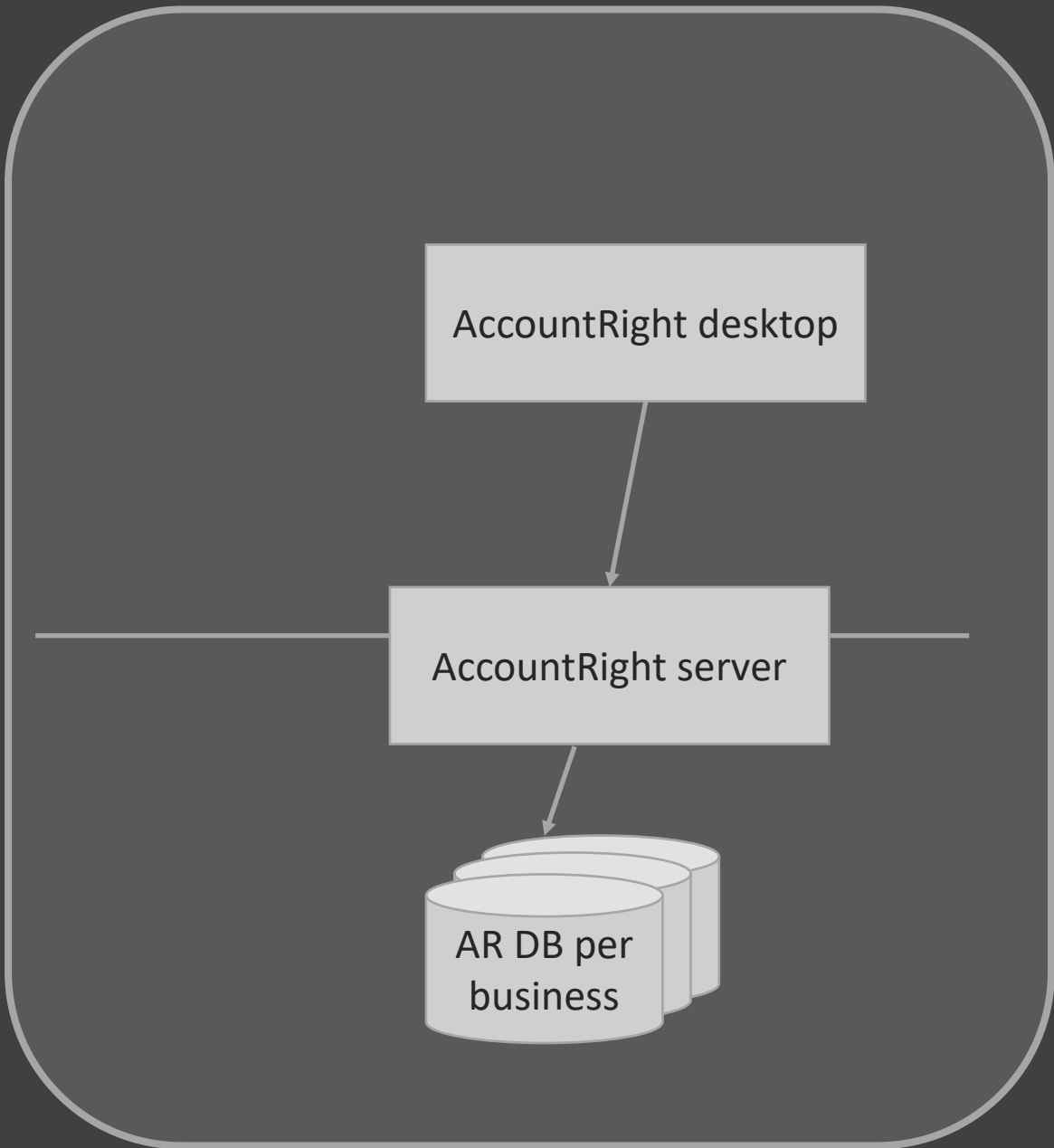
\$0
In credit cards

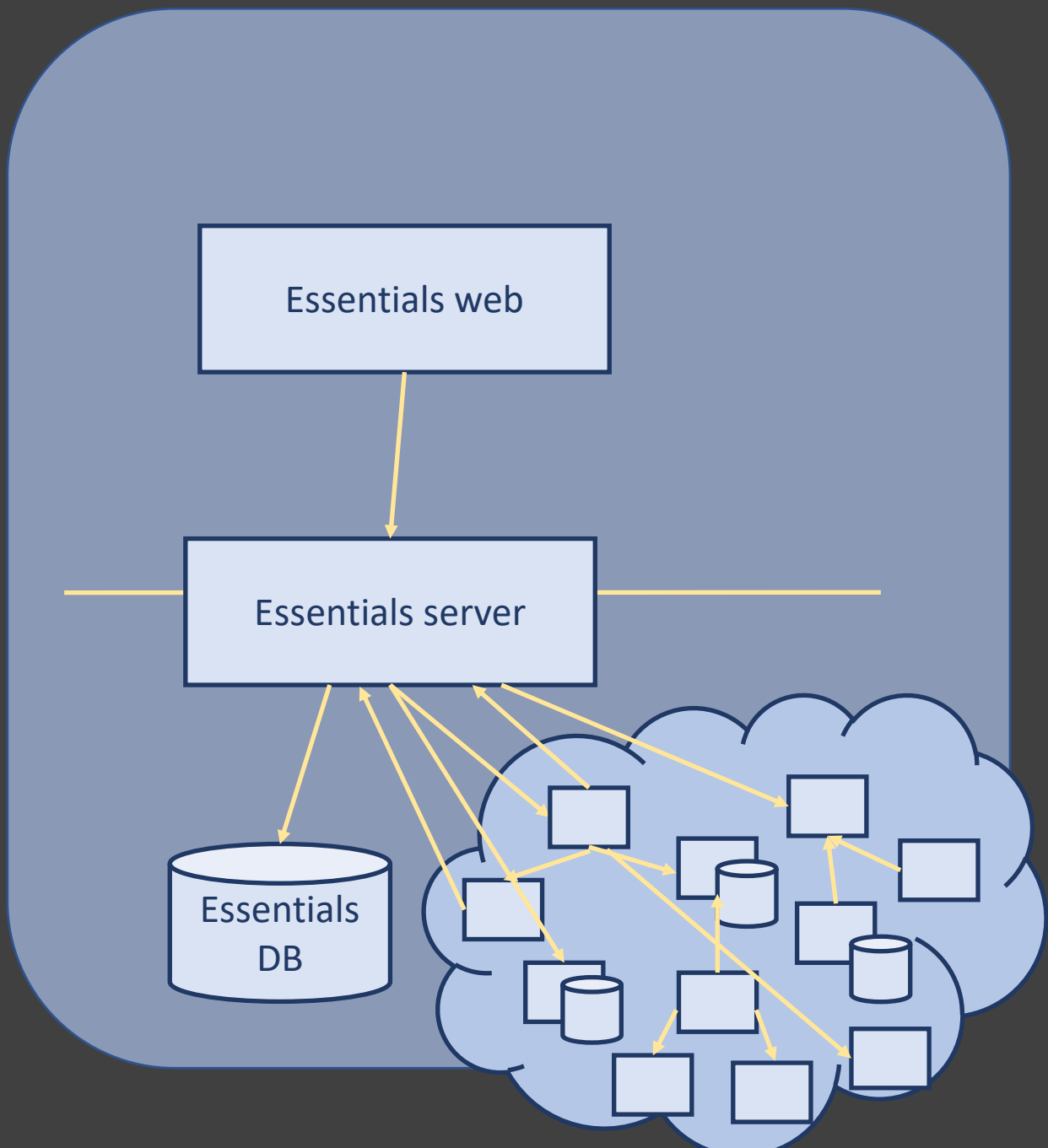
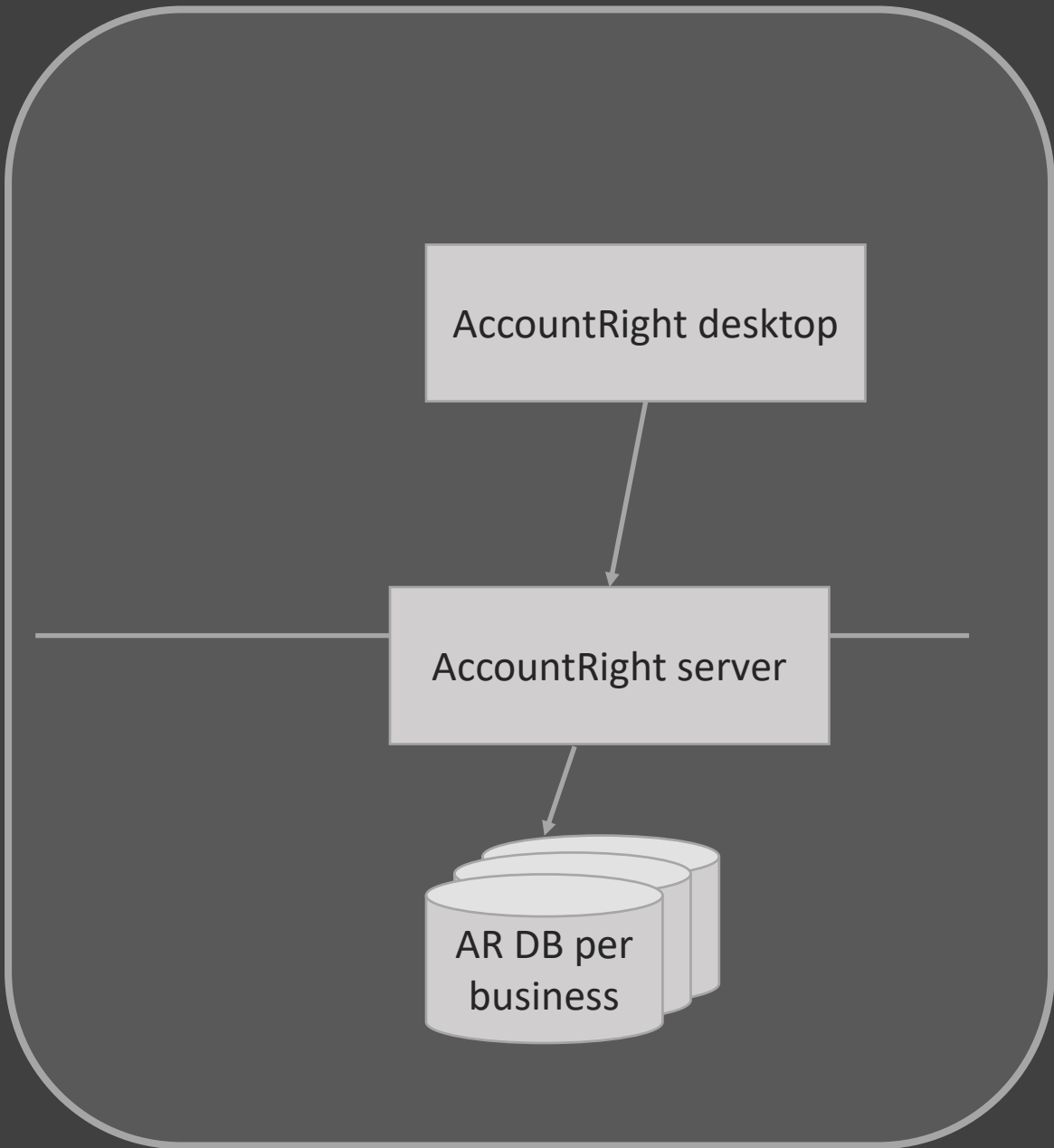
8
Unallocated transactions









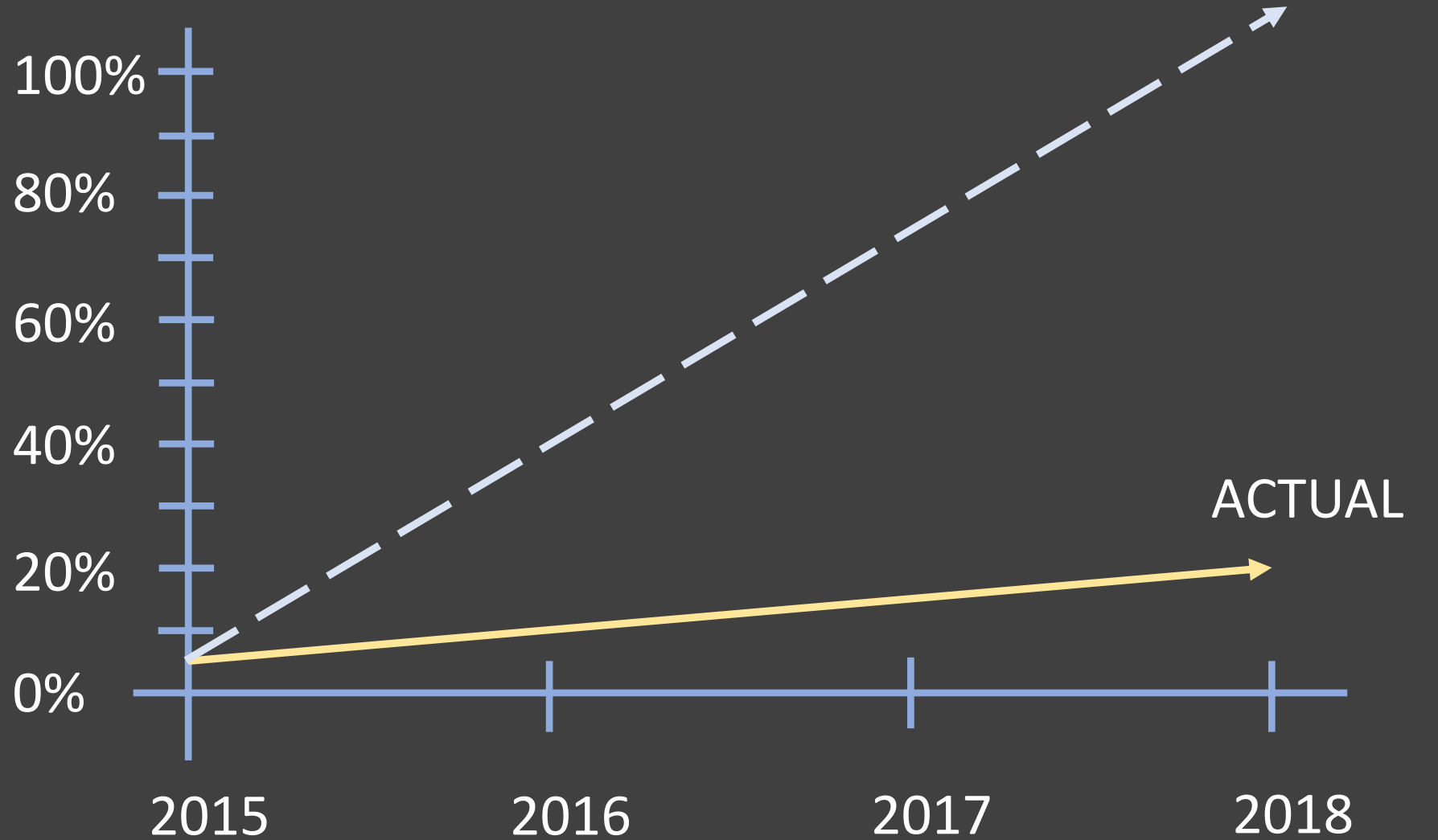


Microservice lessons that everyone knows already

- You're spending a ton on complexity, tooling
- You are supposed to be purchasing team autonomy and release independence
- If fail to achieve the good bits, you don't get a discount on the hard, expensive bits



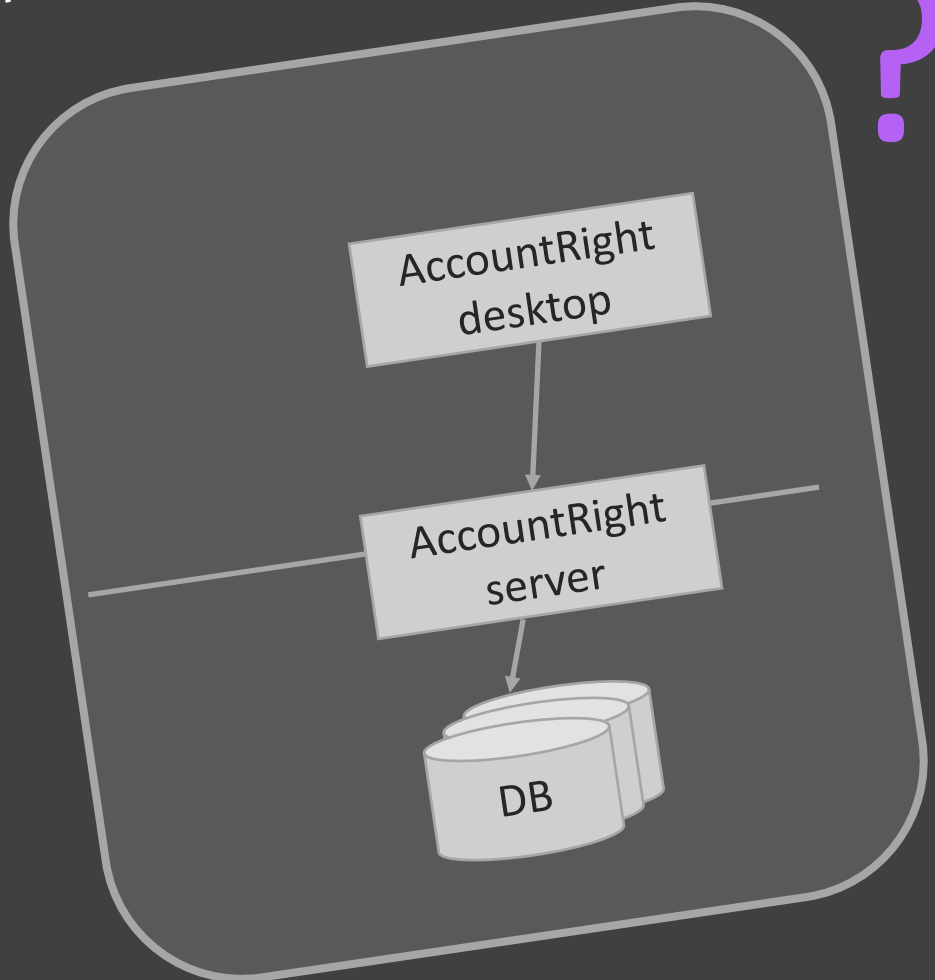
% of Accountright's functionality



ACCONTRIGHT STANDARD
ACCONTRIGHT PLUS
ACCONTRIGHT PREMIER

\$\$
\$\$\$
\$\$\$

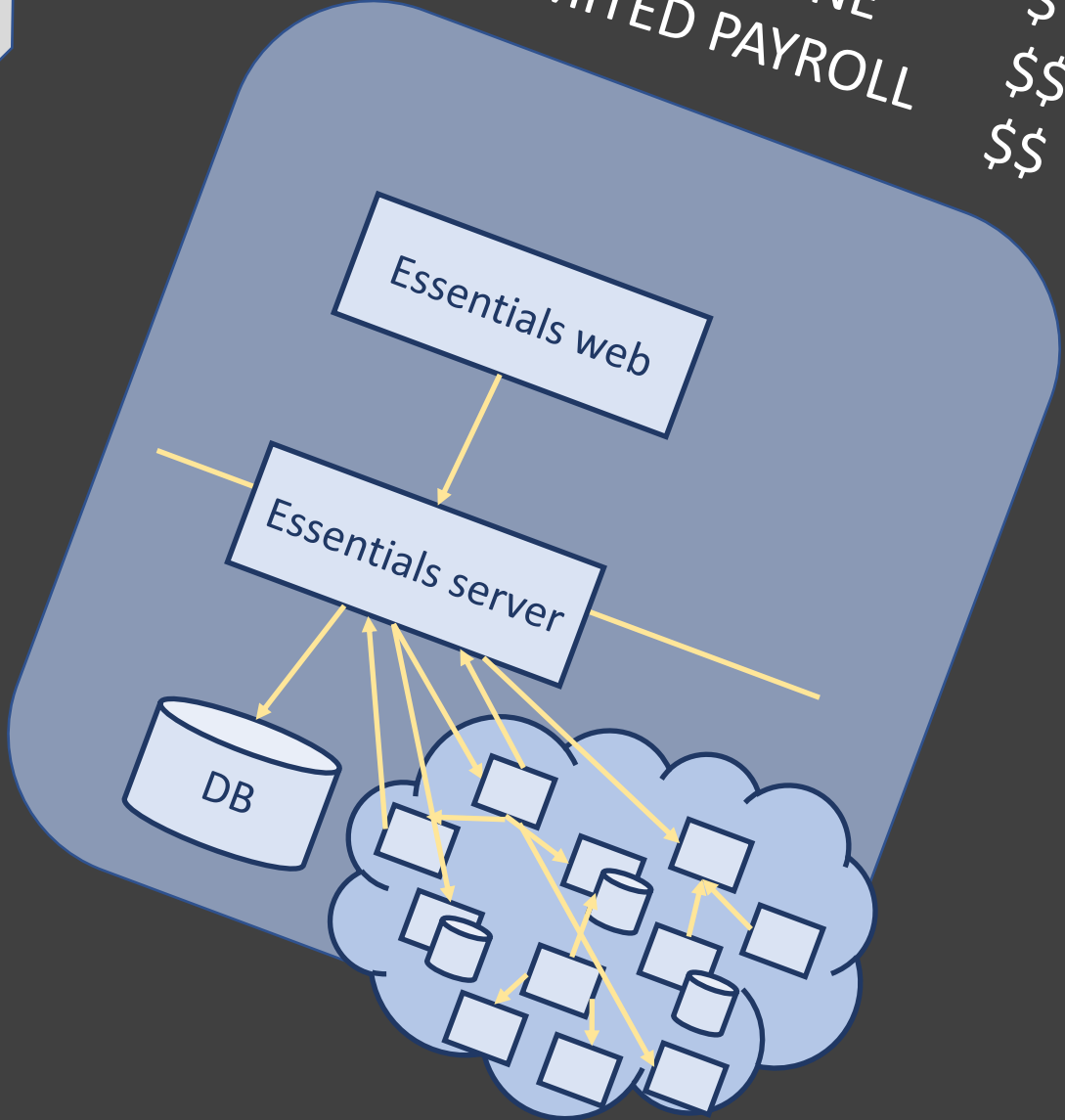
???



???

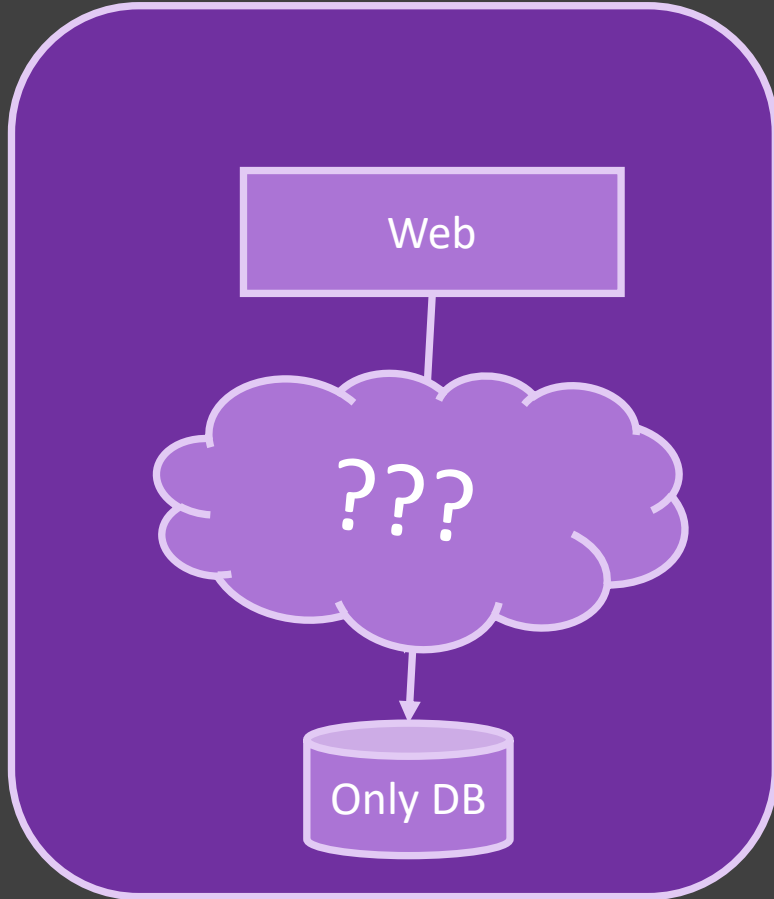
STARTER
PAYROLL FOR ONE
UNLIMITED PAYROLL

\$
\$\$
\$\$



II. The plan

What do we need?



MYOB MINIMUM CHIPS

\$

MYOB SUPER DUPER

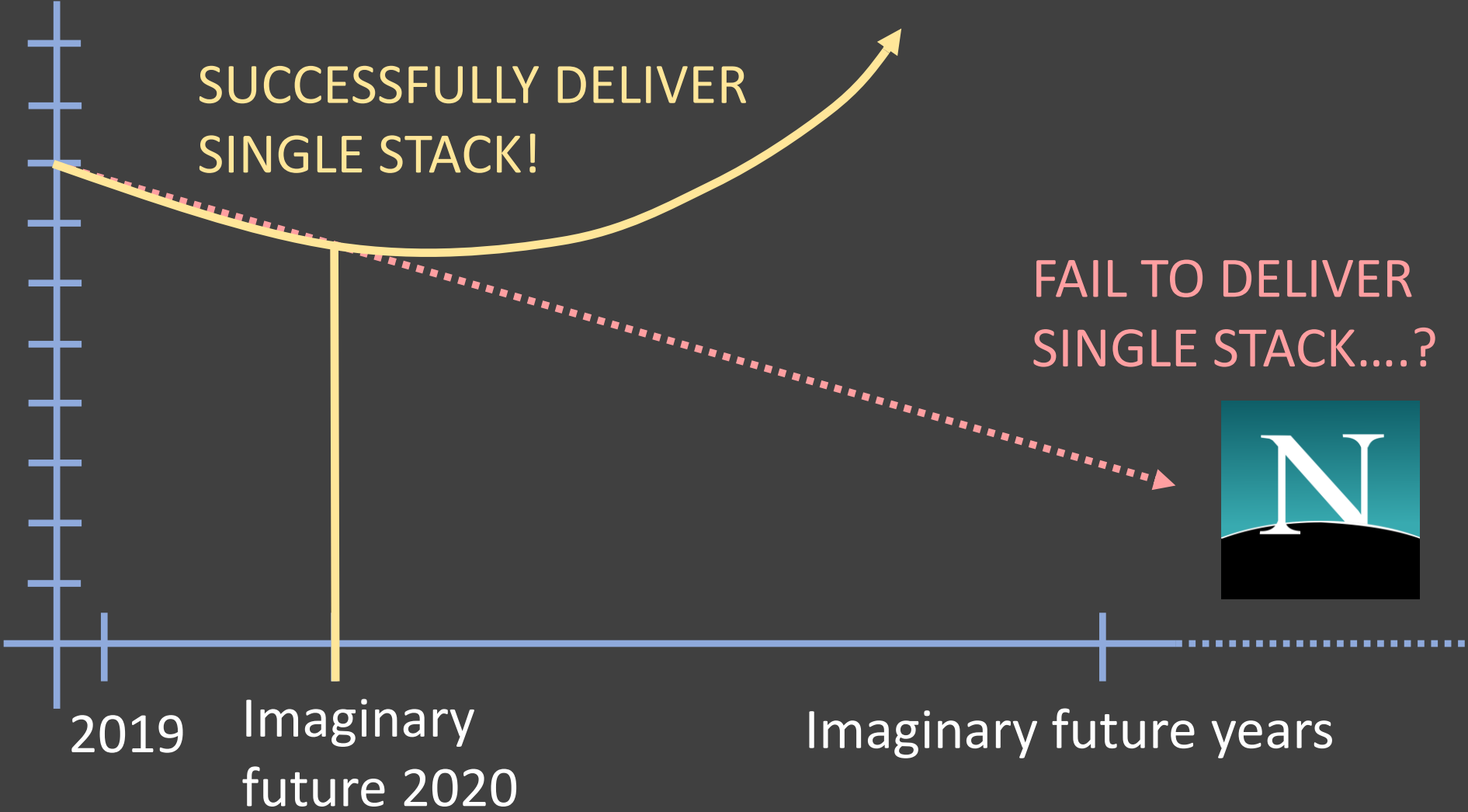
\$\$

MYOB MEGA DELUXE PREMIUM

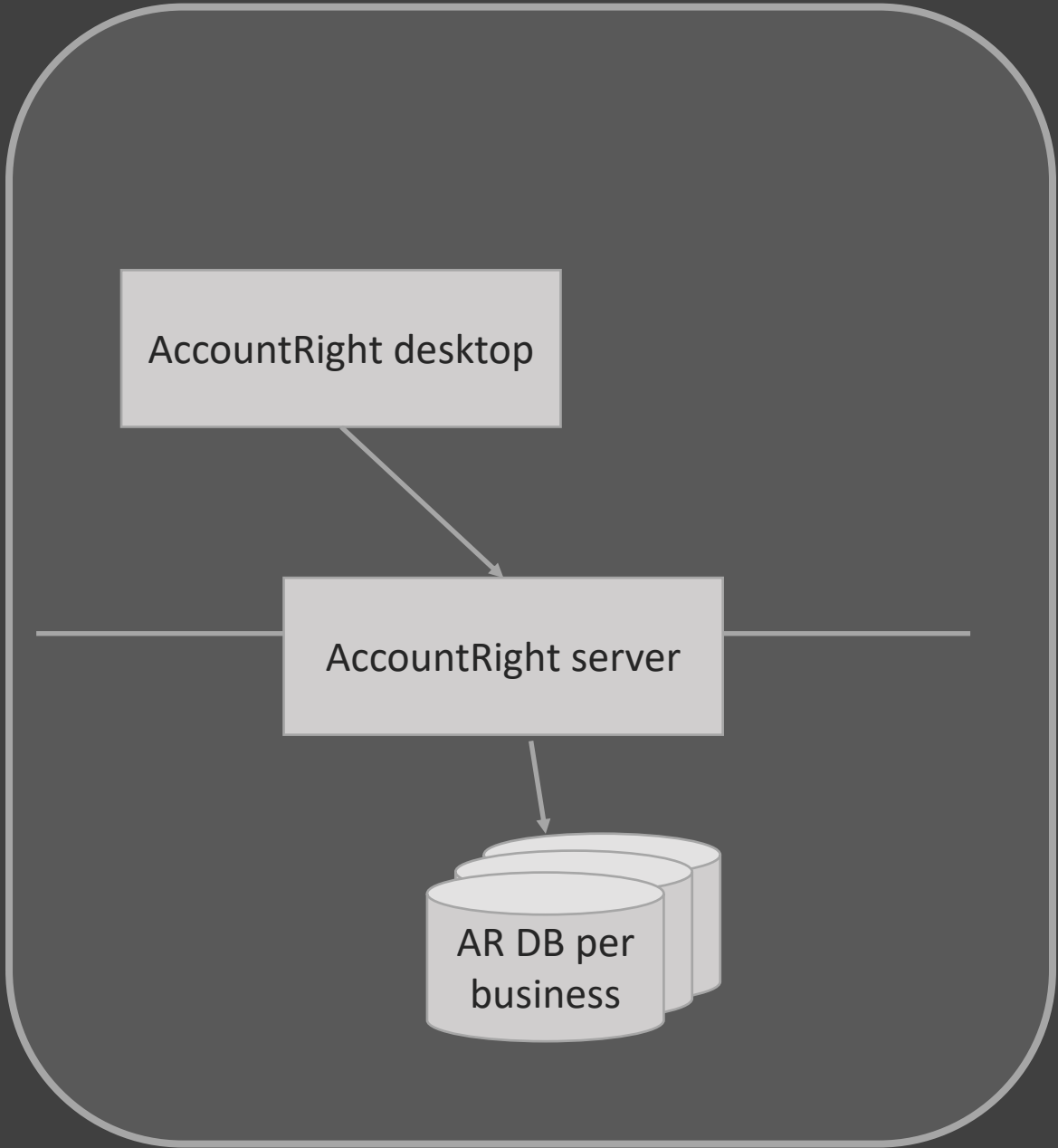
\$\$\$

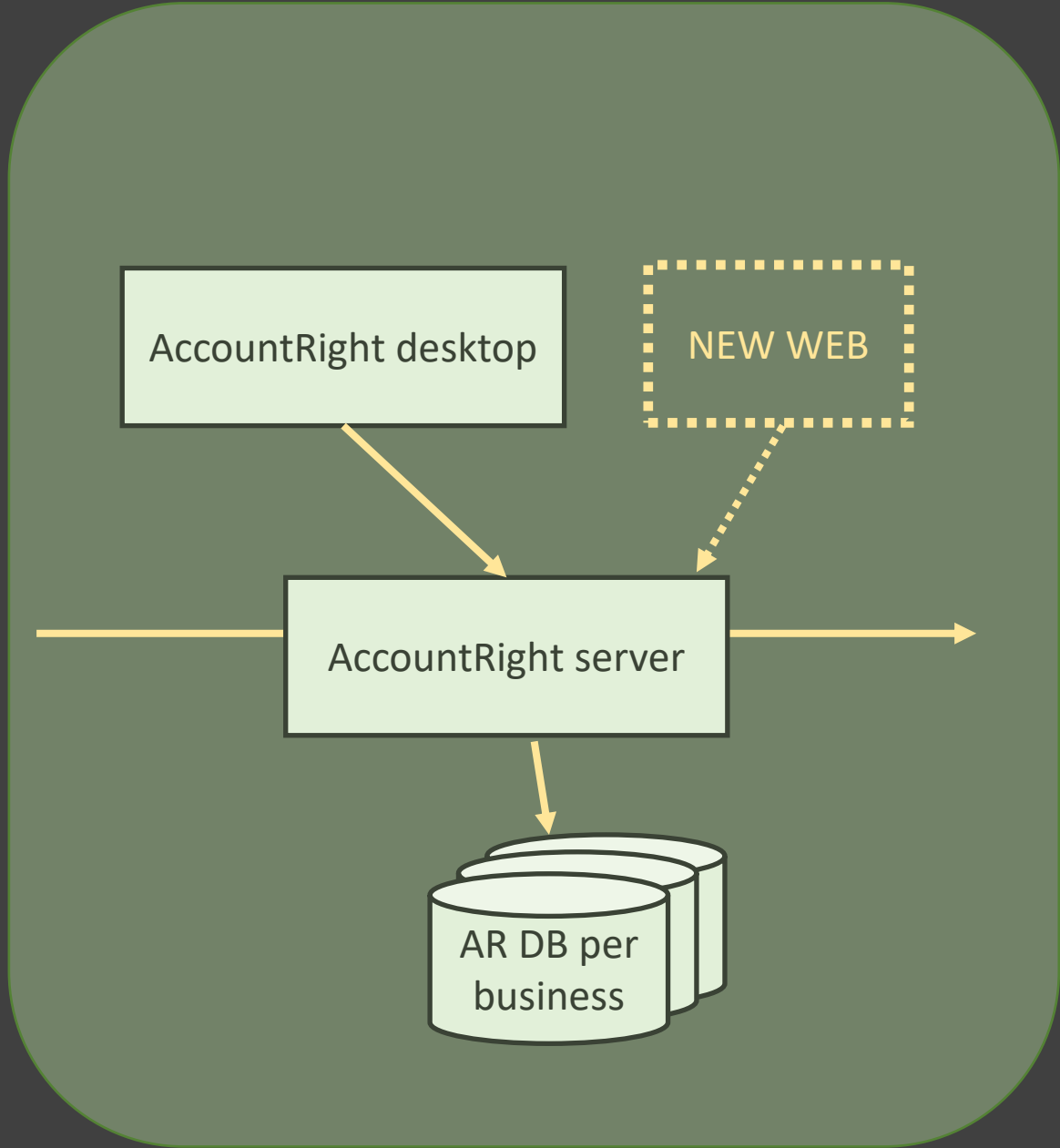
When do we need it?

Axis labels left to the imagination



What do we do?

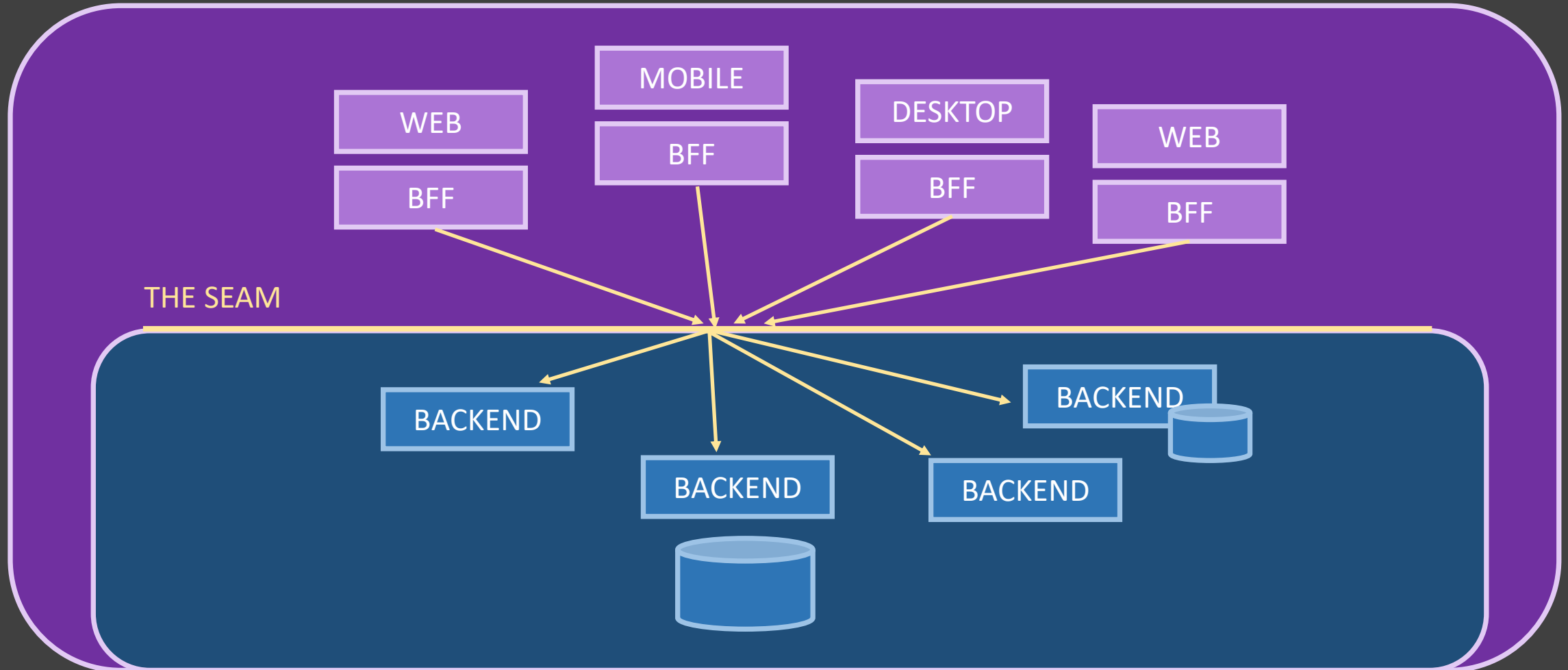




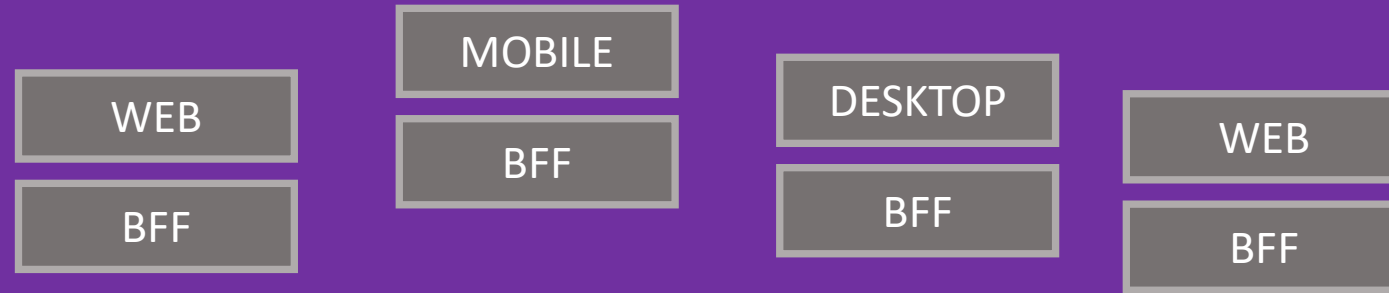
What do we tell people?

New architecture

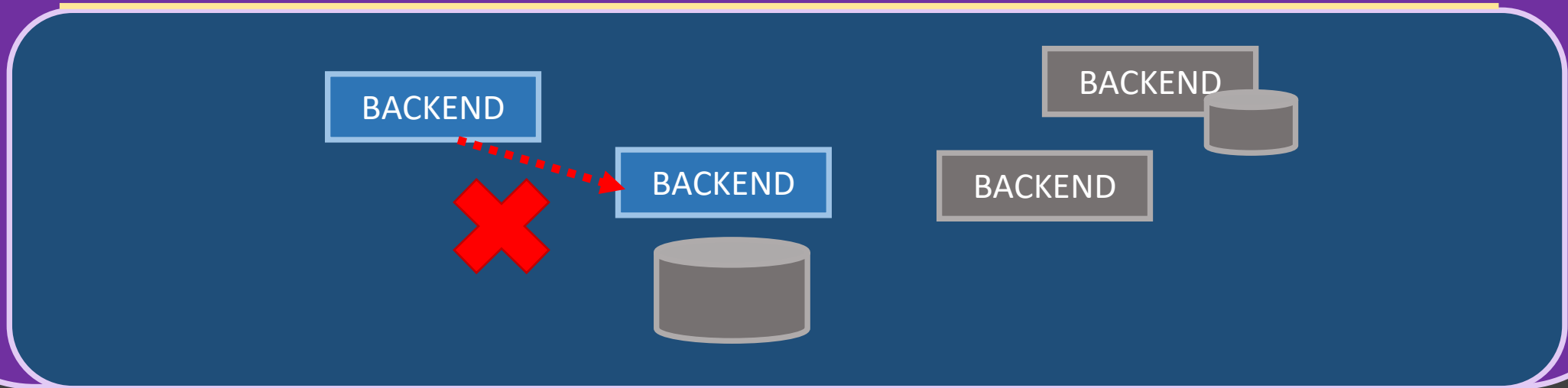
“The seam”



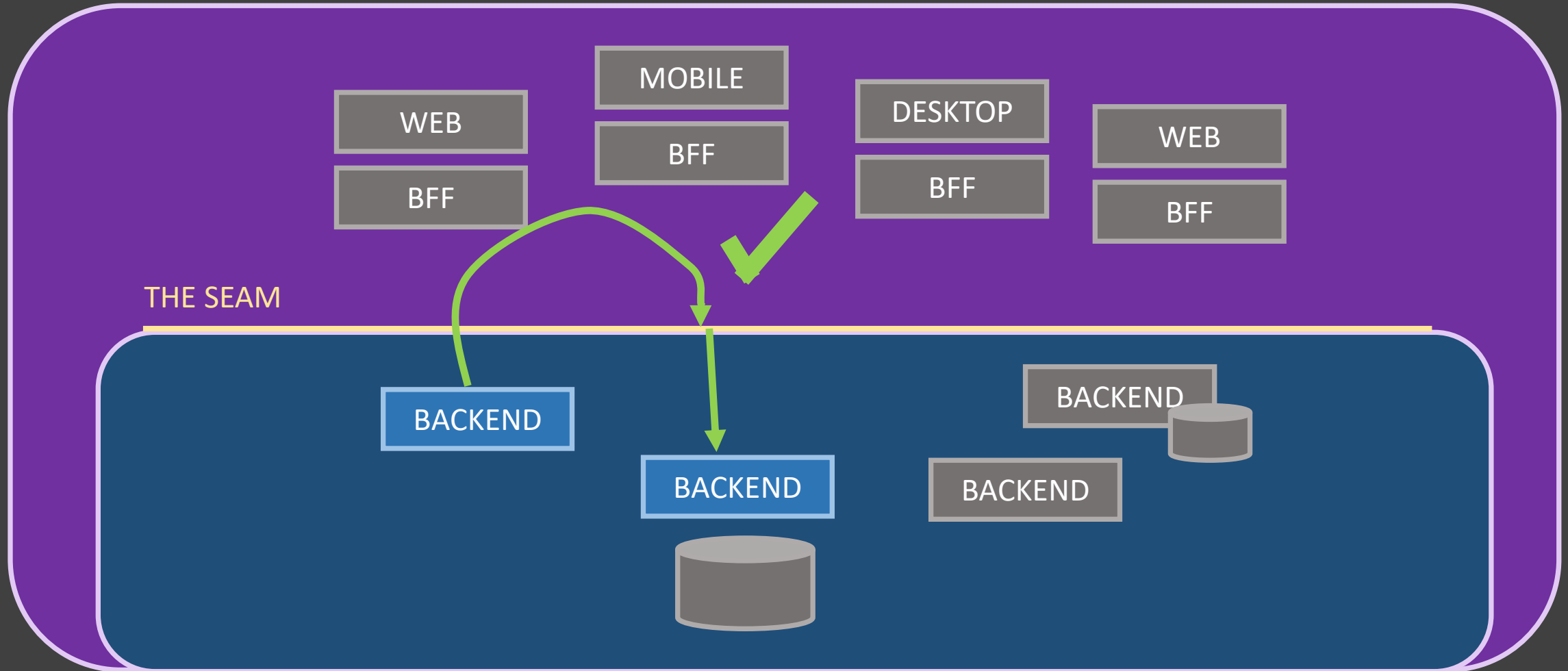
“The seam”



THE SEAM

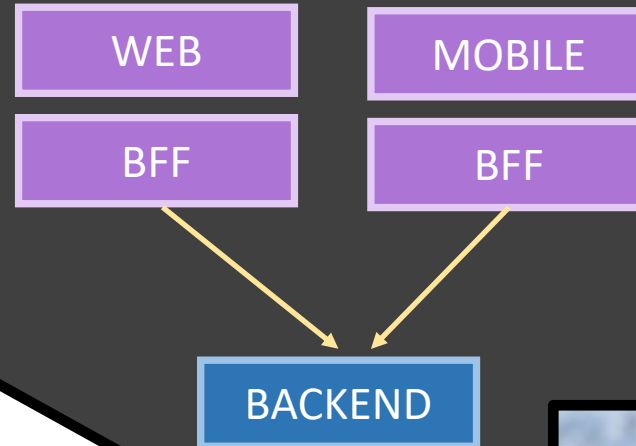


“The seam”



Backends For Frontends

“The simple act of limiting the number of consumers they support makes them much easier to work with and change, and helps teams developing customer-facing applications retain more autonomy.”



Sam Newman

“Pattern: Backends For Frontends”
(2015)

“Malleable monoliths”



Monolith



kenbot

@KenScambler



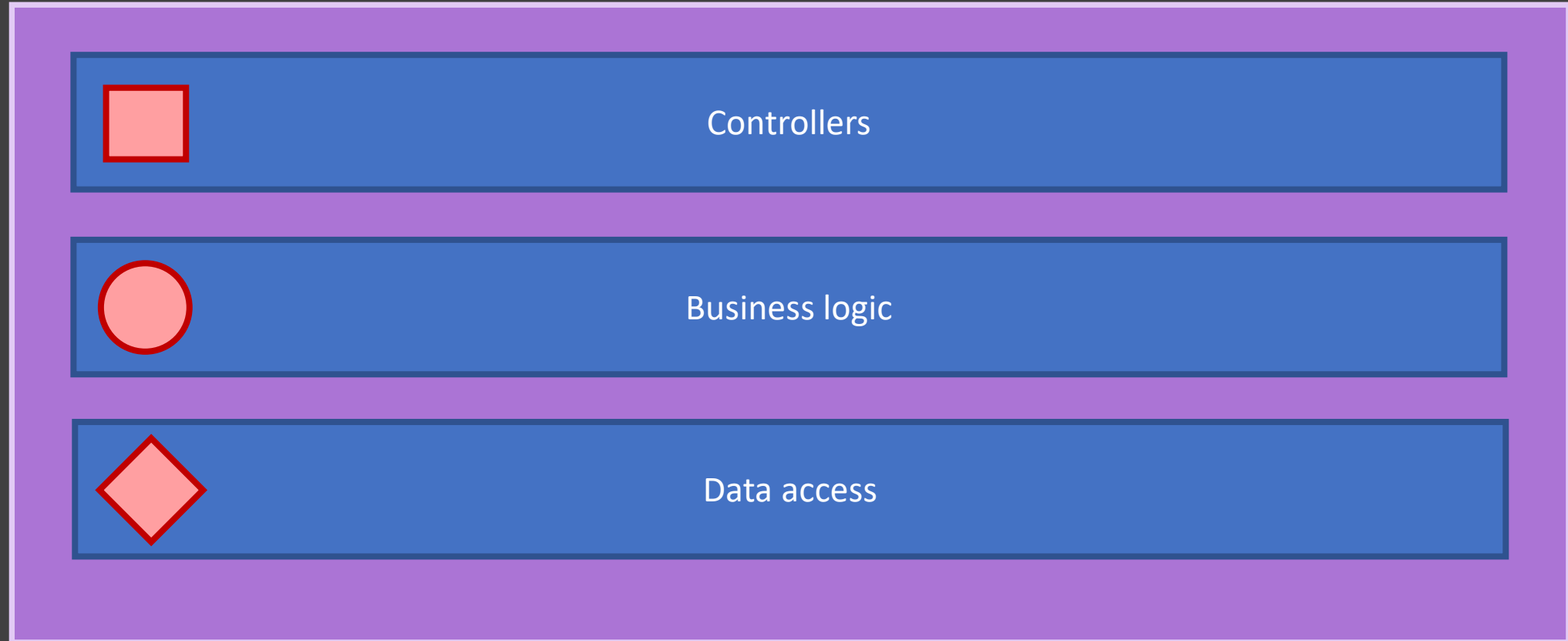
1,000,000 components but you only need to think about 5 at once = a simple system.

20 components but you need to fit them all in your head all the time = a complicated system.

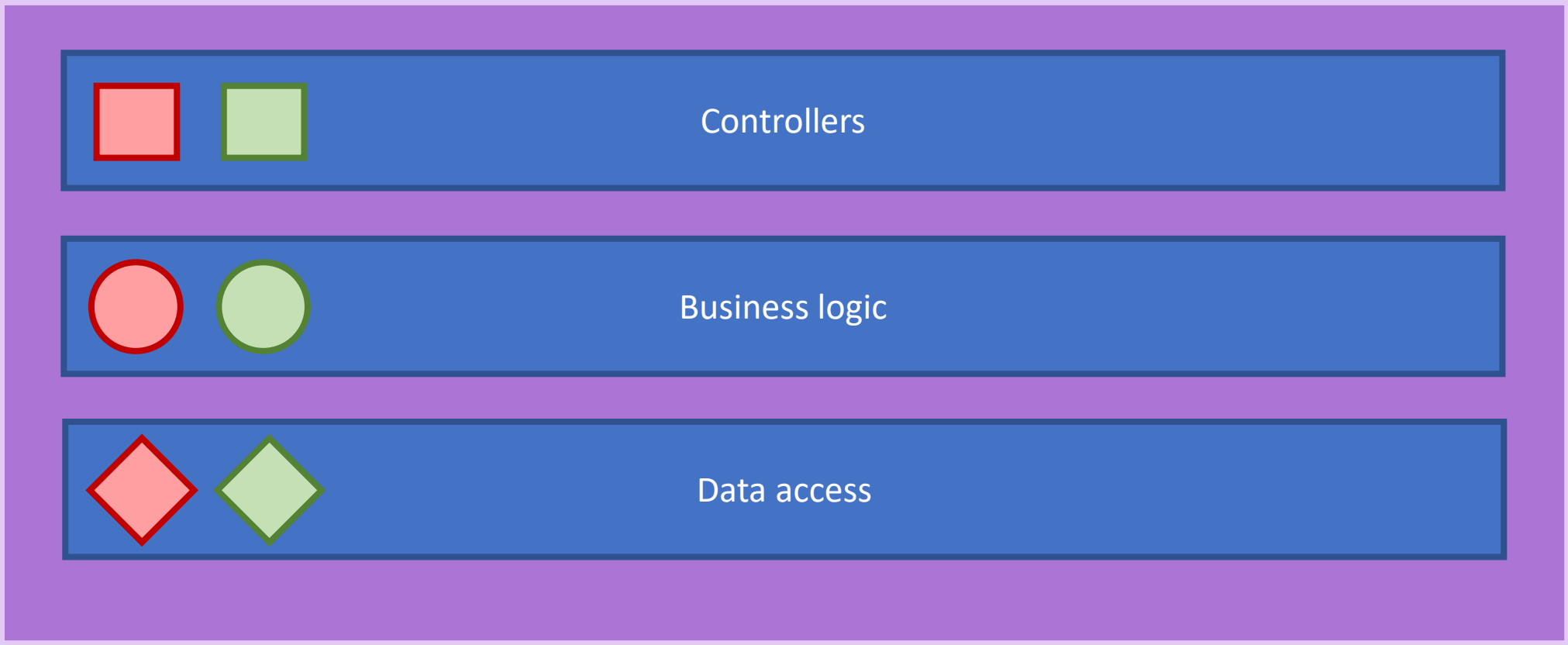
Stopping things from knowing about each other is the big game in growing a system

8:26 PM · Jun 18, 2018 · Twitter for Android

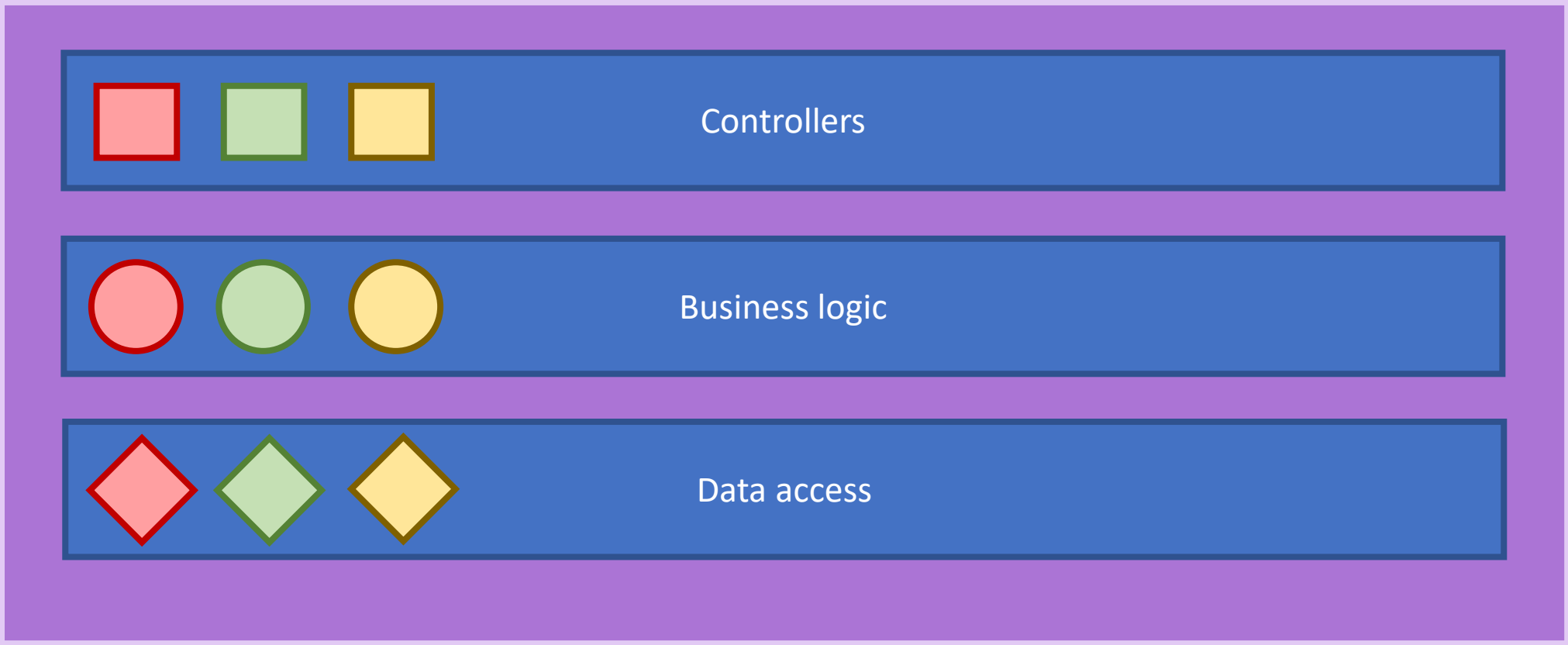
Horizontal slicing



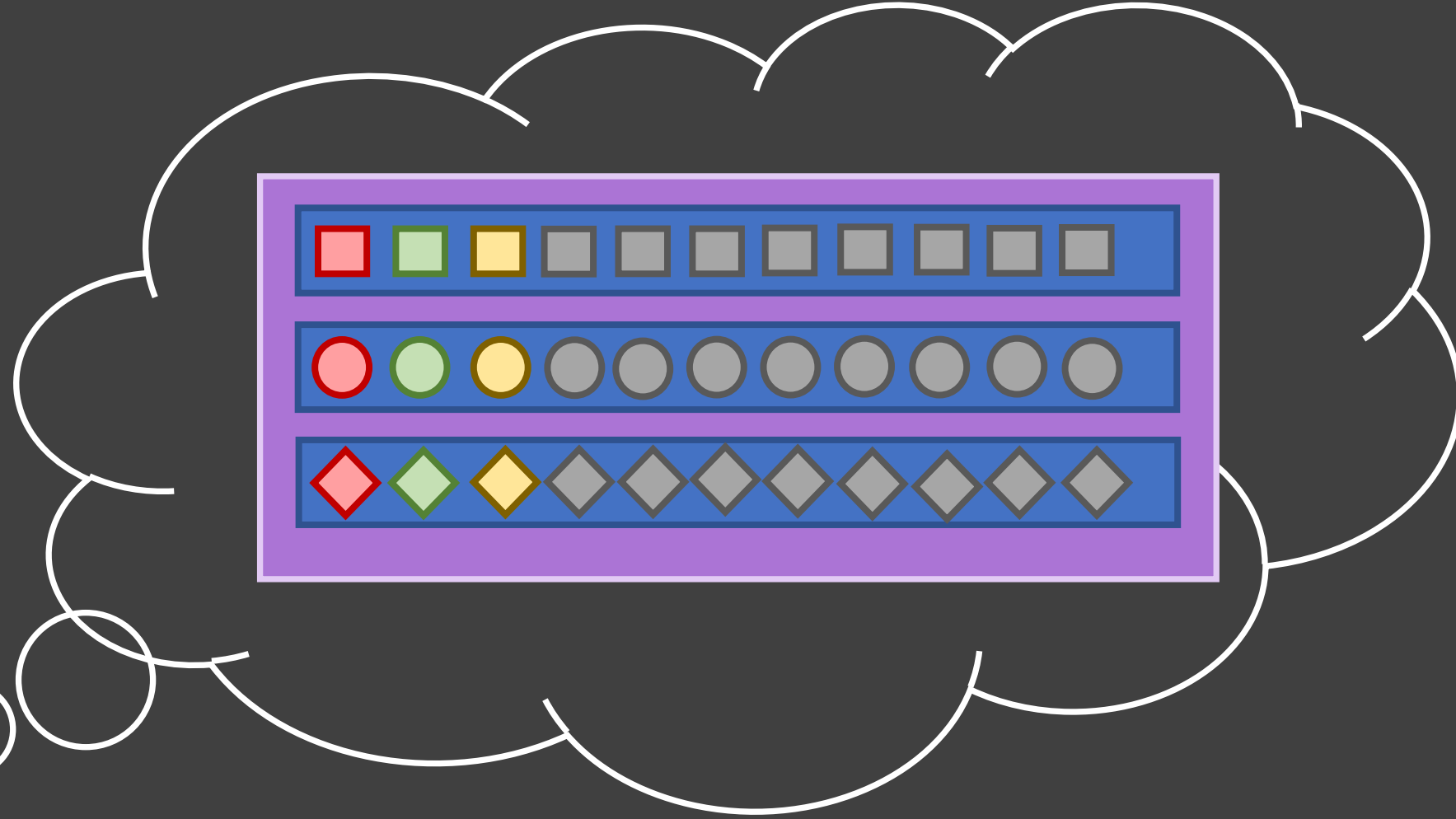
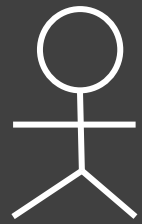
Horizontal slicing



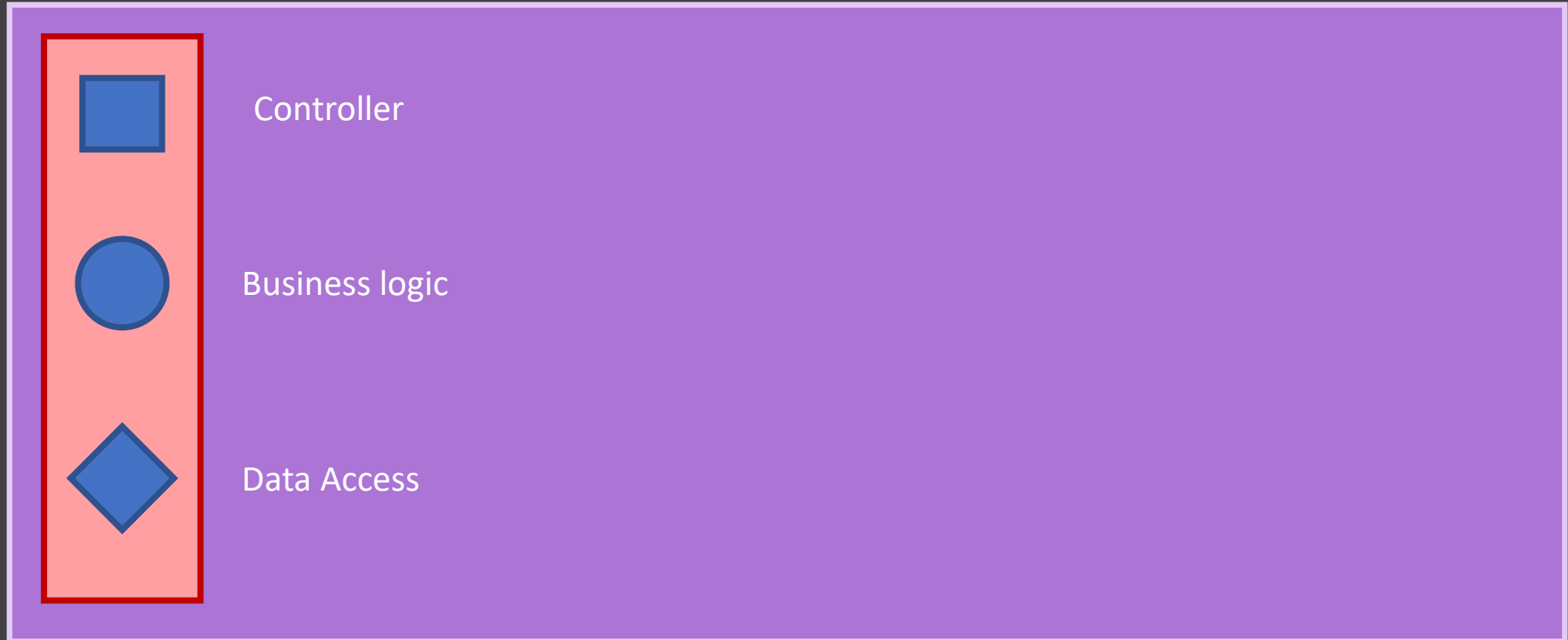
Horizontal slicing



Cognitive load!



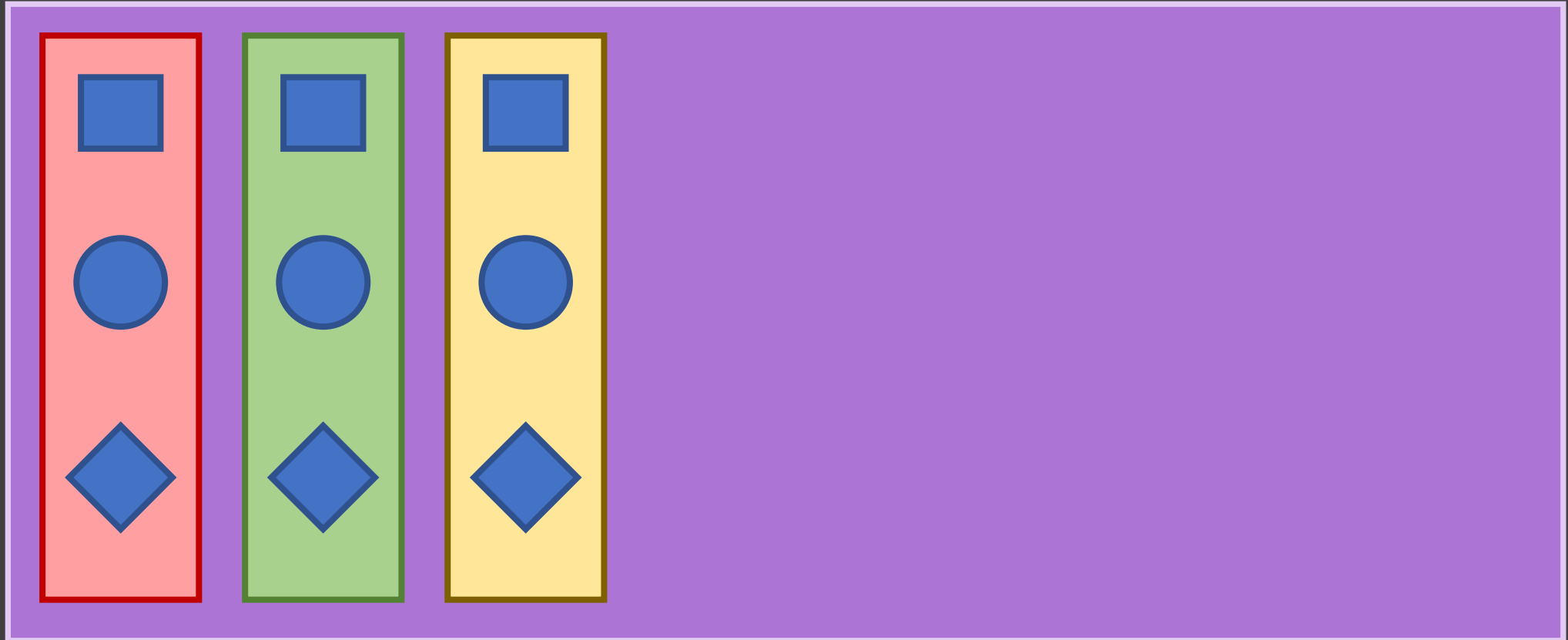
Vertical slicing



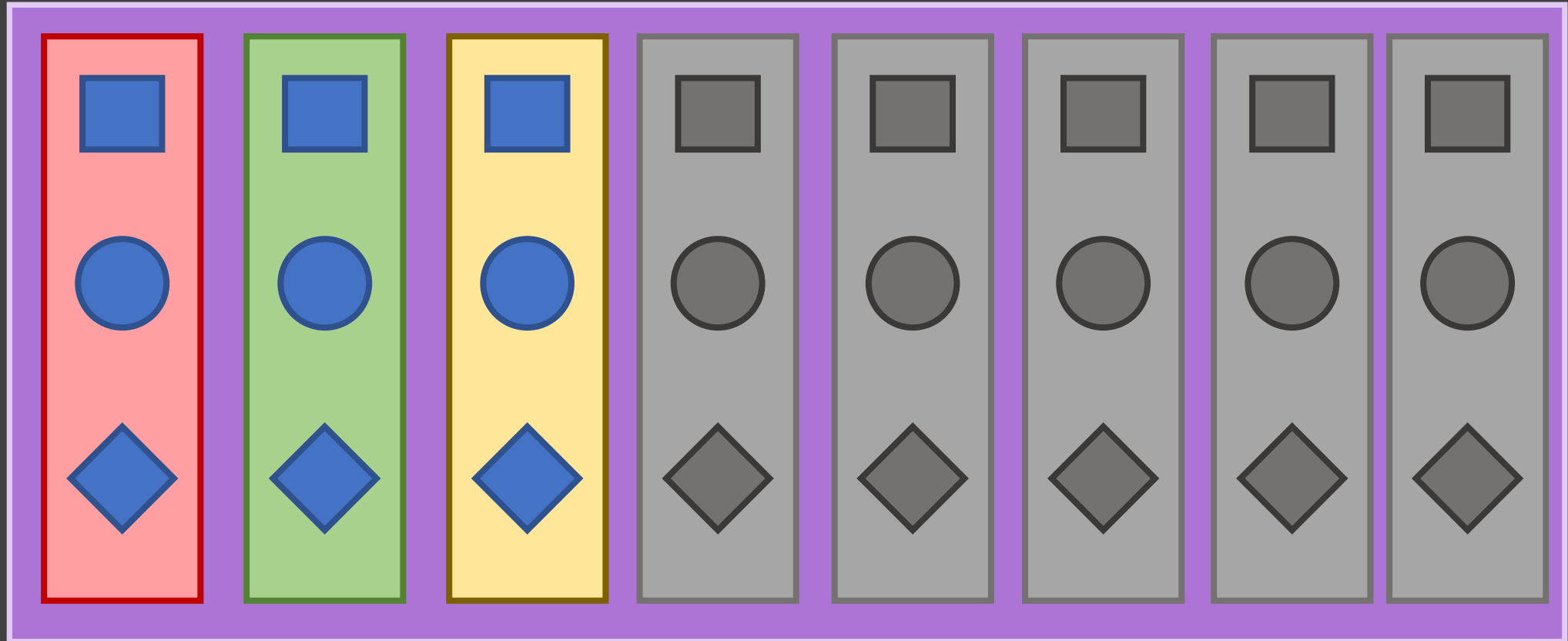
Vertical slicing



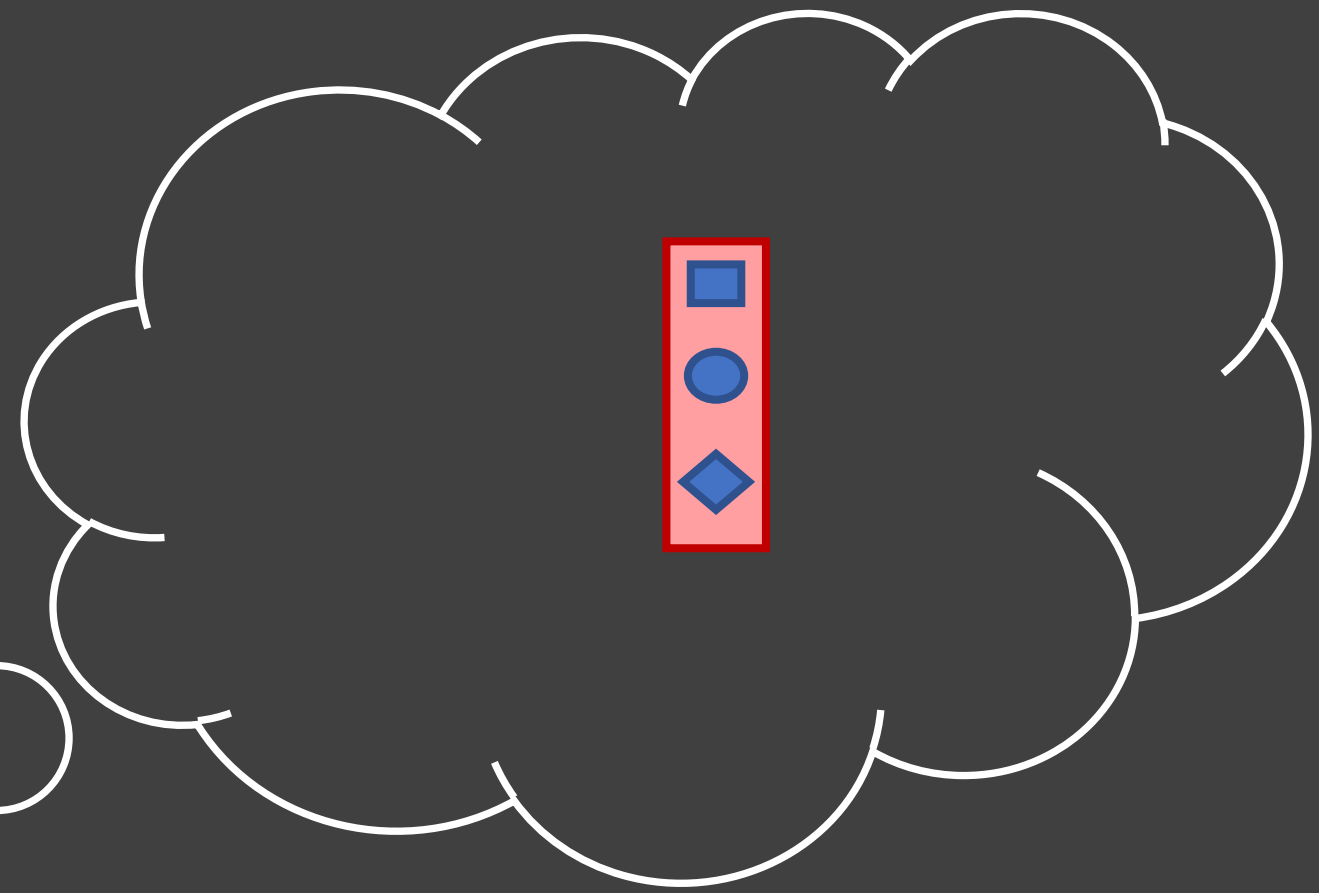
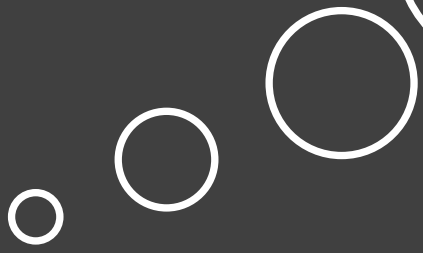
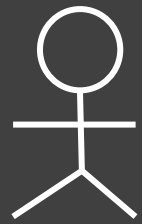
Vertical slicing



Vertical slicing



Cognitive load!

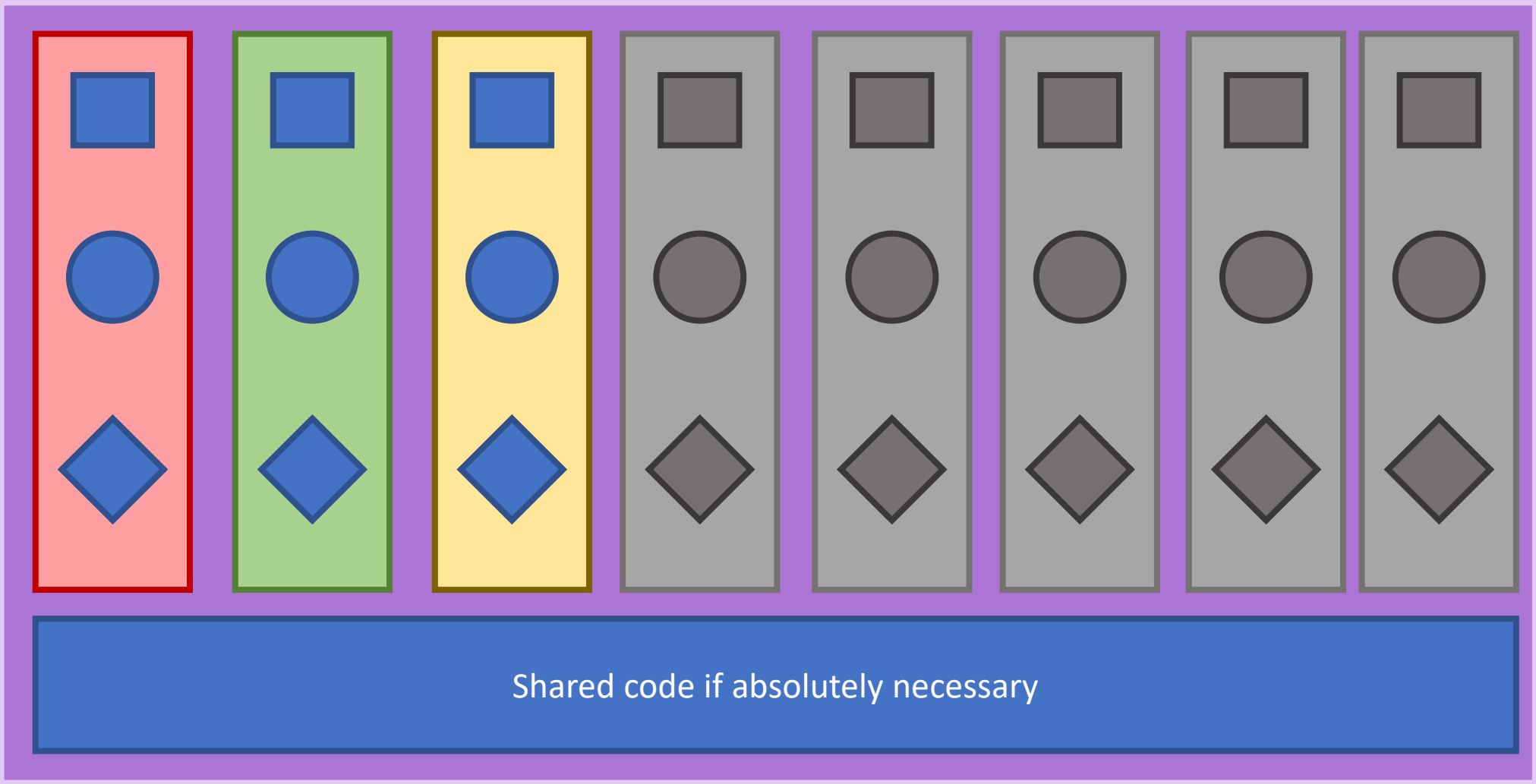


“One of the problems with this style of code... if you have orders, and customers, and you need to make a link between the two; where do you go to make that link?”

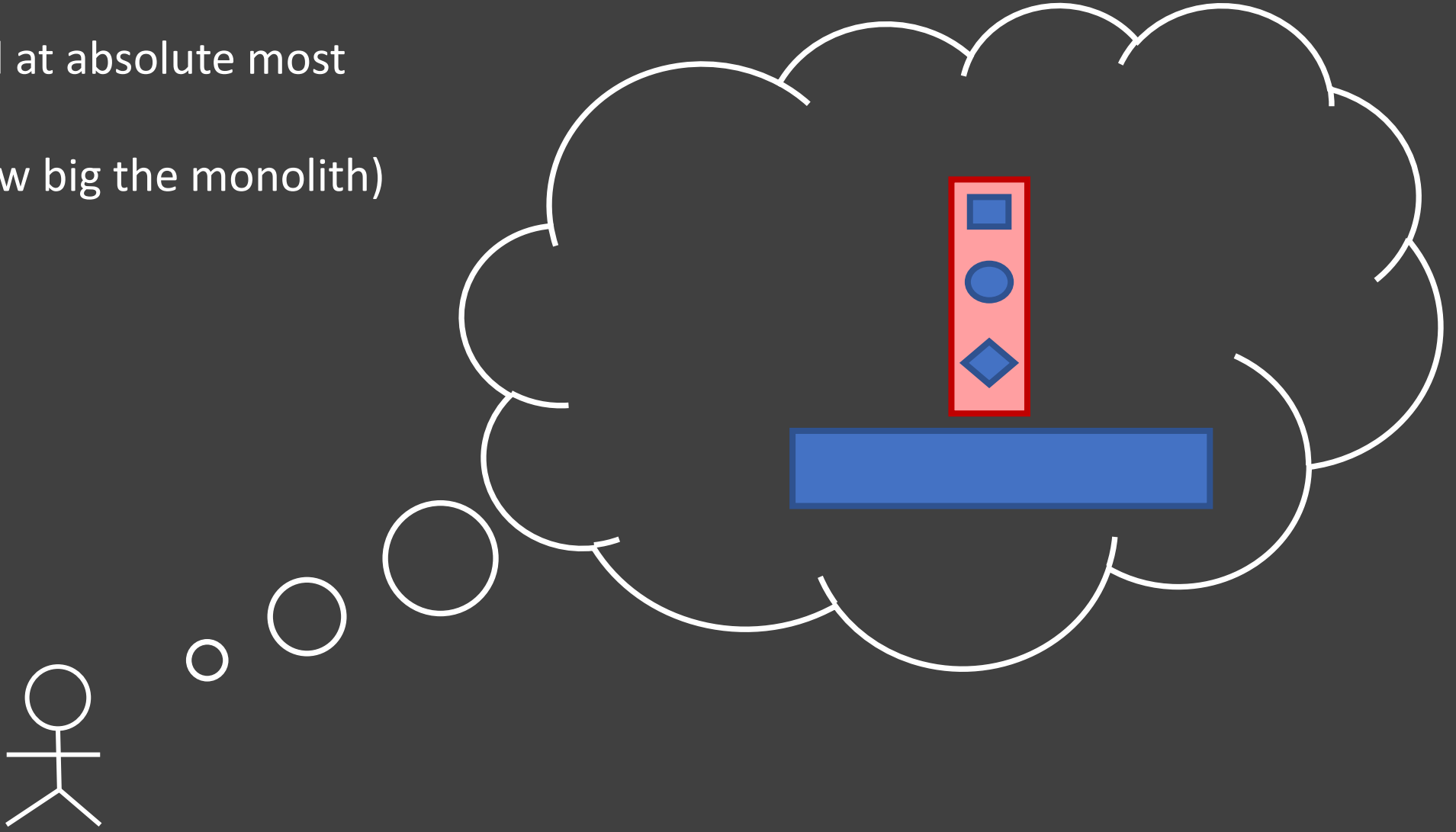


Simon Brown
“Modular Monoliths” (2018)

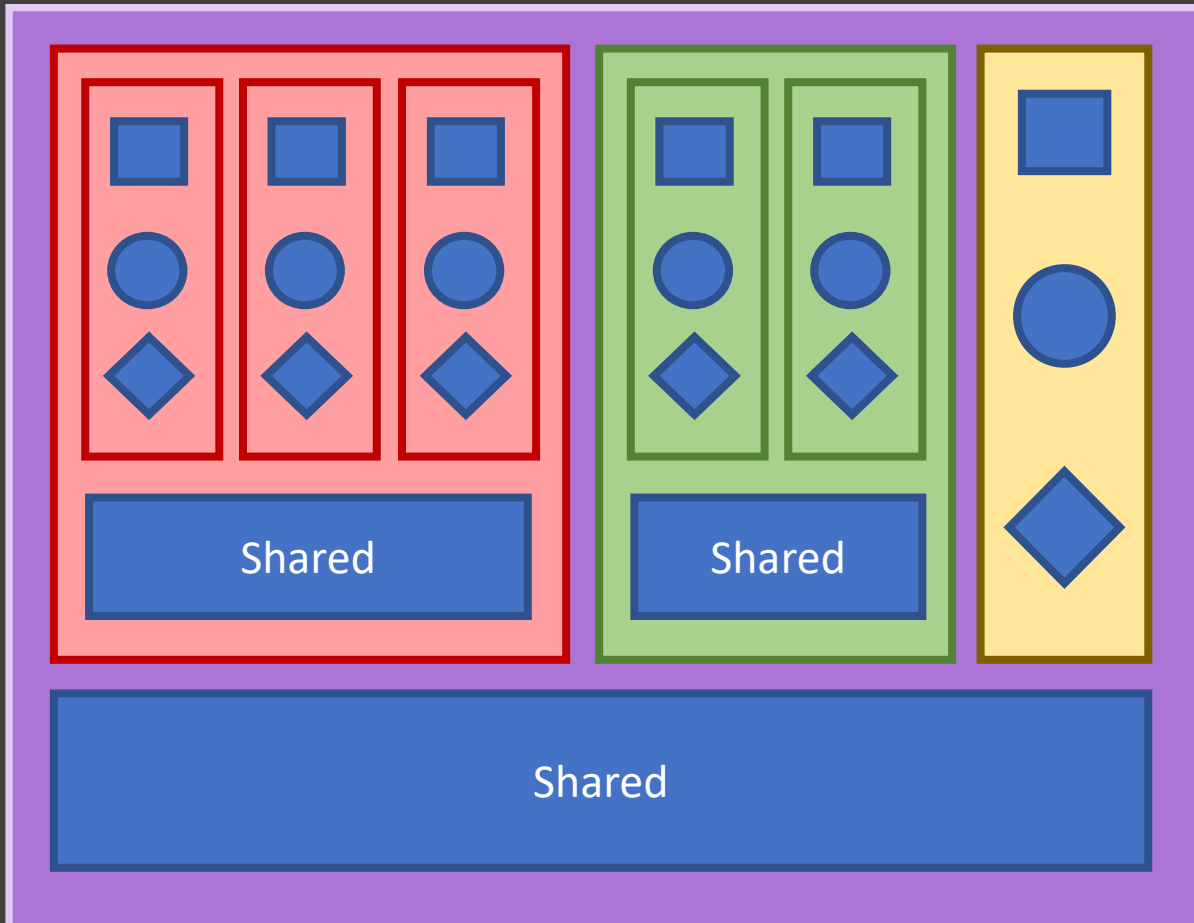
Vertical slicing



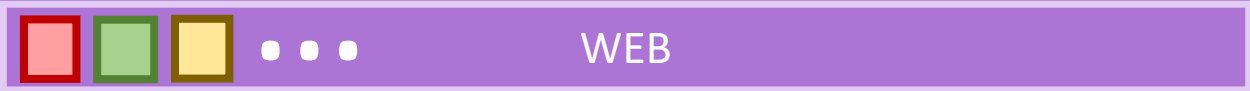
Cognitive load at absolute most
(no matter how big the monolith)



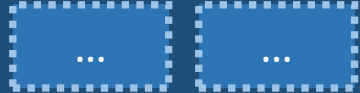
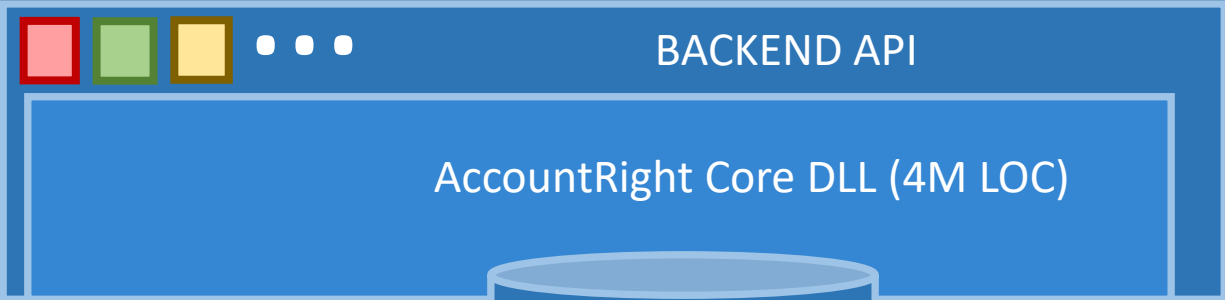
Nested vertical slicing



- Features
 - Accounts
 - Card
 - GeneralJournal
 - Delete
 - GetById
 - List
 - NextEventId
 - Patch
 - Mappings
 - GeneralJournalLinePatchProfile.cs
 - GeneralJournalPatchProfile.cs
 - GeneralJournalCreateController.cs
 - GeneralJournalLinePatch.cs
 - GeneralJournalPatch.cs
 - GeneralJournalPatchResult.cs
 - GeneralJournalPatchService.cs
 - GeneralJournalUpdateController.cs
 - TaxSettingService.cs
 - Shared
 - DTOs
 - Mappings
 - GeneralJournalConverter.cs
 - GSTReportingMethod.cs
 - PropertyObject.cs
 - TaxTransaction.cs
 - TaxTransactionComponent.cs
 - ValidationResult.cs



THE SEAM



Things working against us

- Huge amount of work, 100s of features
- AccountRight wasn't designed for this!
- Hard hard hard deadline
- We'd never delivered that fast before
- Cannot release incrementally
- What if the new callstack is too slow?
- What if the market gets spooked?
- What if customers don't like it?

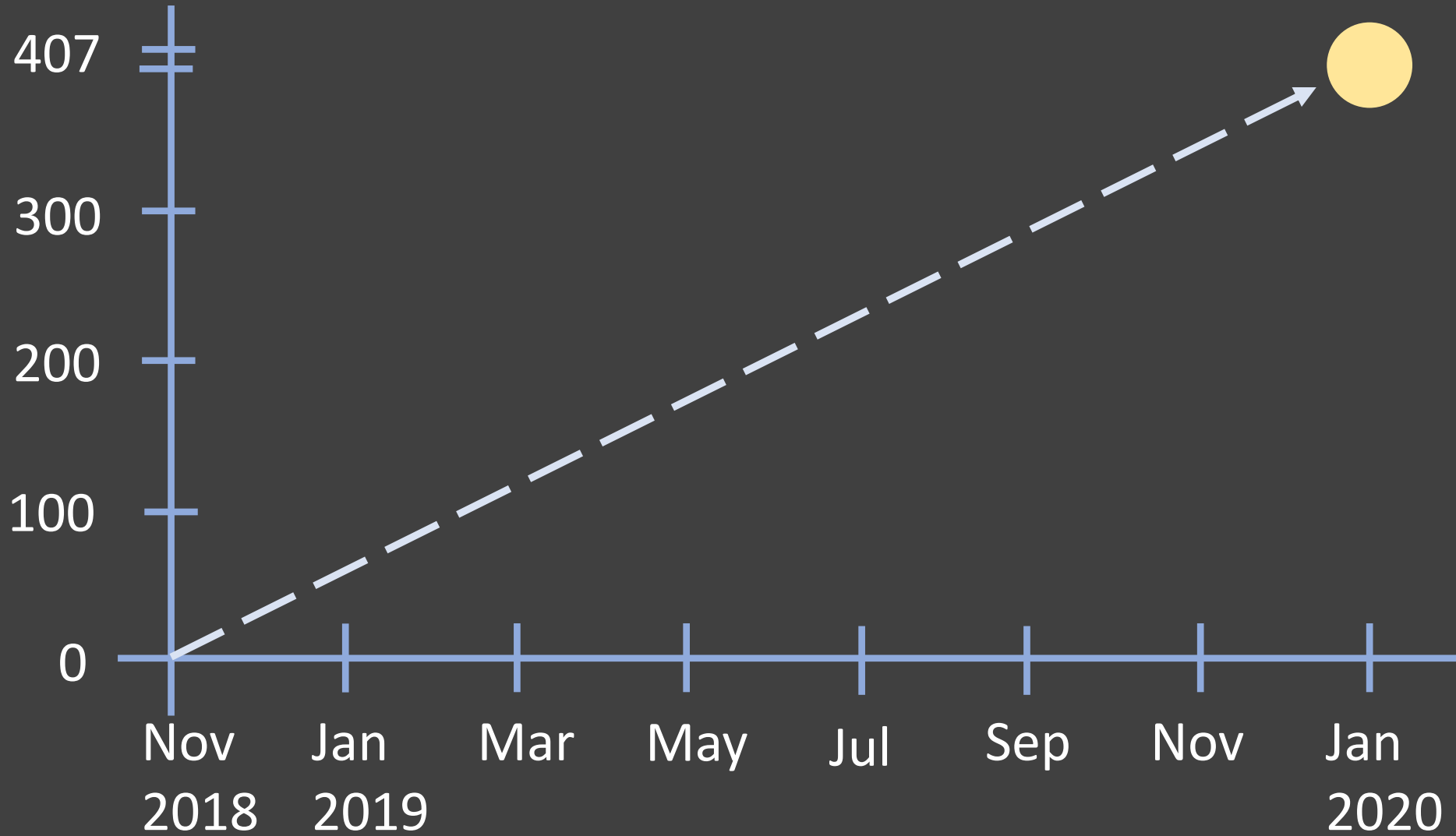
Things in our favour

- We could reuse existing functionality
- AccountRight was surprisingly tidy
- We basically knew the full scope already

III. The project

Features

DESIRED



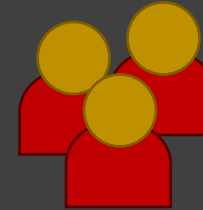
Team Structure



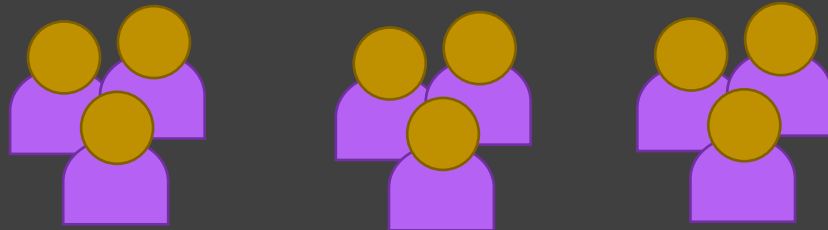
Front end



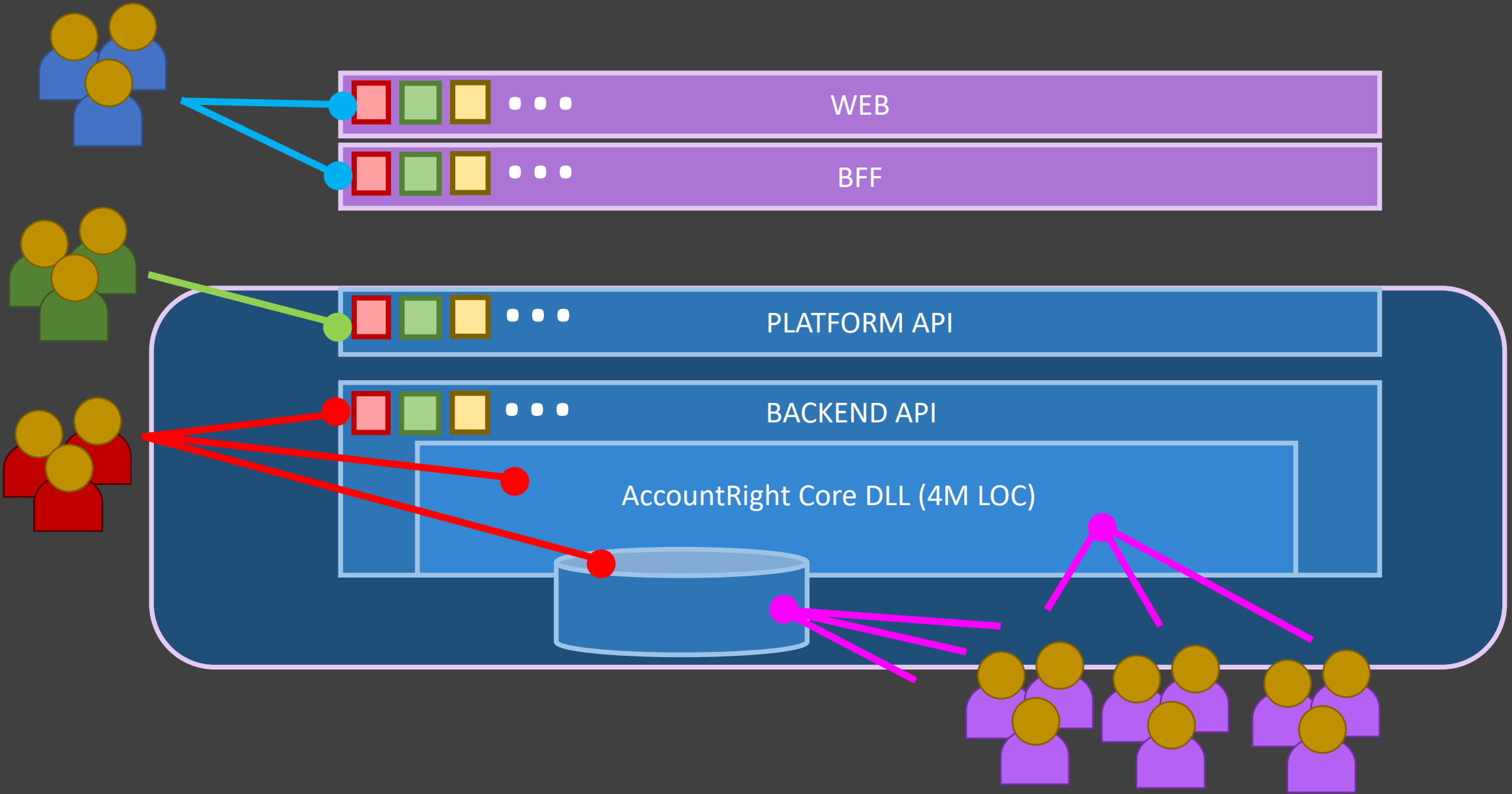
Analysis & API



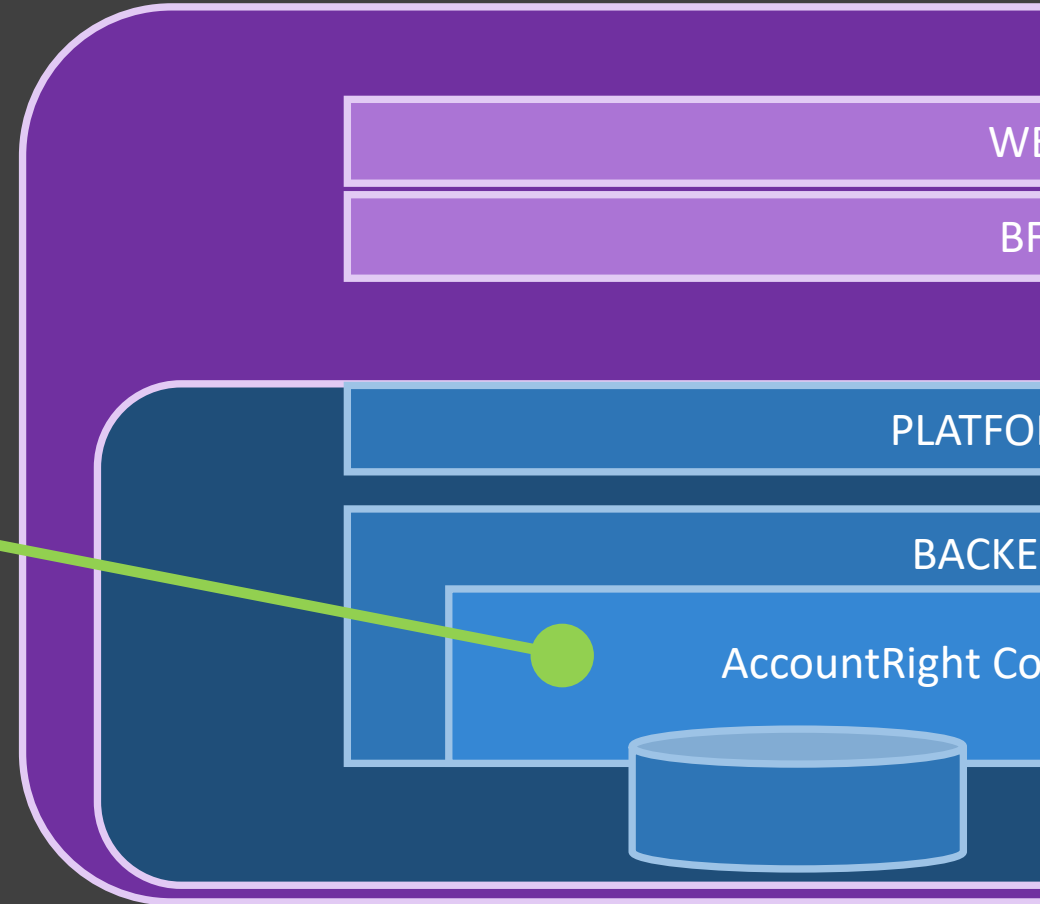
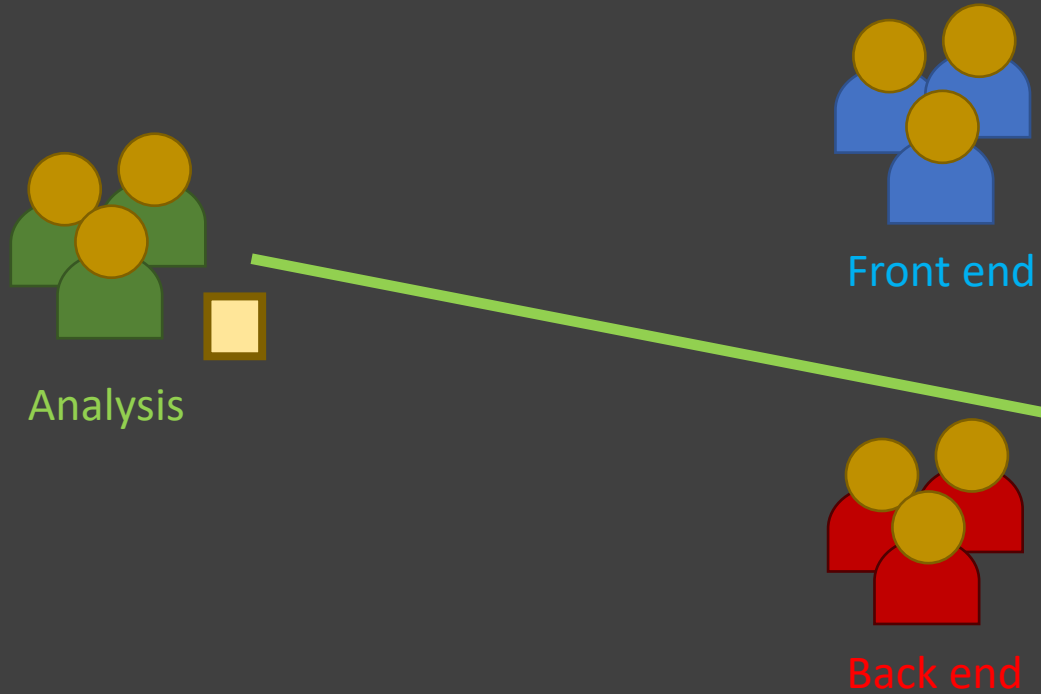
Backend



AccountRight Gaps

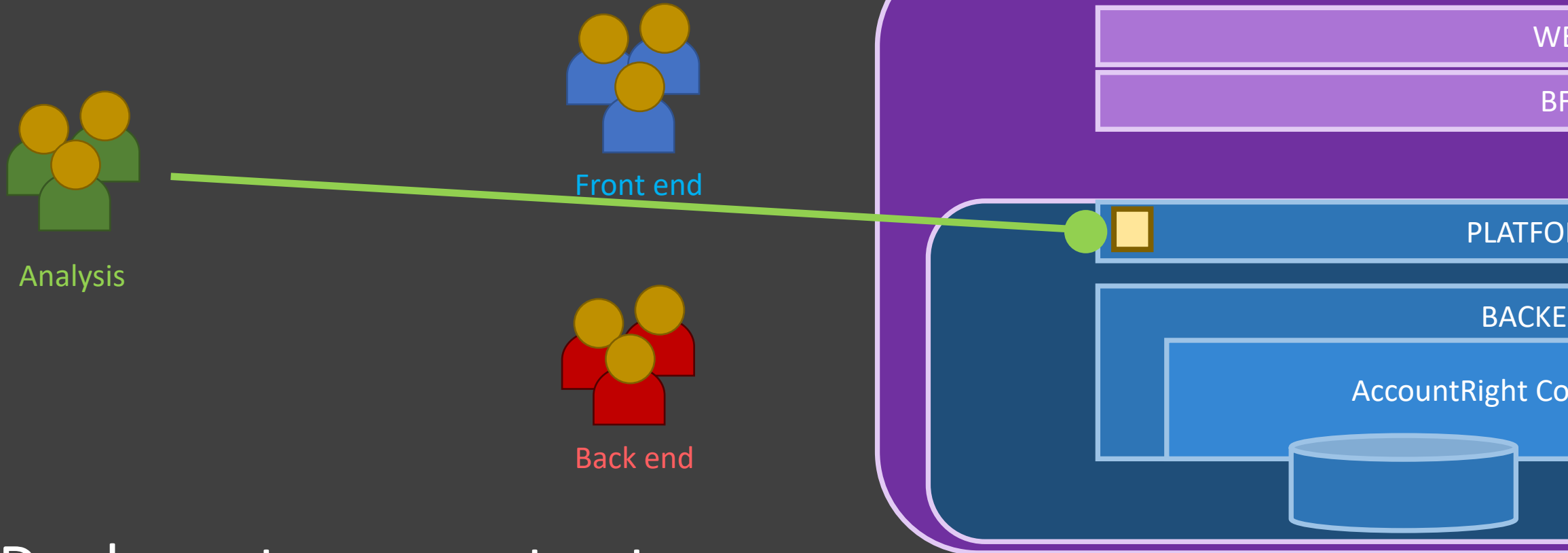


System of work (take 1)



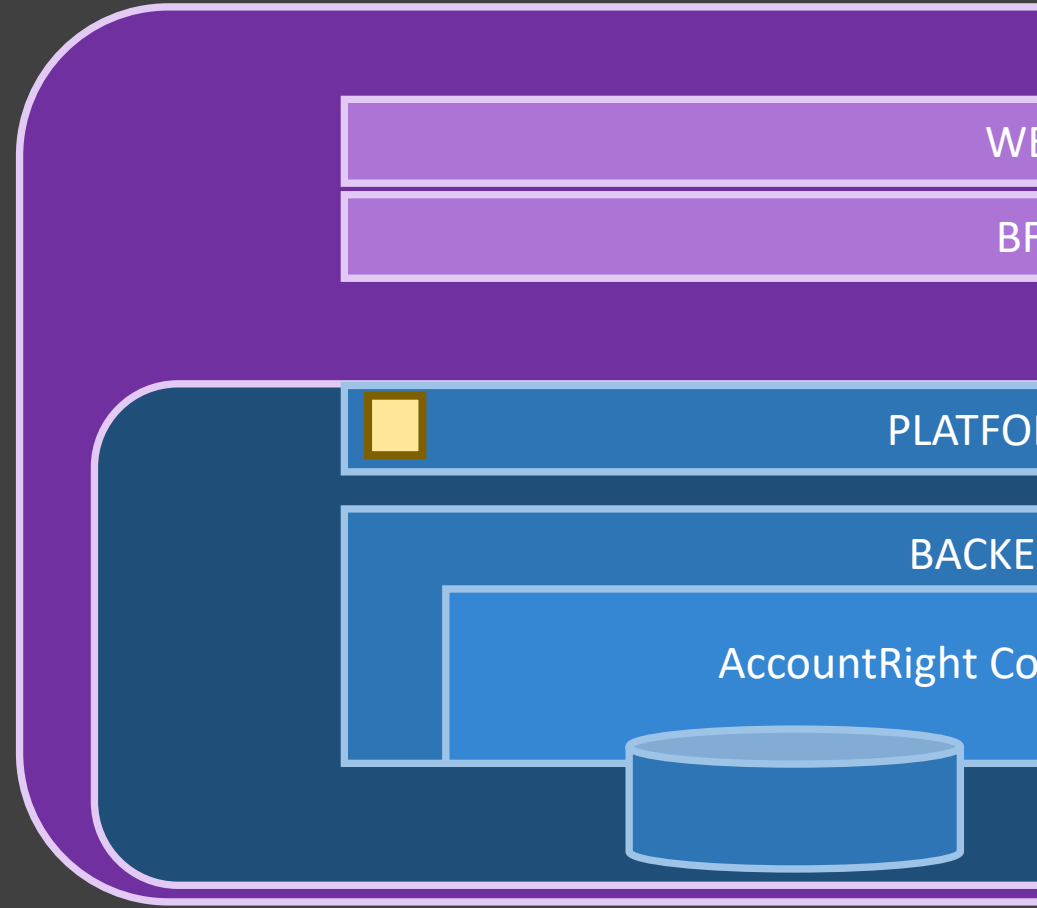
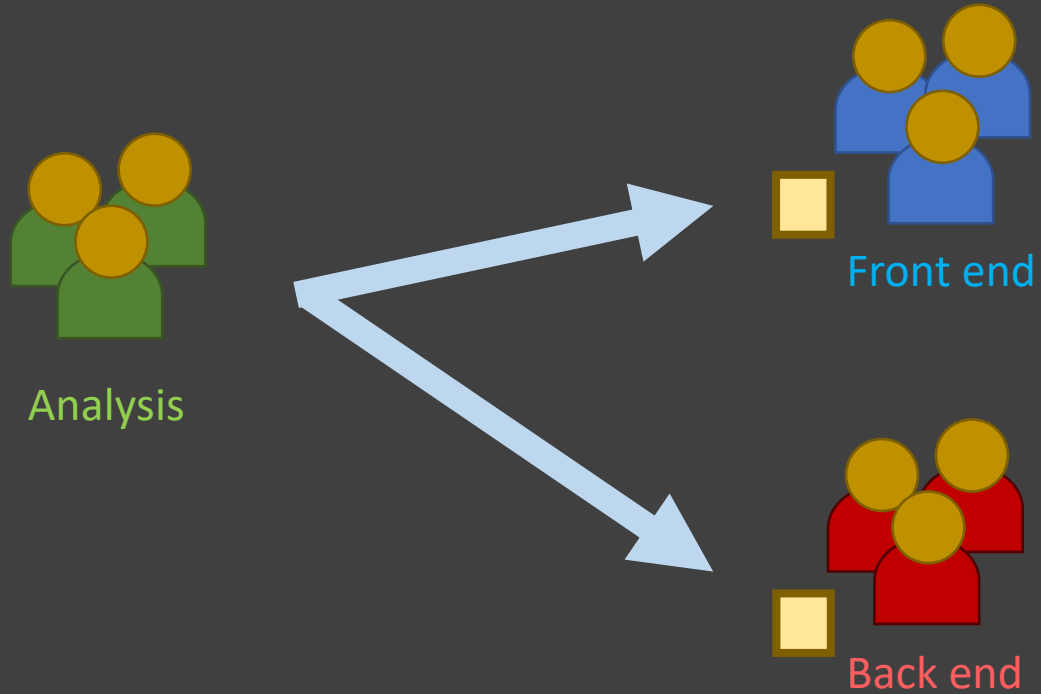
1. Analyse existing AccountRight functionality

System of work (take 1)



2. Deploy gateway contracts

System of work (take 1)



3. Handover

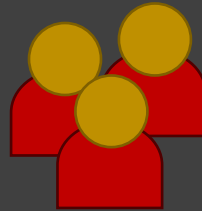
System of work (take 1)



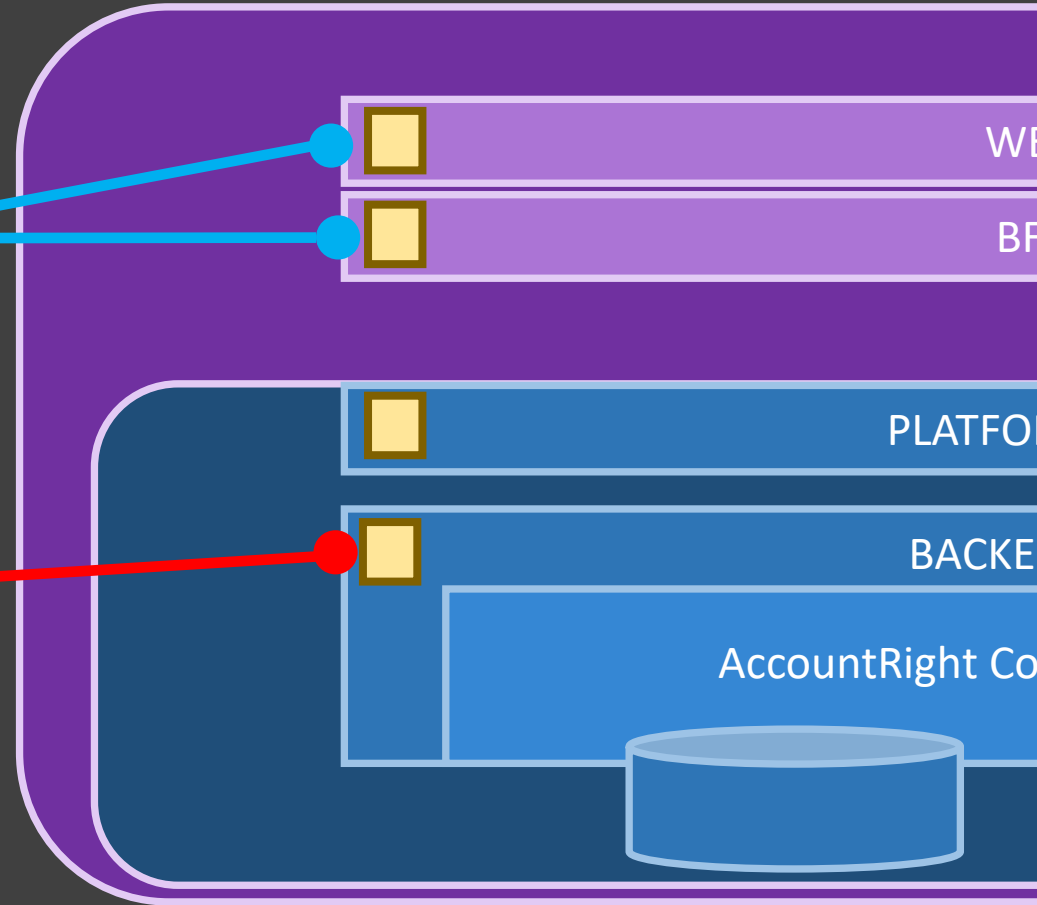
Analysis



Front end



Back end



4. Development

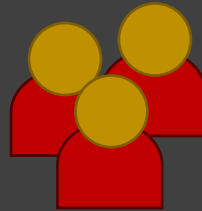
System of work (take 1)



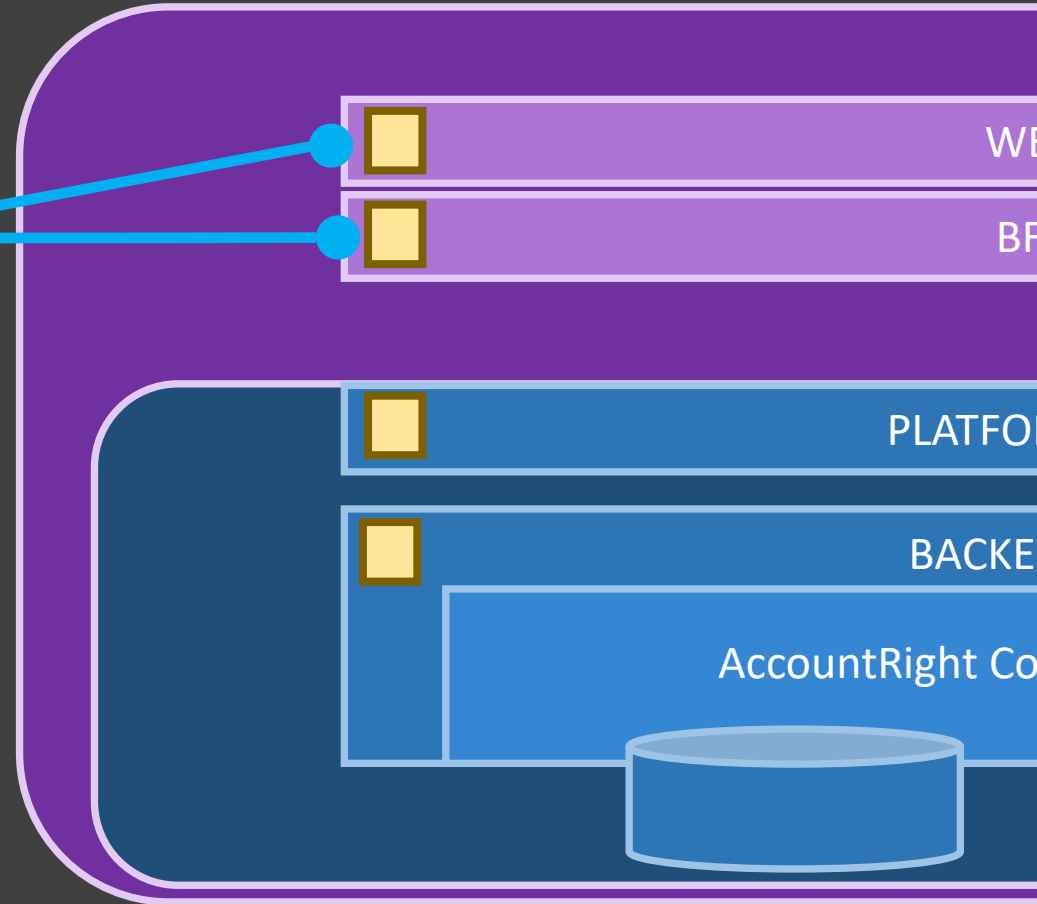
Analysis



Front end

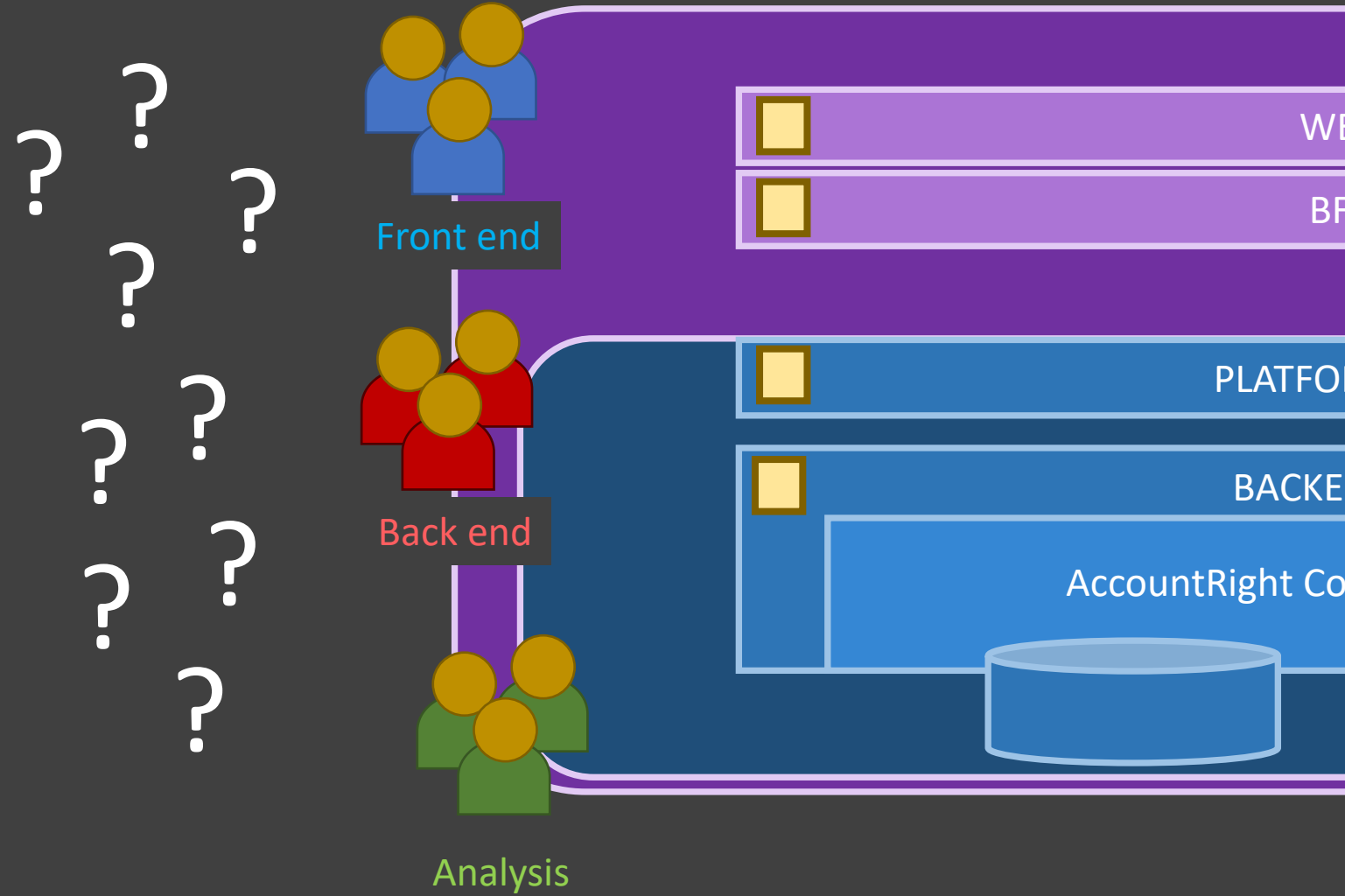


Back end



5. One or the other done first

System of work (take 1)



5. Both done

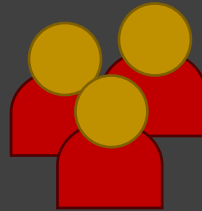
System of work (take 1)



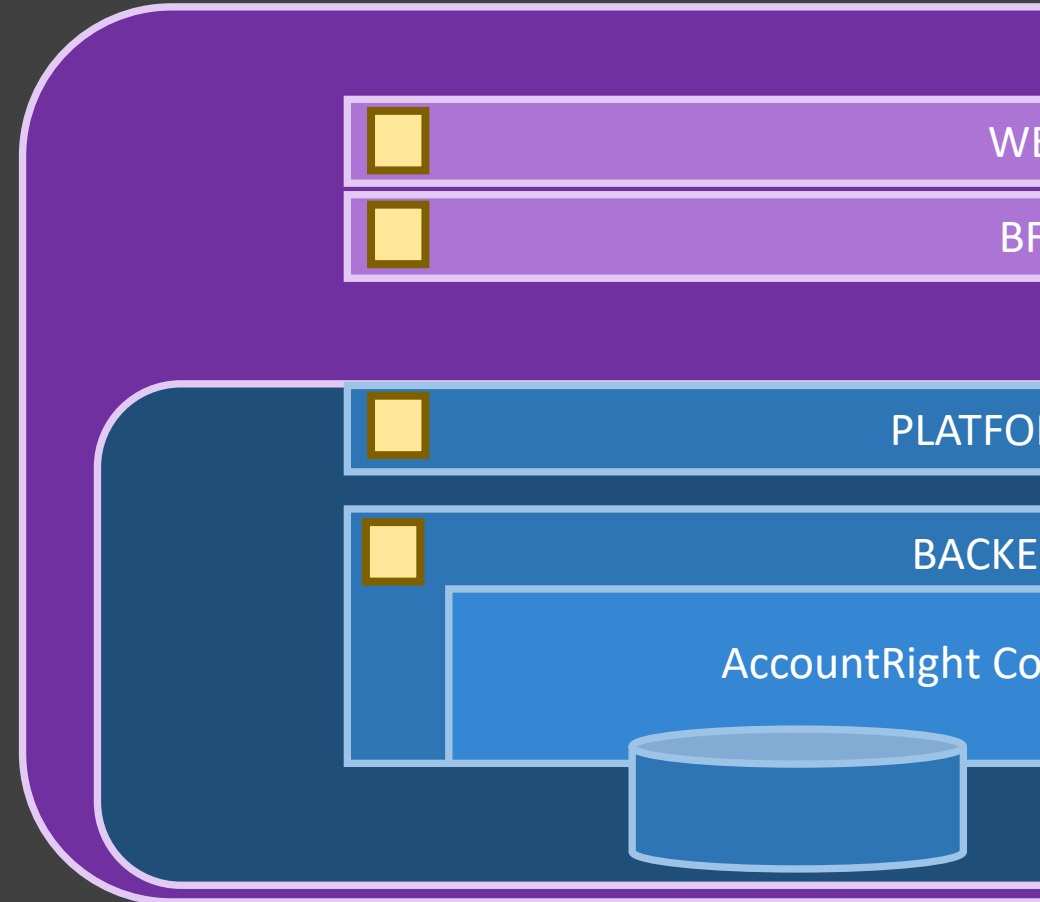
Analysis



Front end



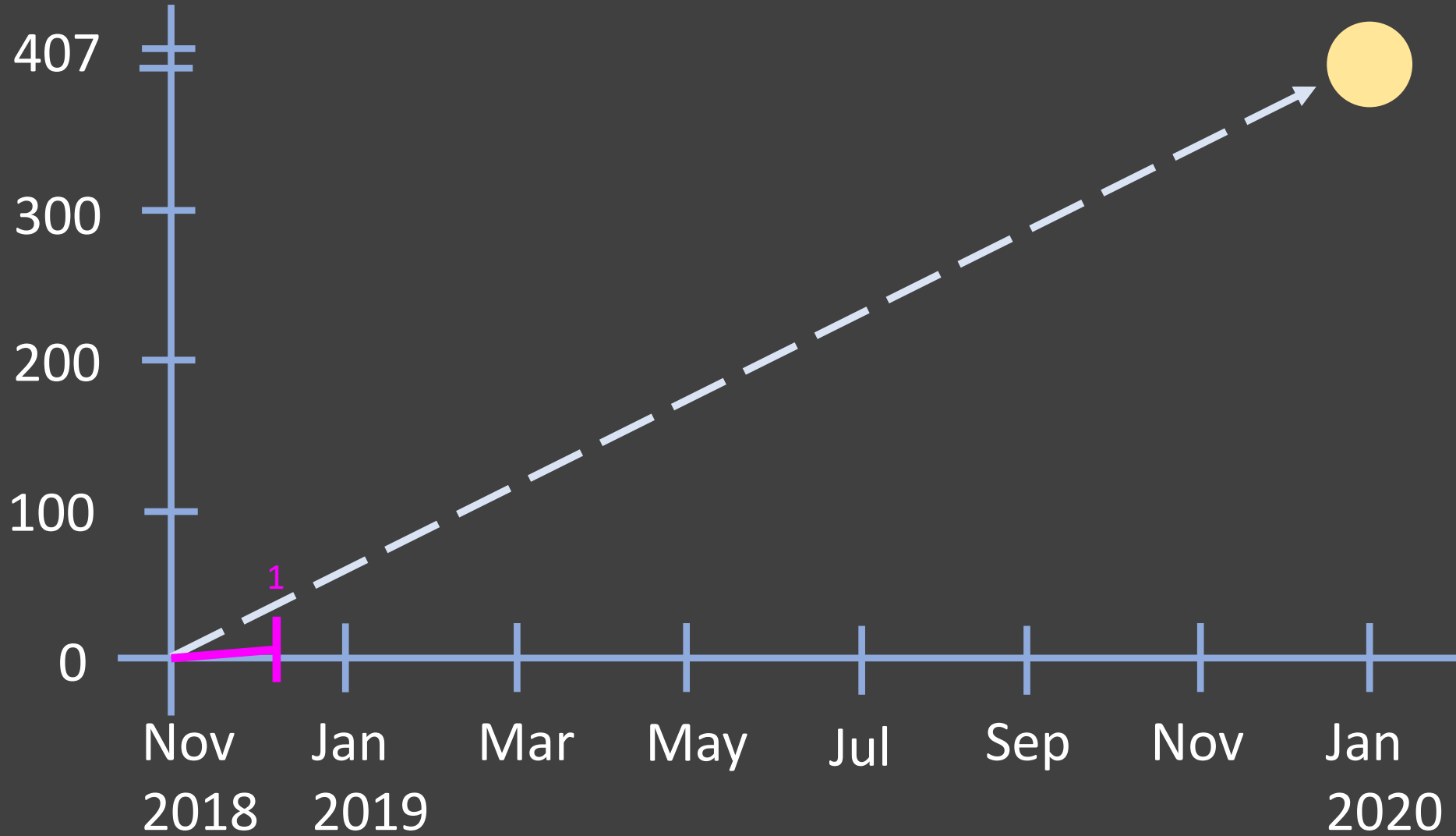
Back end



6. Done done (?)

Features

DESIRED

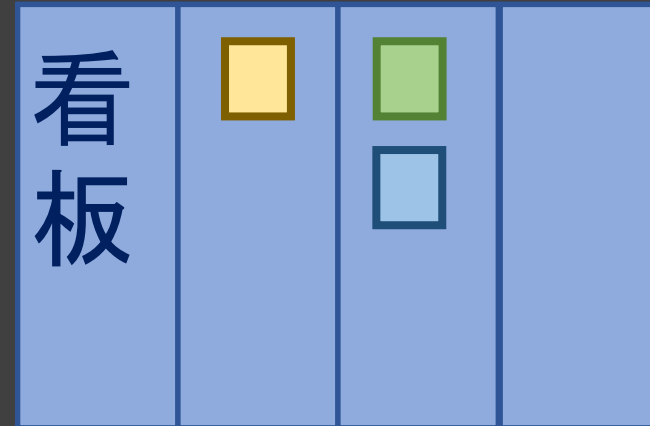


System of work (take 2)

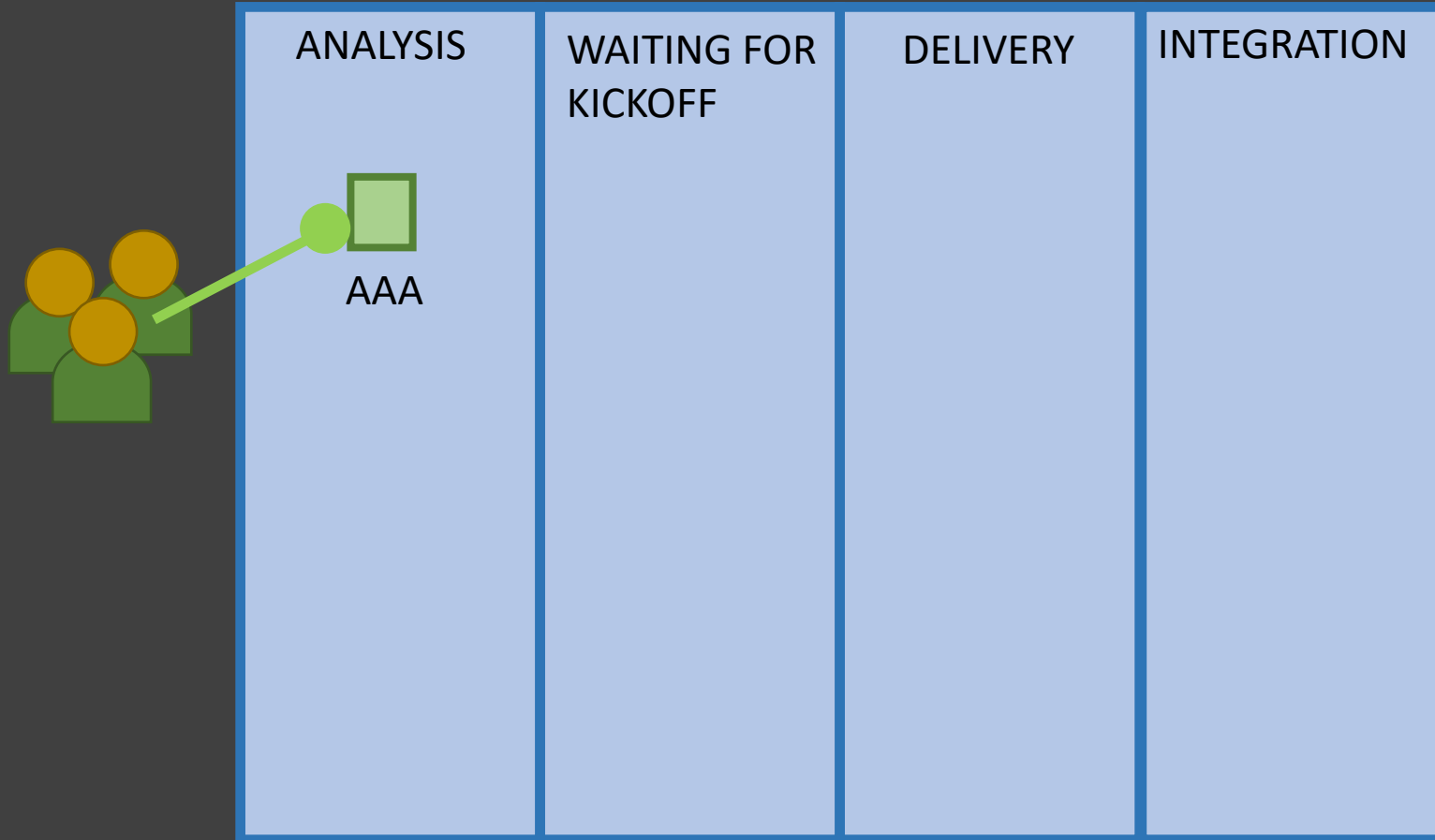
Whole tribe with end-to-end focus



Kanban & data

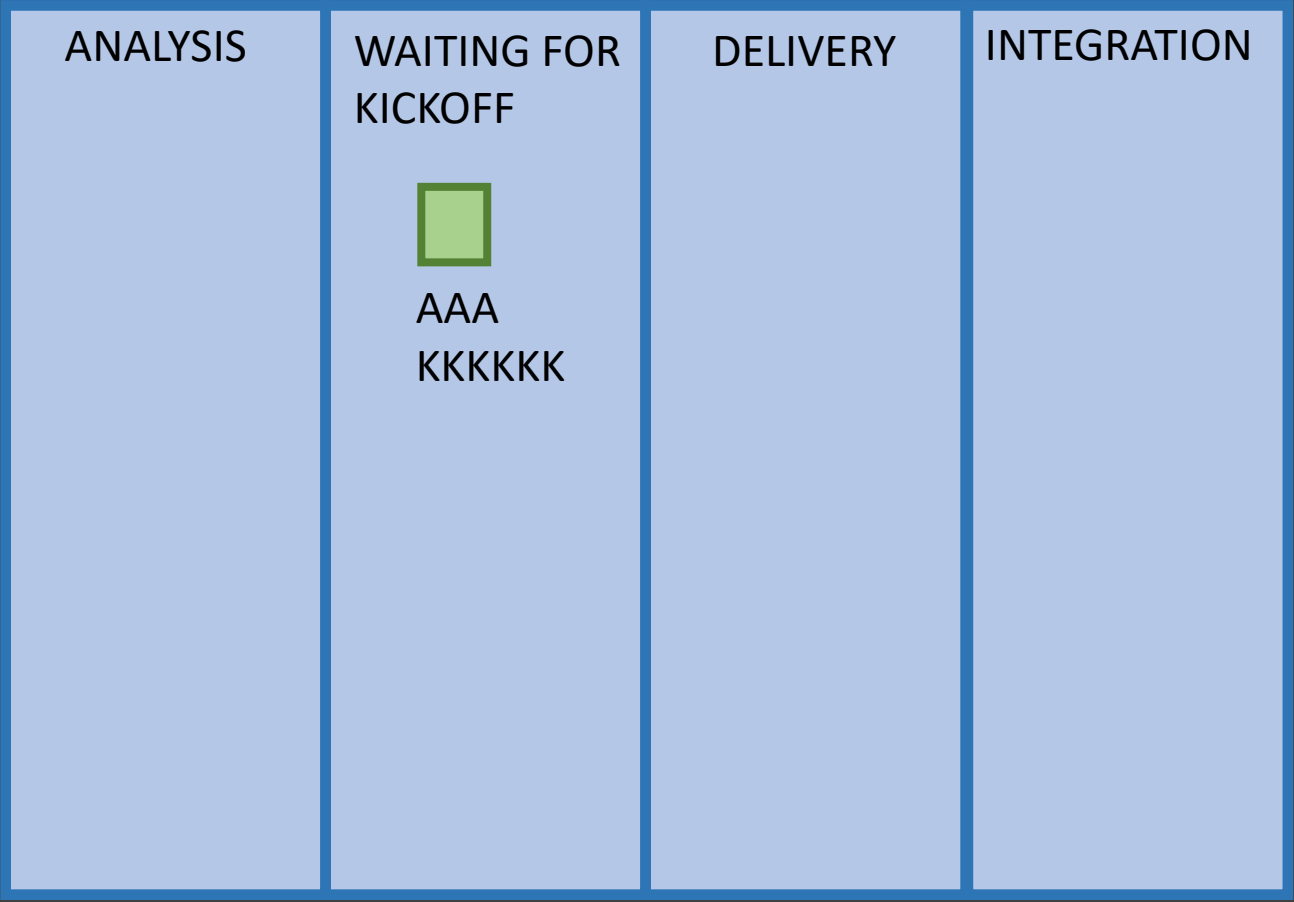
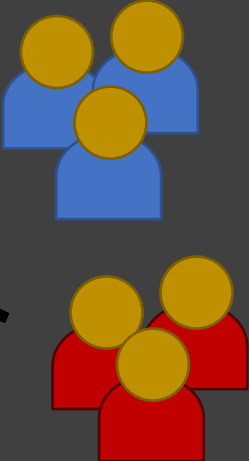


System of work (take 2)

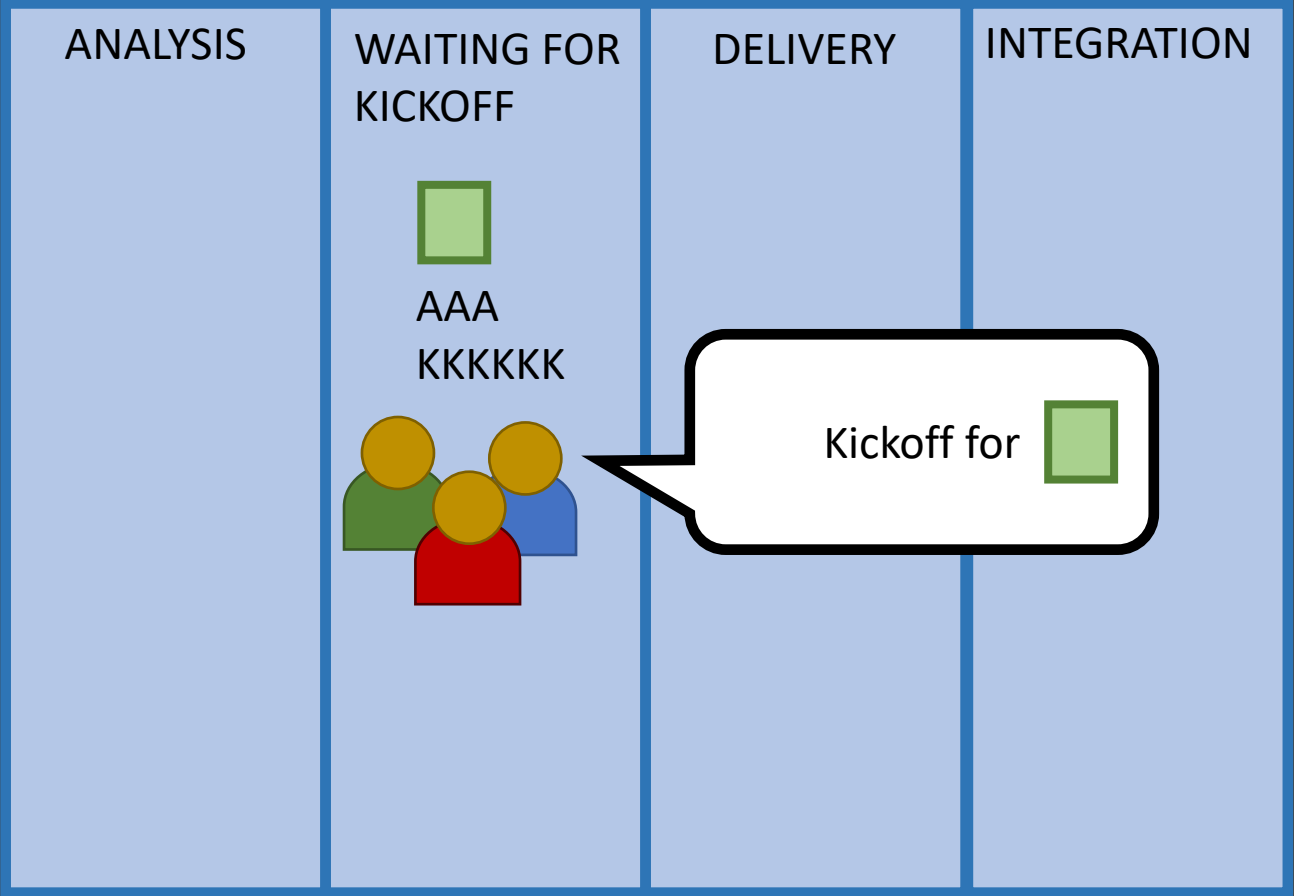



System of work (take 2)

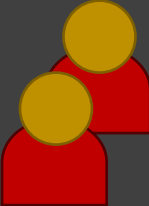
We're ready!



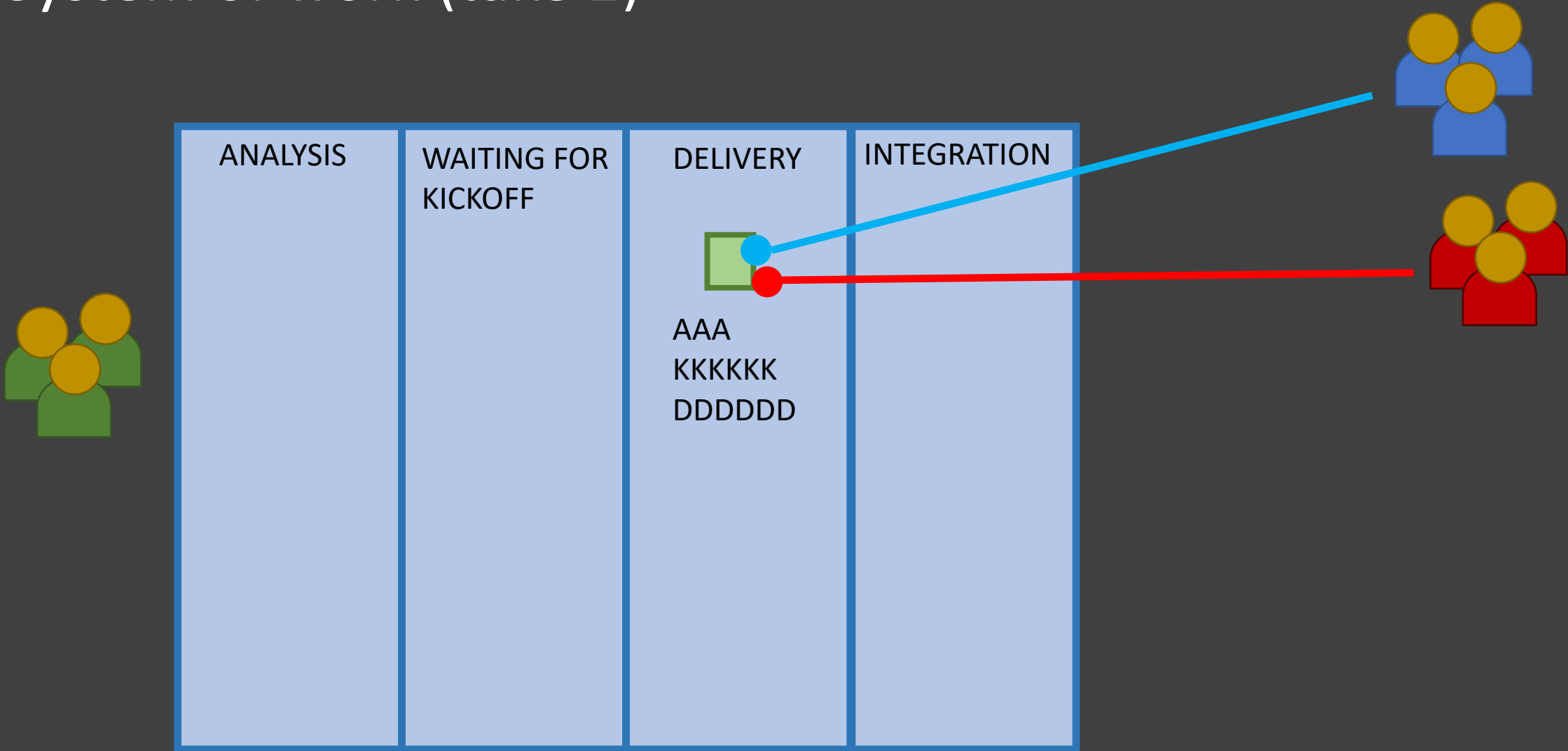
System of work (take 2)



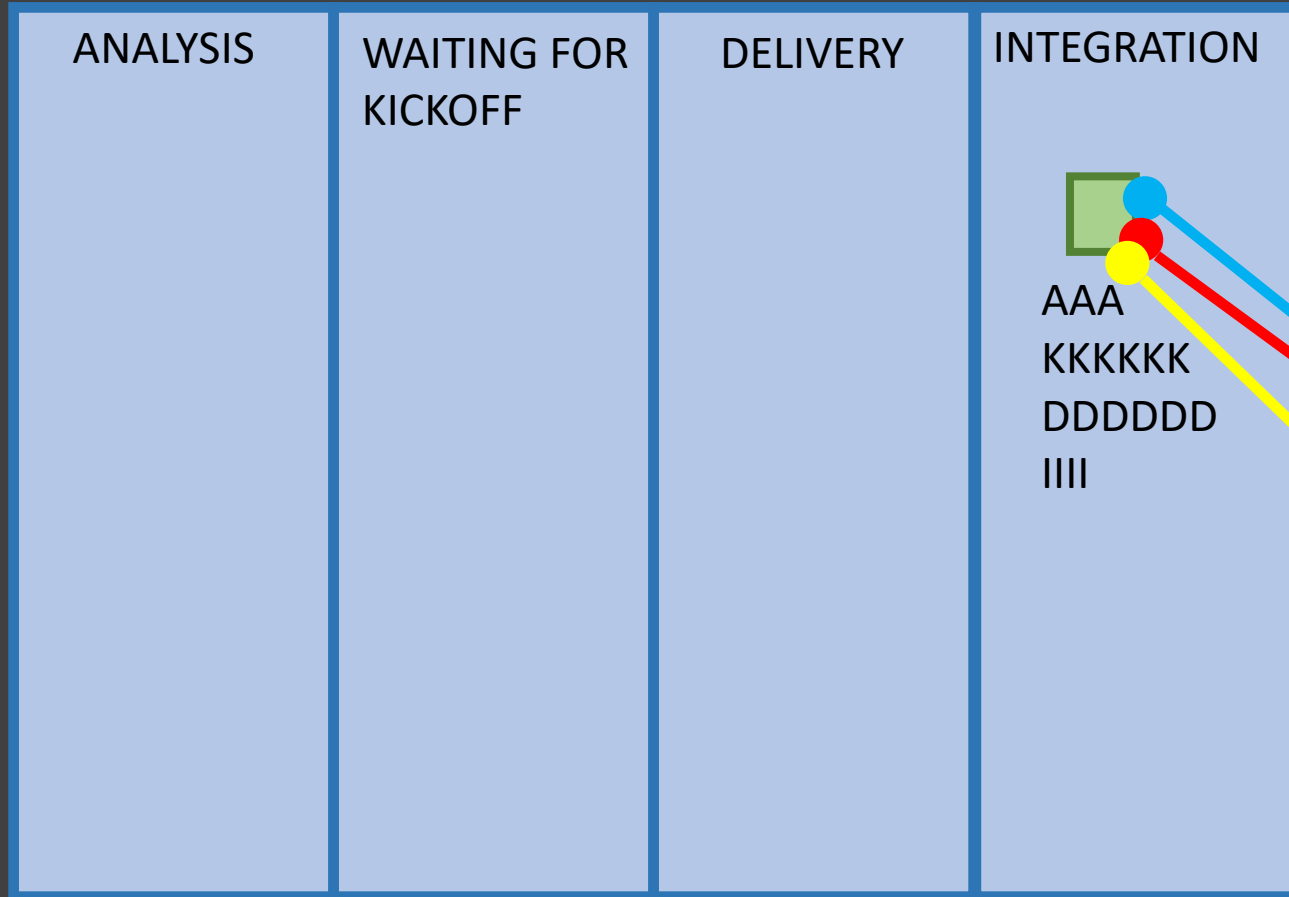
Kickoff for 



System of work (take 2)



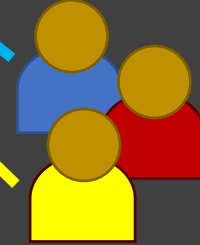
System of work (take 2)



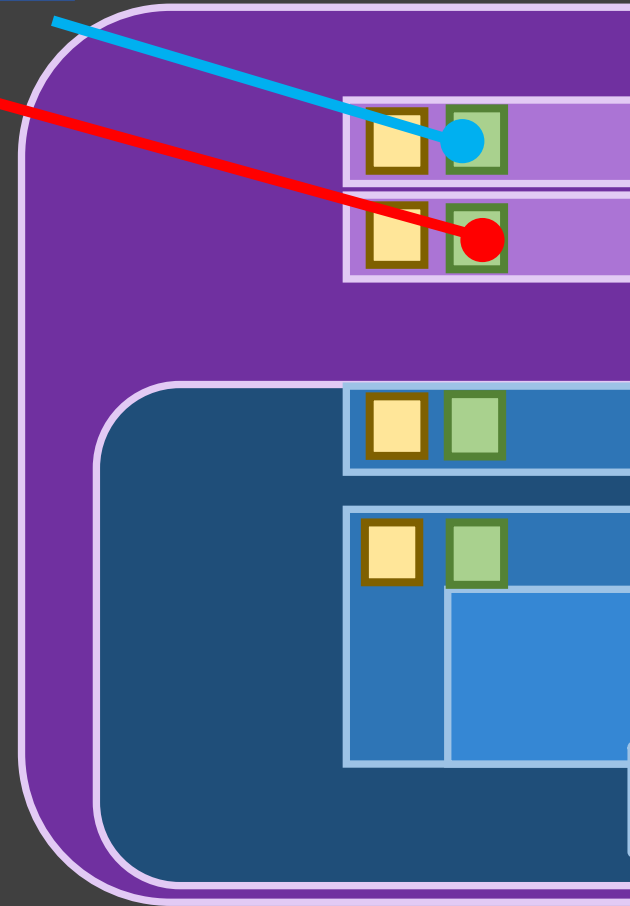
Original pair
(Rework if necessary)



Different pair



QA



System of work (take 2)



Cycle time
= 3 + 6 + 6 + 4
= 19 days end-to-end

Data driven process



Cycle time

“Is our delivery system healthy?”

“Where are the bottlenecks?”



IMPROVES

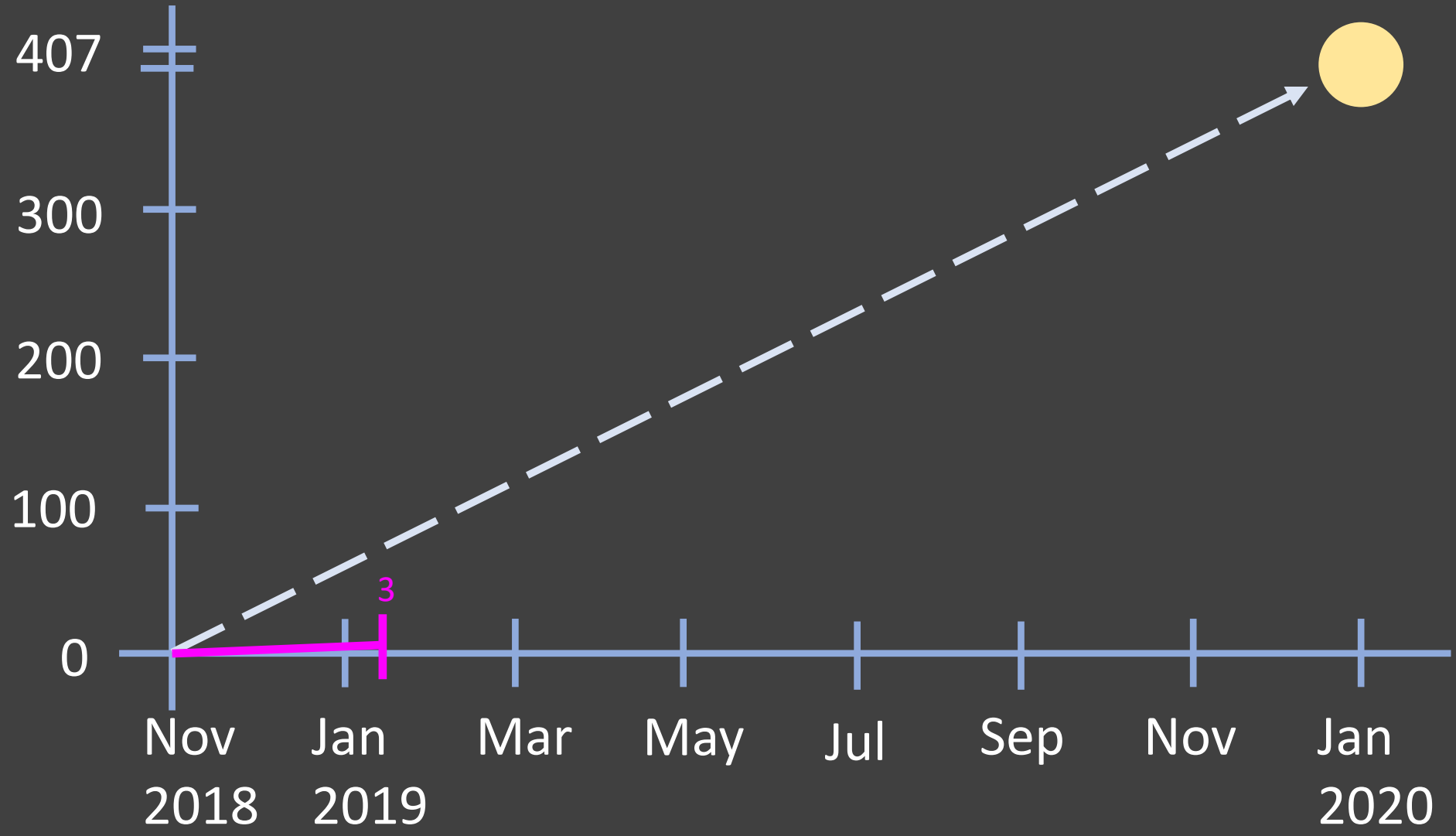


Throughput

“Are we going to succeed?”

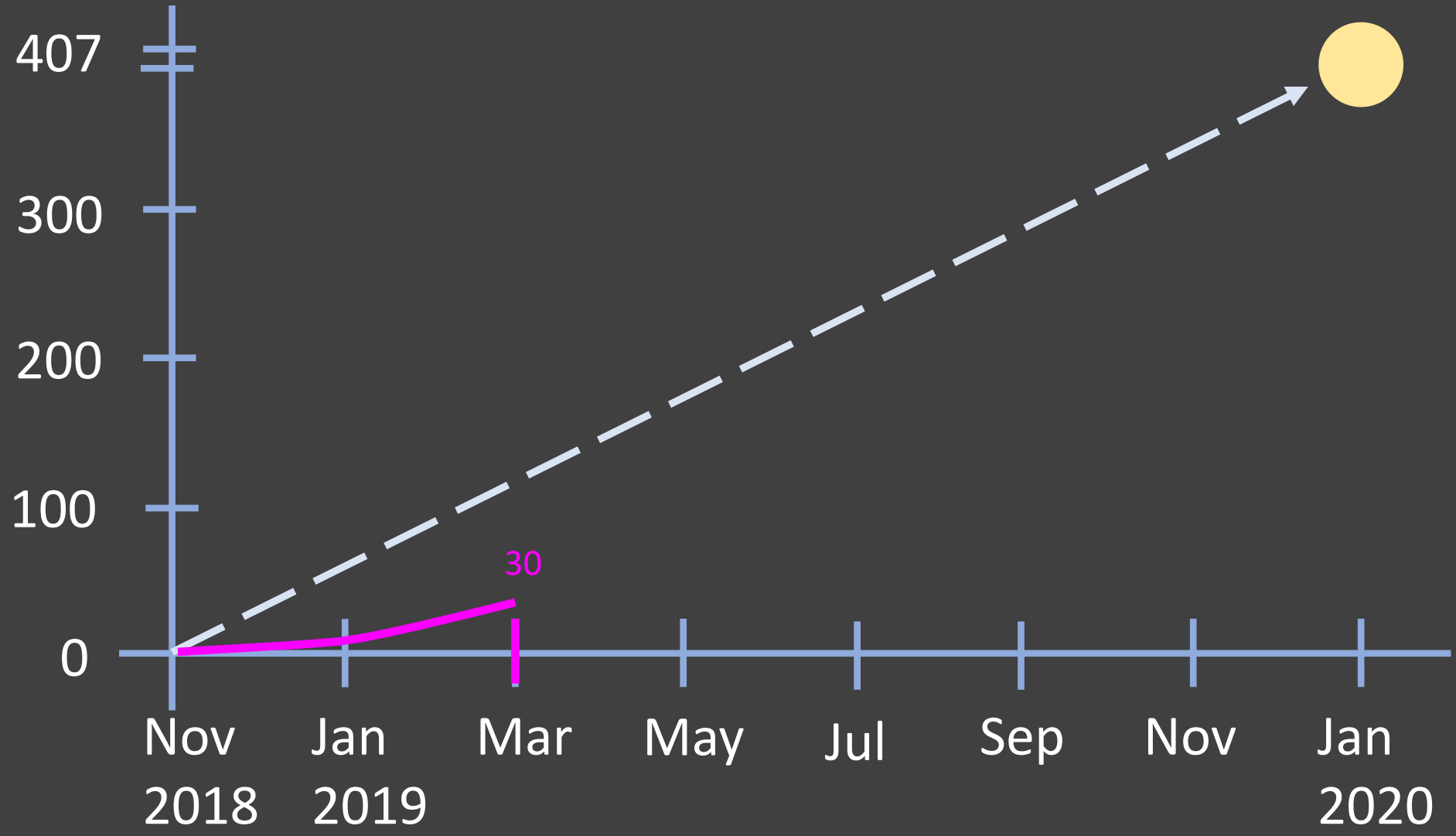
Features

DESIRED

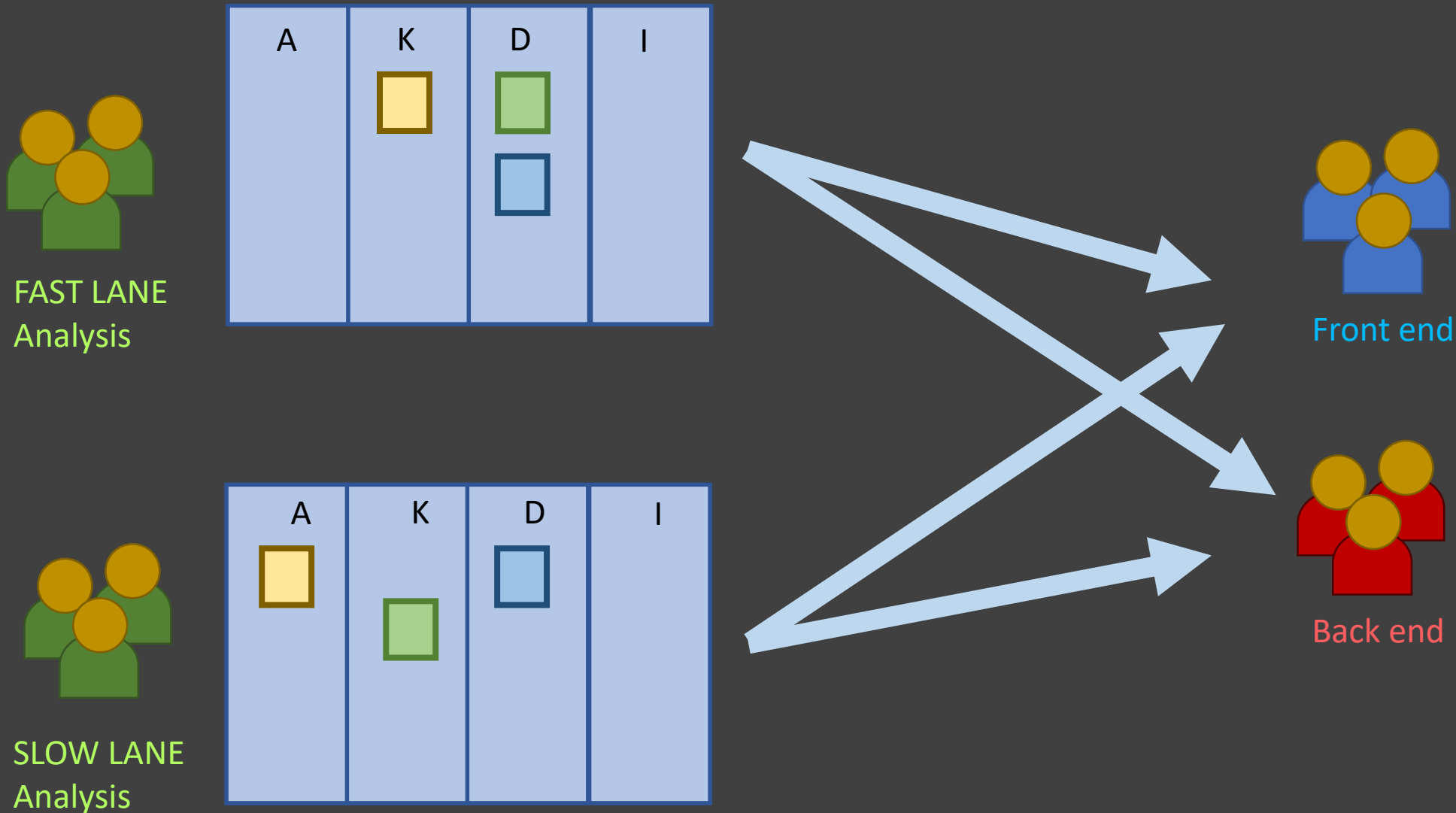


Features

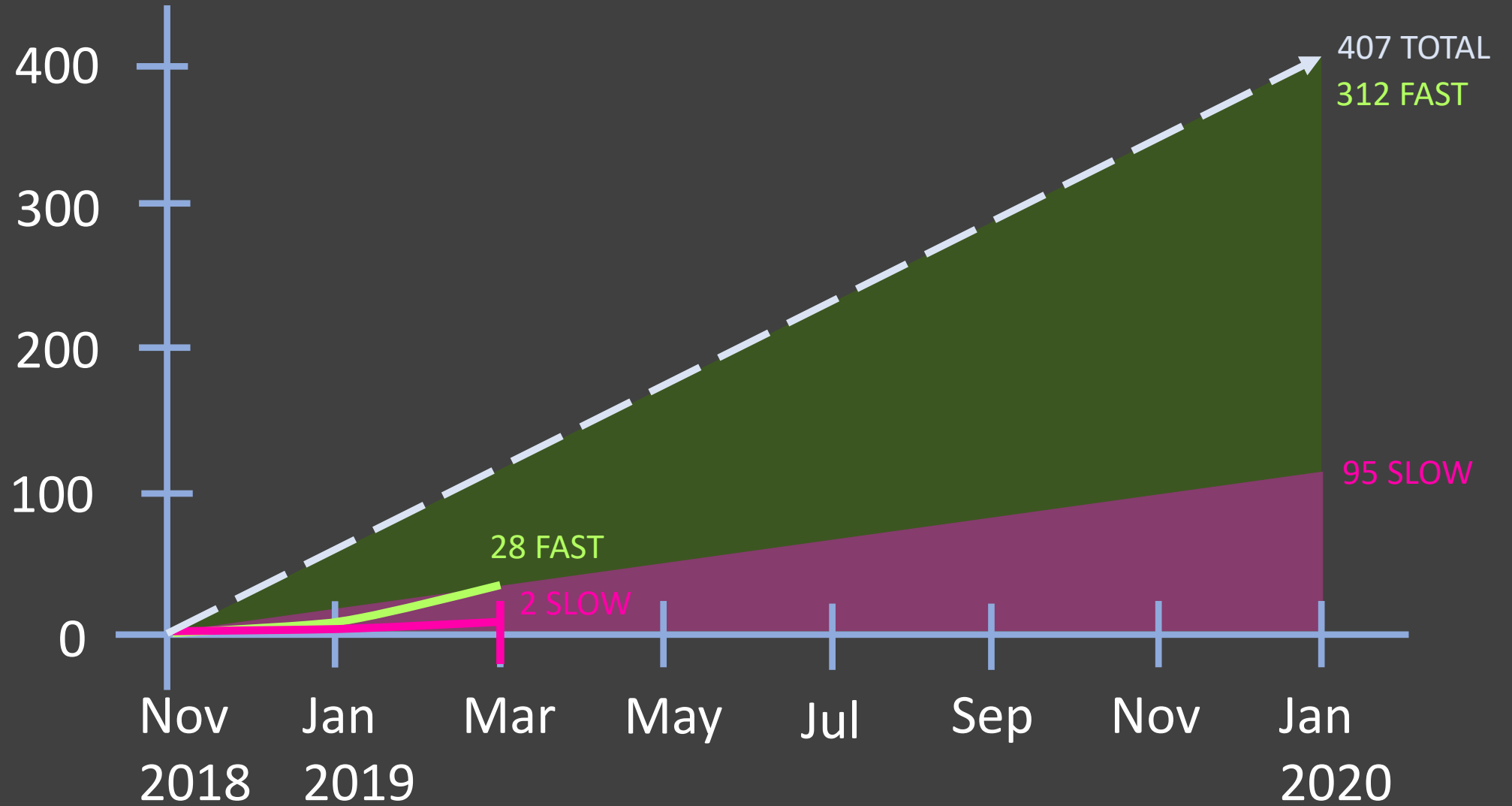
DESIRED



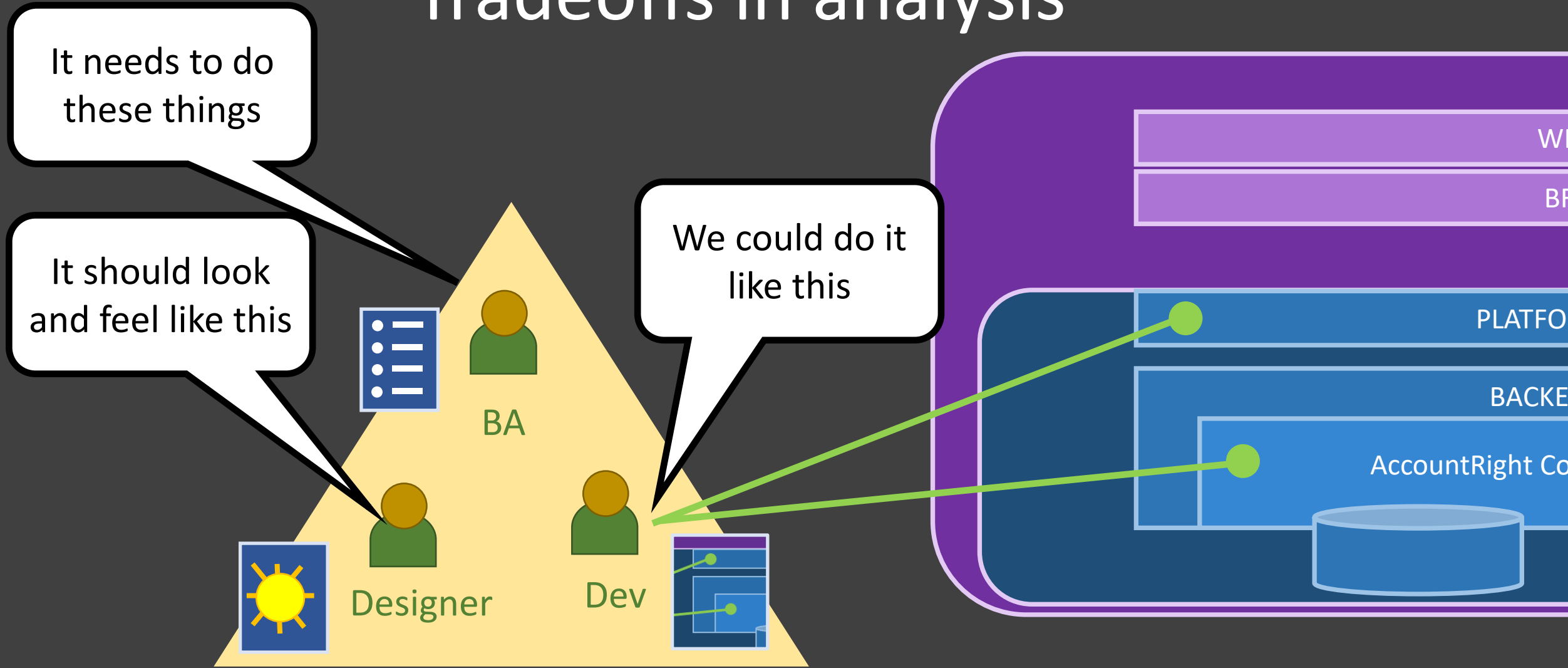
System of work (take 3) – Fast lane, slow lane



Features



Tradeoffs in analysis



Analysis of one feature

Tradeoffs in analysis

Urrrgh.

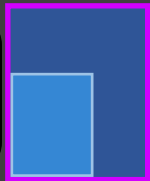
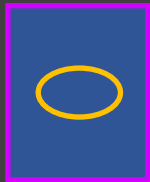


Designer

So pretty!



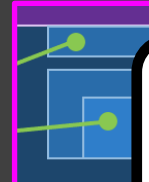
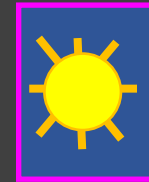
Designer



So simple!



Dev



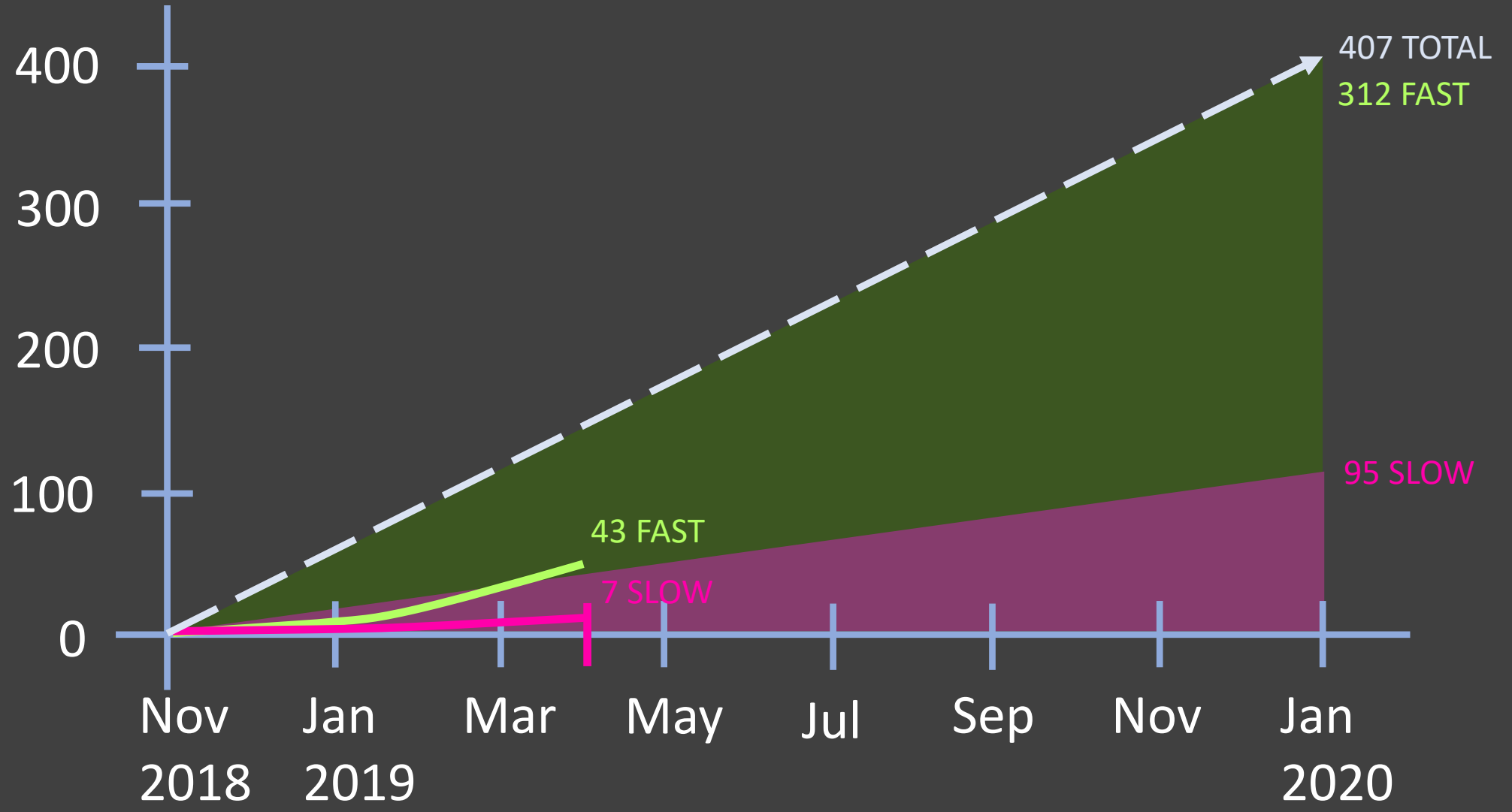
So complicated.



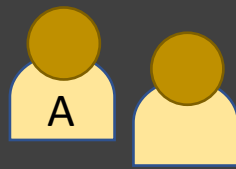
Dev



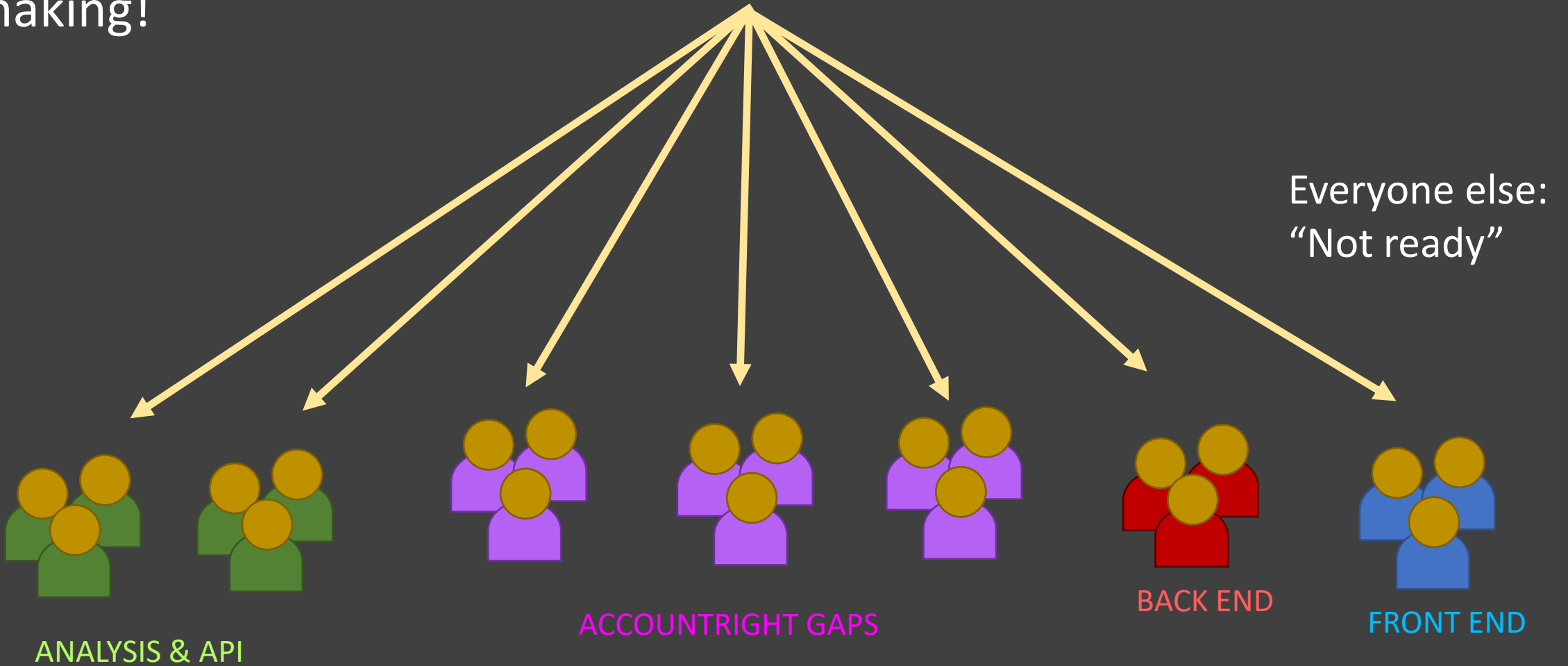
Features



Unsustainable top-heavy tech decision making!



ARCHITECT / PRINCIPAL DEV

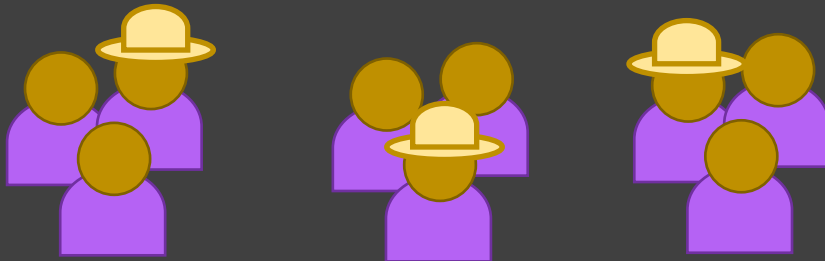




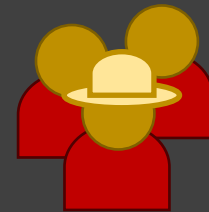
New: Tech Lead “hat”
in every team



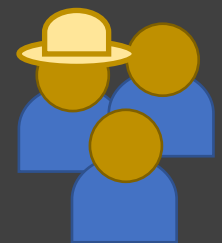
ANALYSIS & API



ACCOUNTRIGHT GAPS

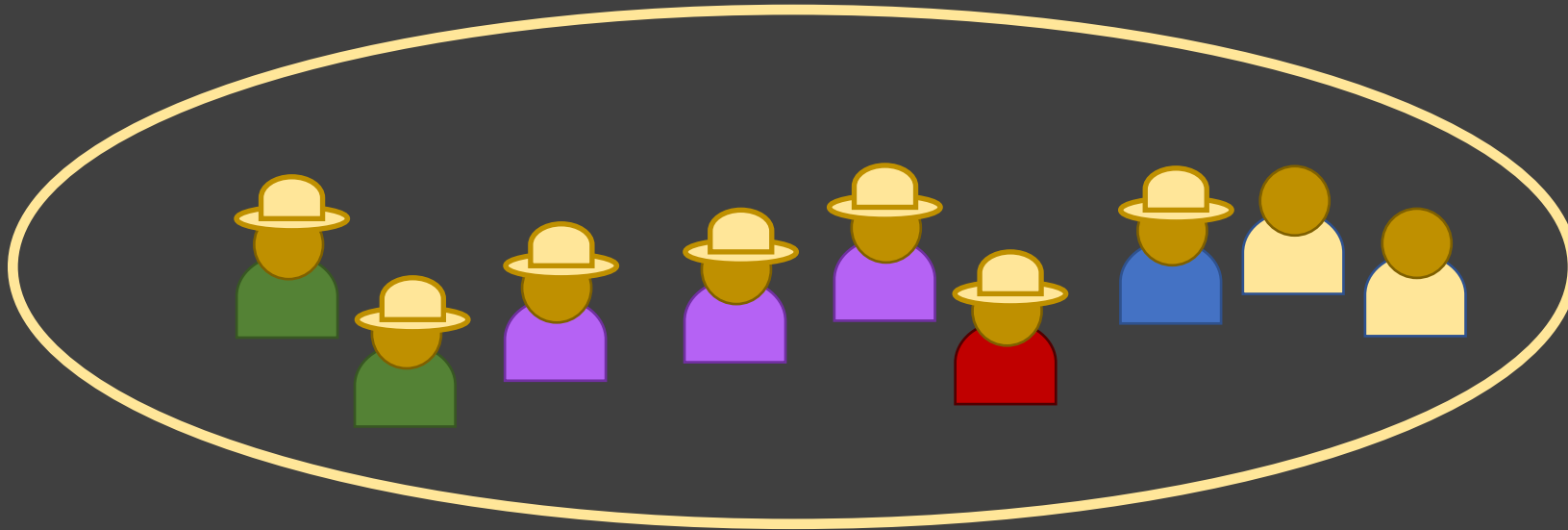


BACK END



FRONT END

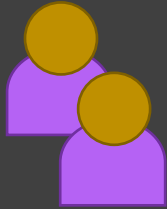
Tech Leadership Group



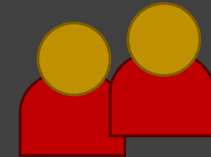
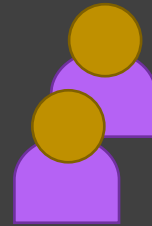
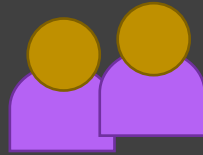
- Align on principles
- Share knowledge
- Track progress on tech decisions
- Mentor / grow



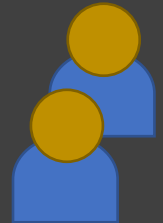
ANALYSIS & API



ACCOUNTRIGHT GAPS

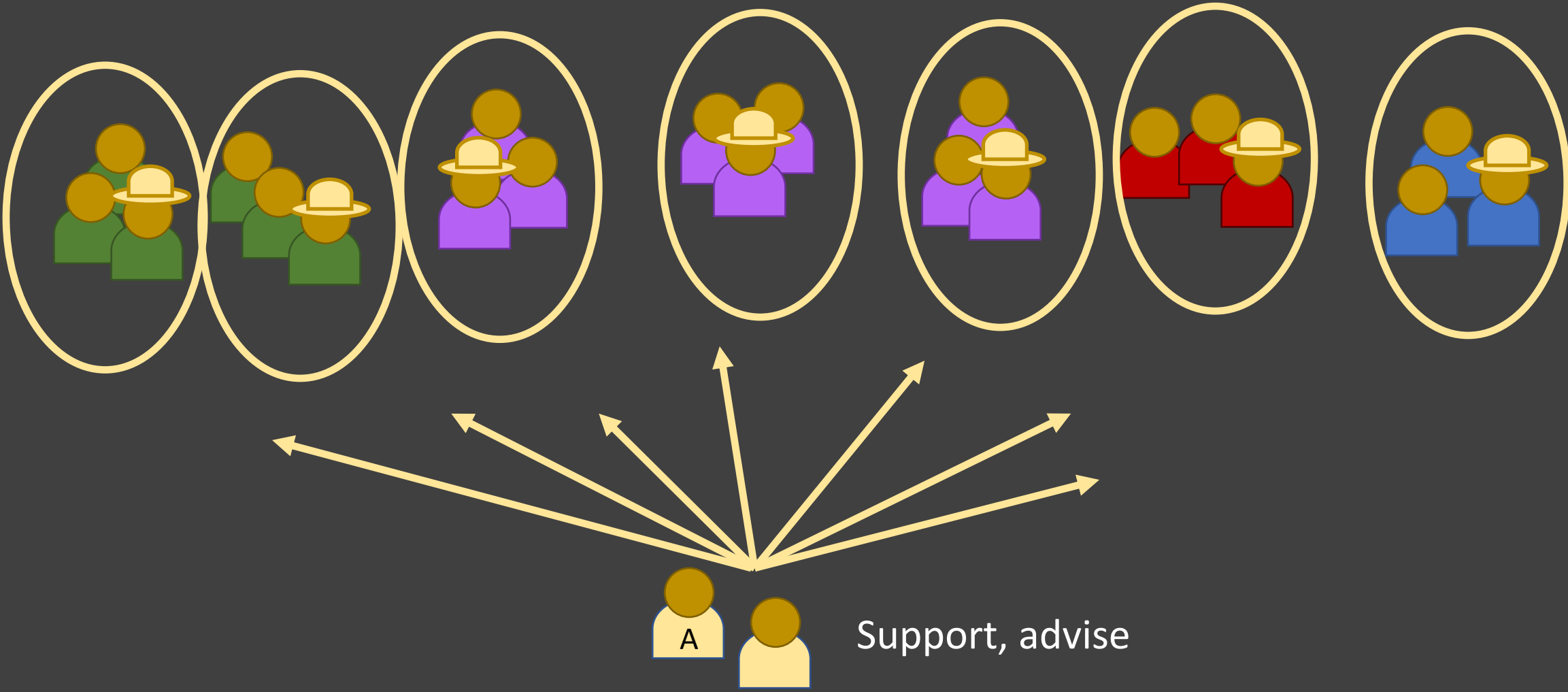


BACK END



FRONT END

~~Make~~-Produce decisions where the work is



Tech leadership group

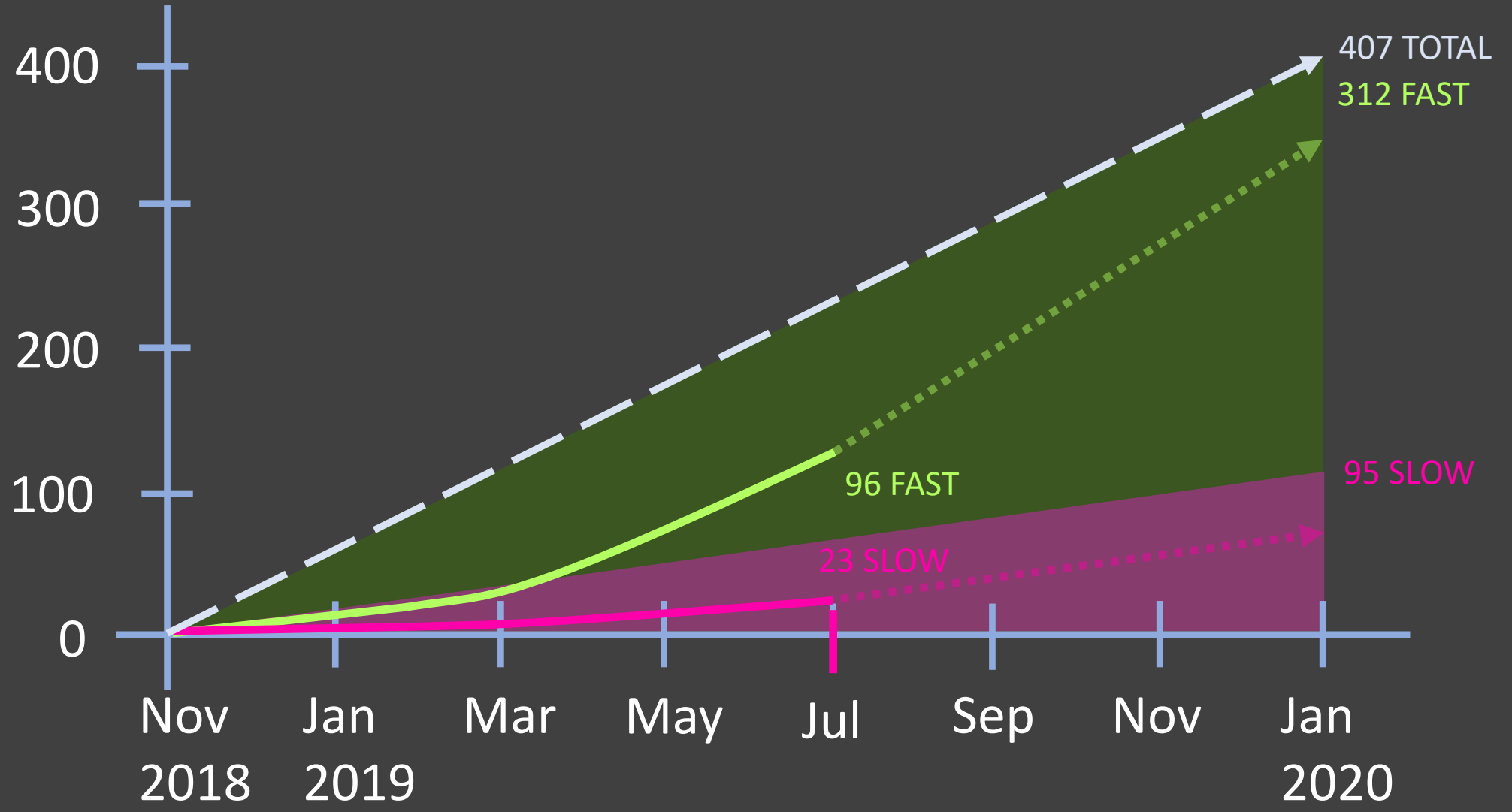
What worked

- Rapid growth in leadership talent
- Talented devs can exercise more influence
- Broke the bottleneck
- More, better decisions produced

What didn't work

- Too much work for tech leads
- Too much distraction from team work
- Laundering work outside of org structure = bad

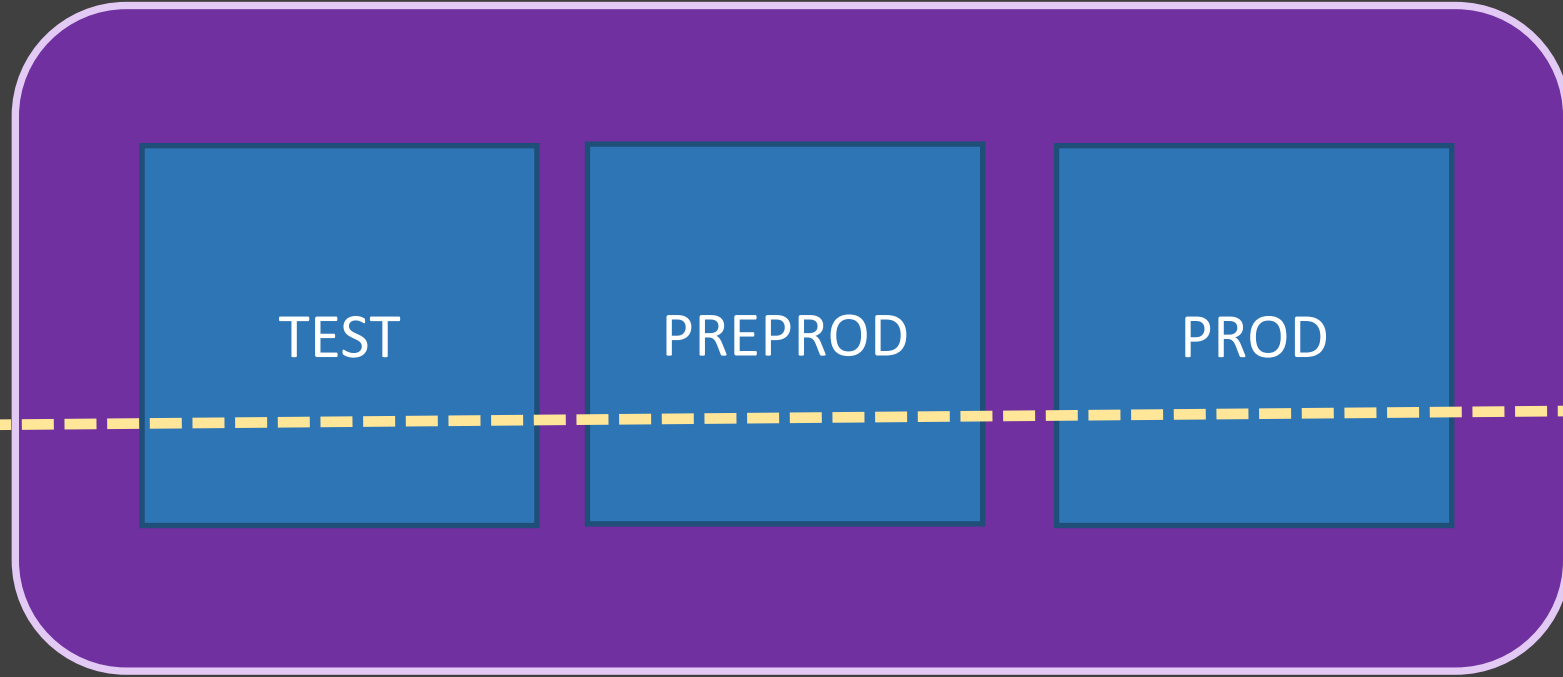
Features



Early access program

DEVELOPERS

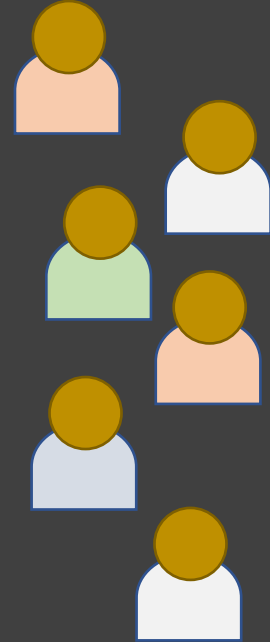
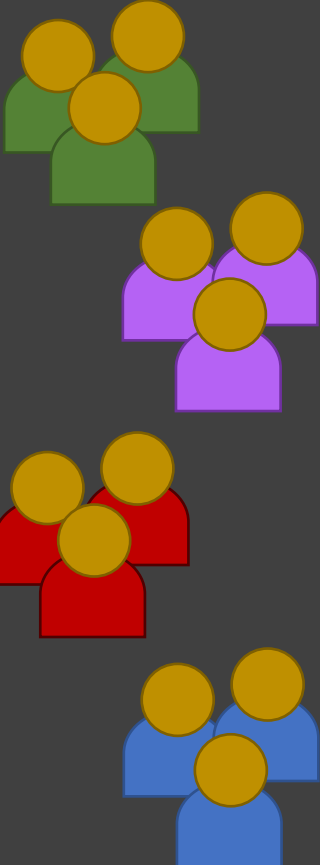
EAP USERS



TEST

PREPROD

PROD





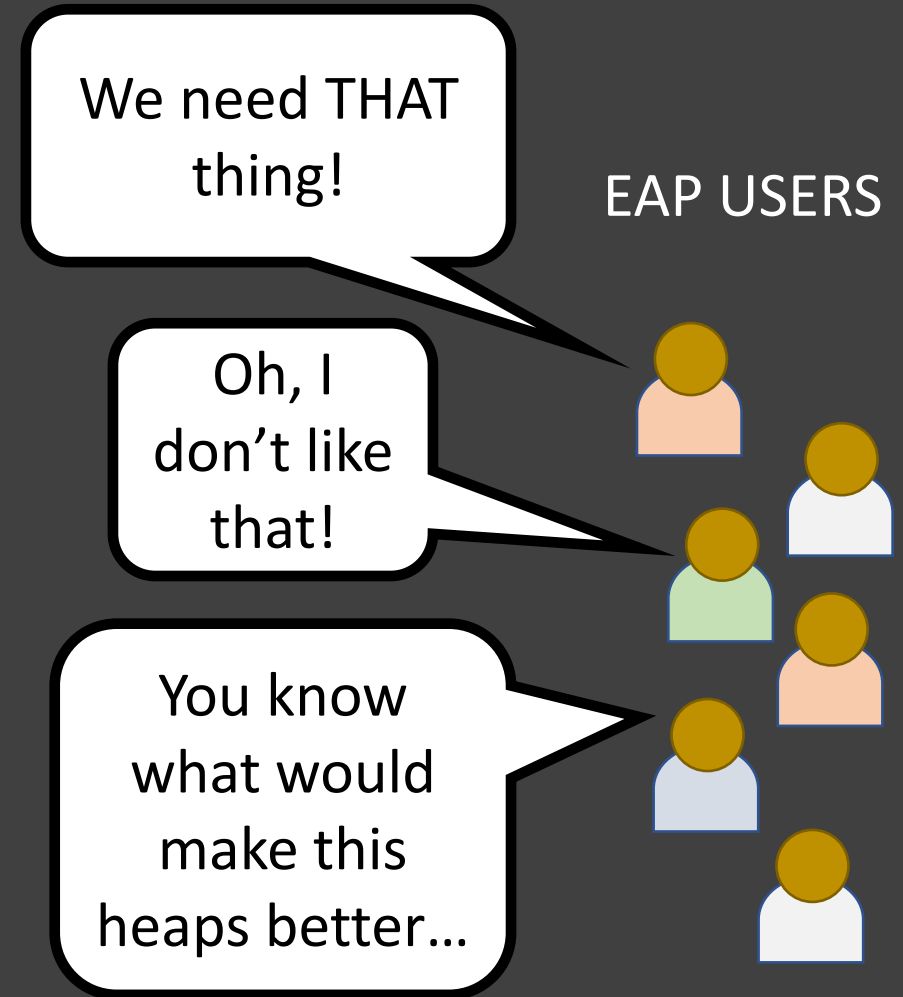
Users don't like
half finished
software

Tradeoffs

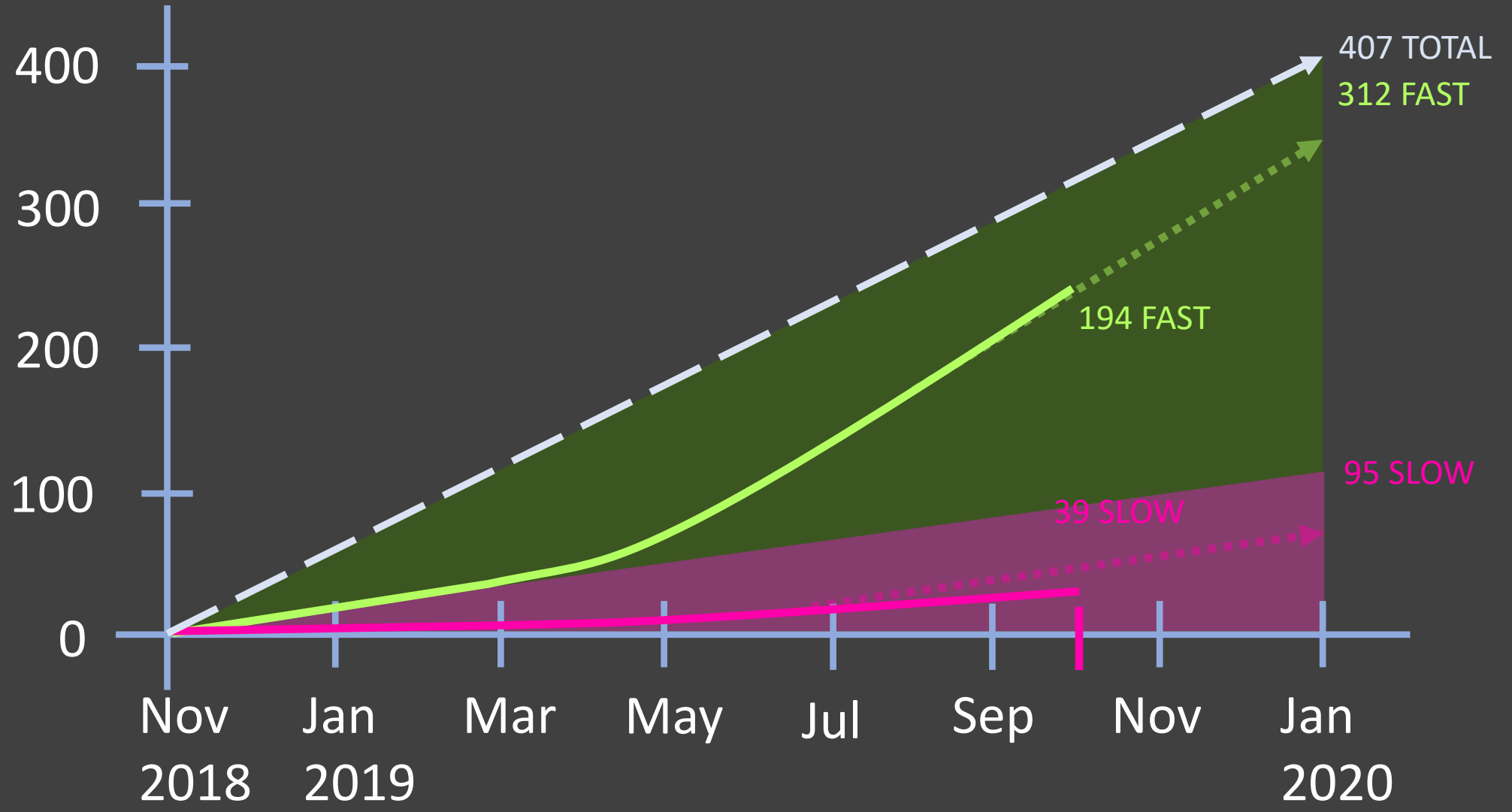
- Great to have real user feedback!
- Much less risk to the end product

BUT

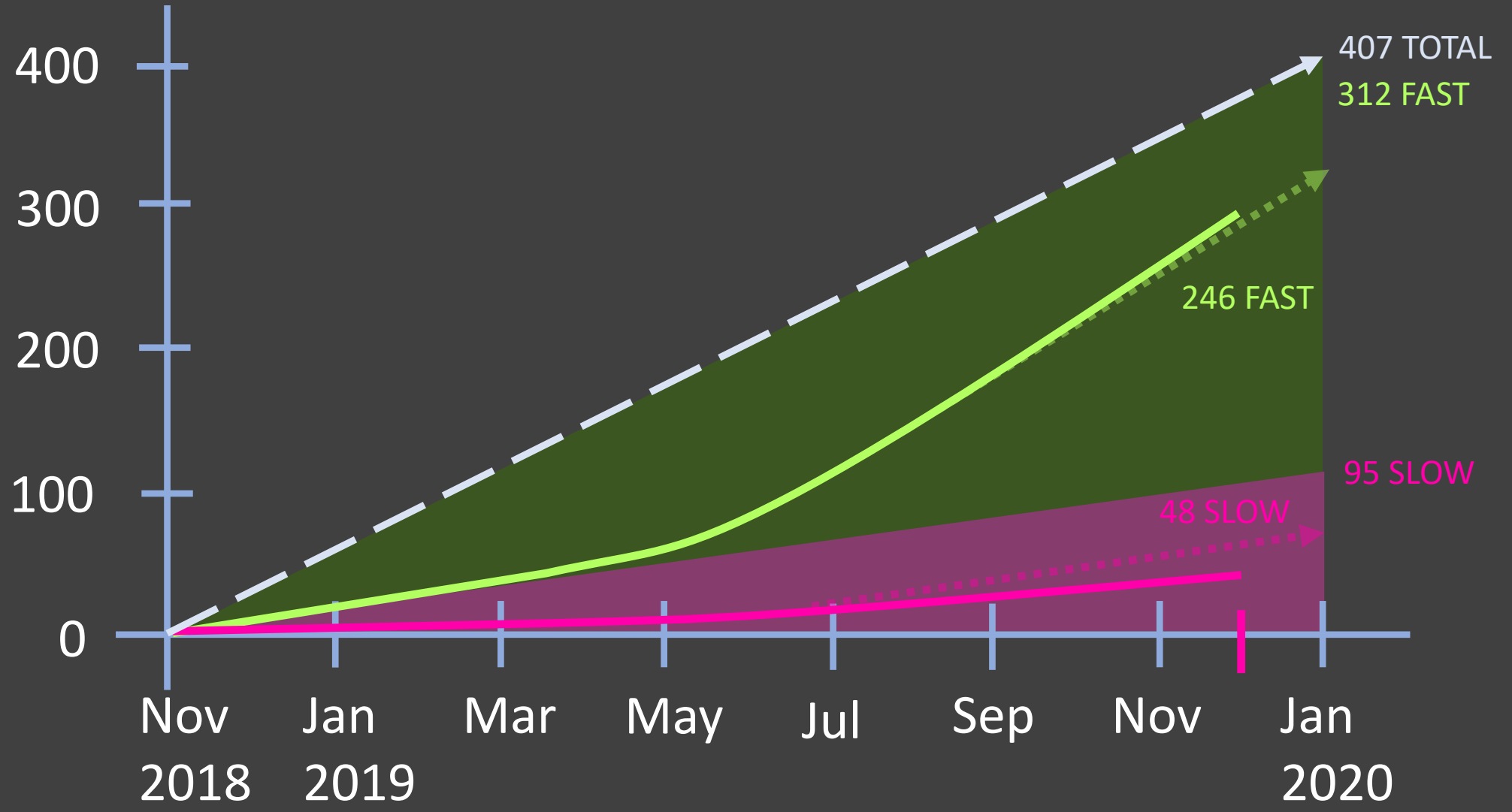
- Inevitably a new source of requirements
- More project complexity
- Distracting



Features



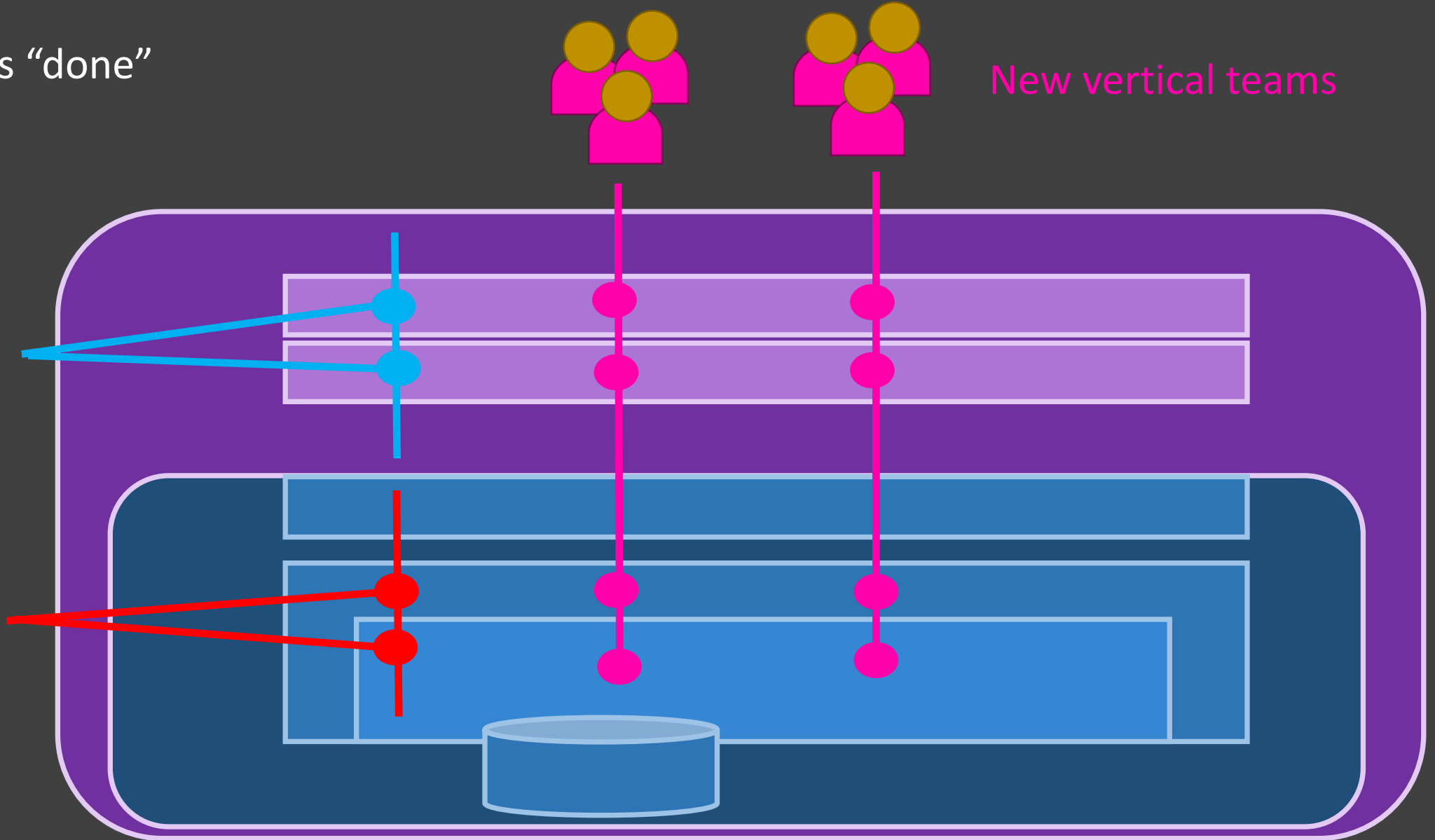
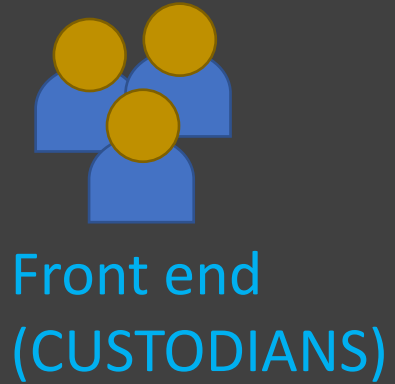
Features



System of work (take 4 – home stretch)

Analysis is “done”

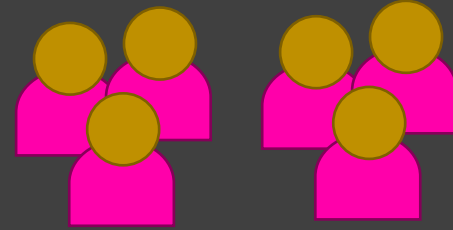
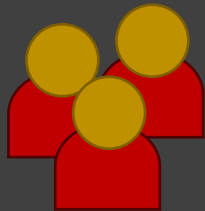
New vertical teams



HORIZONTAL

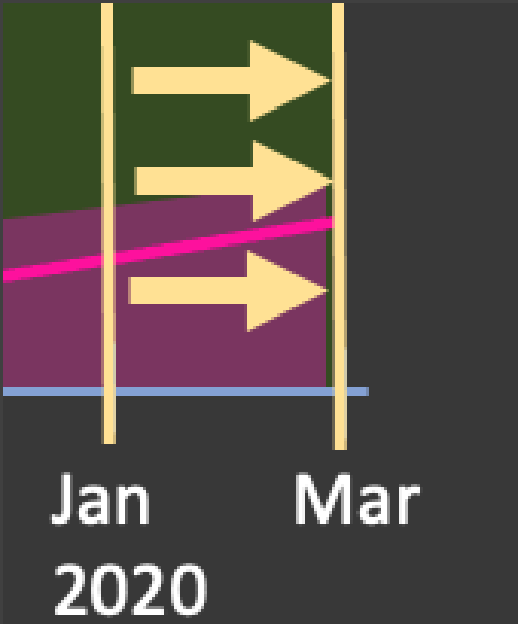


- Hardened “performing” teams
- E2E incoherency → defects
- Mitigated by checkpoints
- New custodianship burden



- Fresh “storming” teams
- Biz aligned
- Bad at tech patterns

VERTICAL



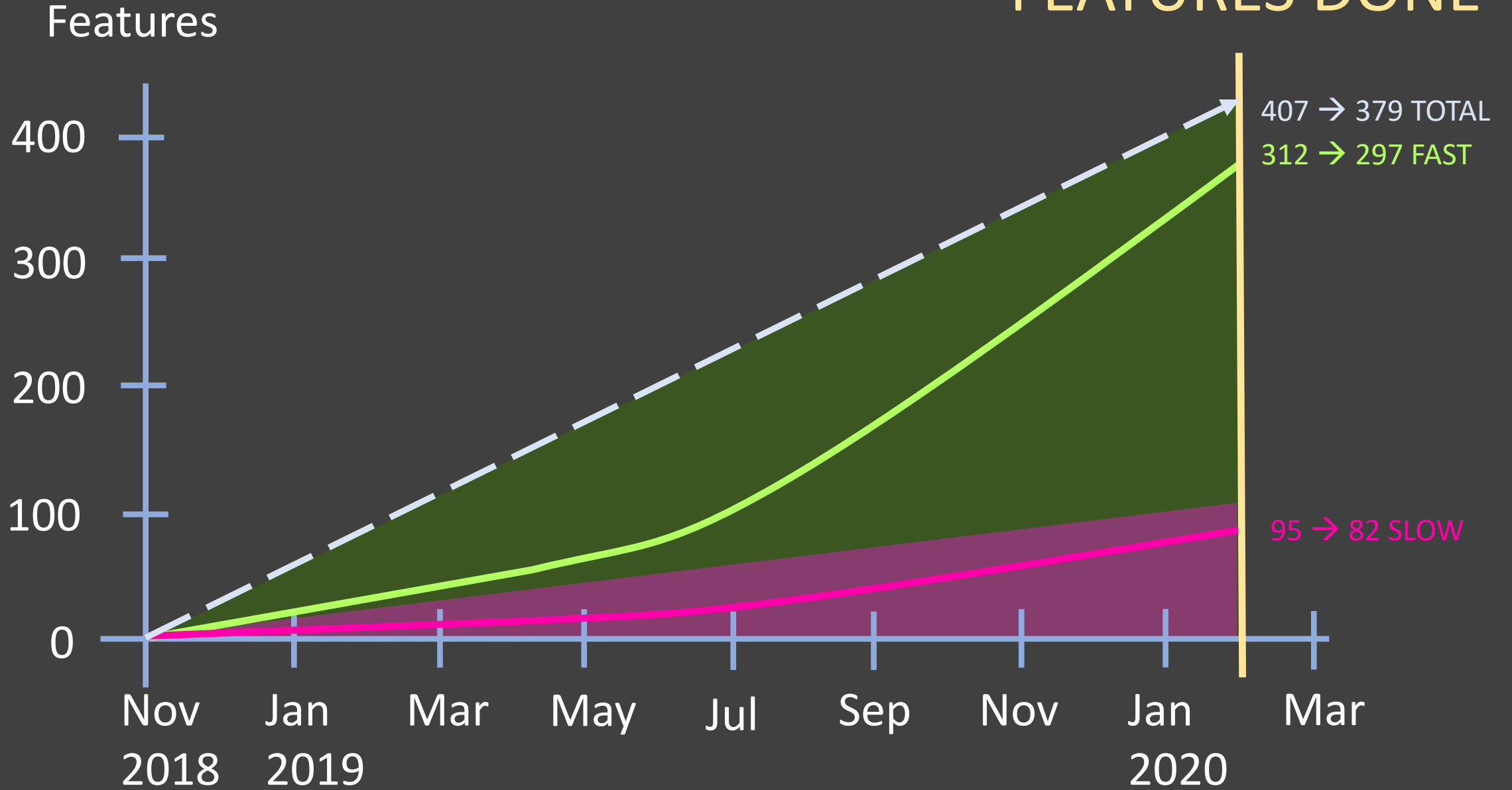
1. Extension to March

2. Dropped scope



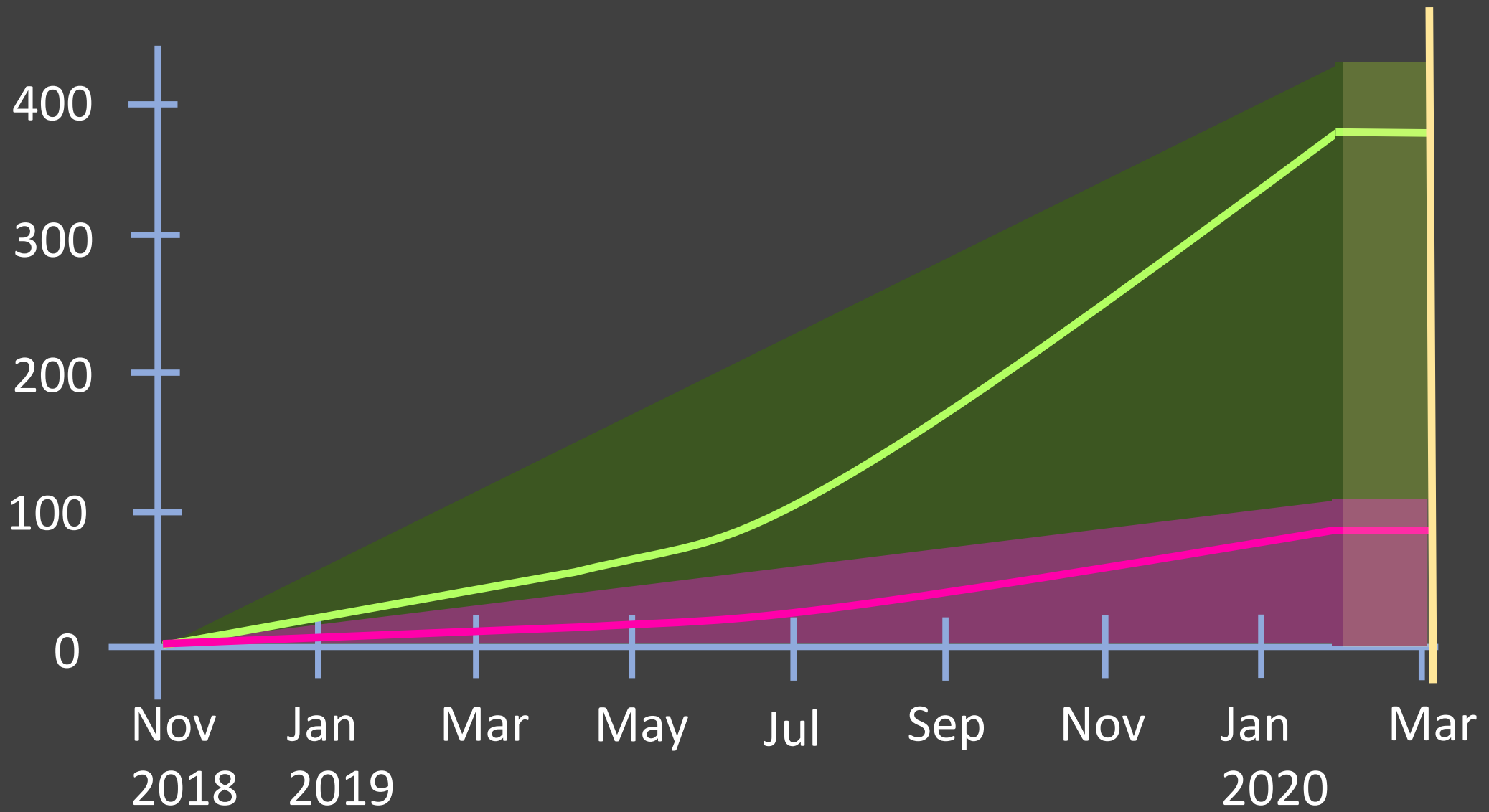
3. Tech debt loans

FEATURES DONE



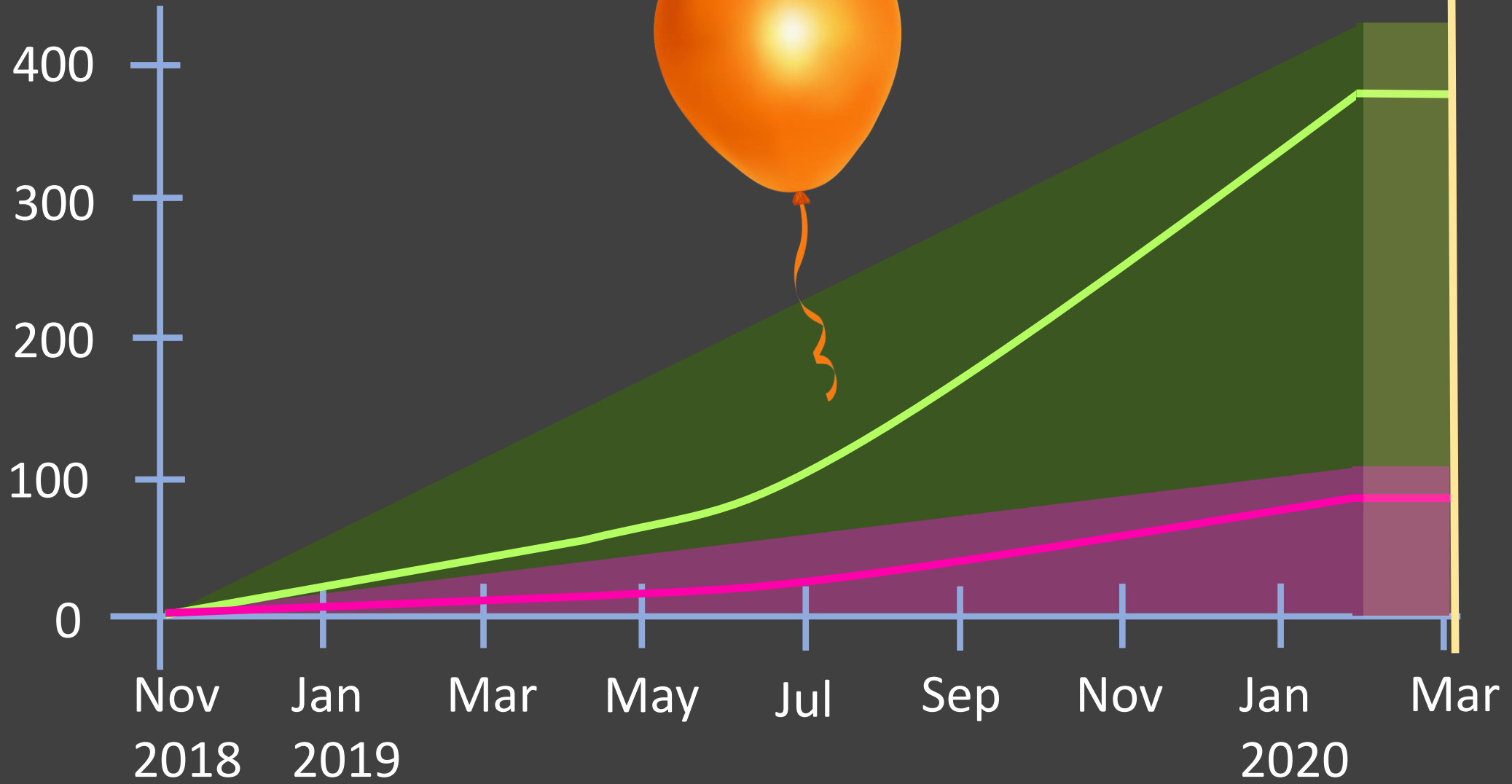
Features

RELEASED



Features

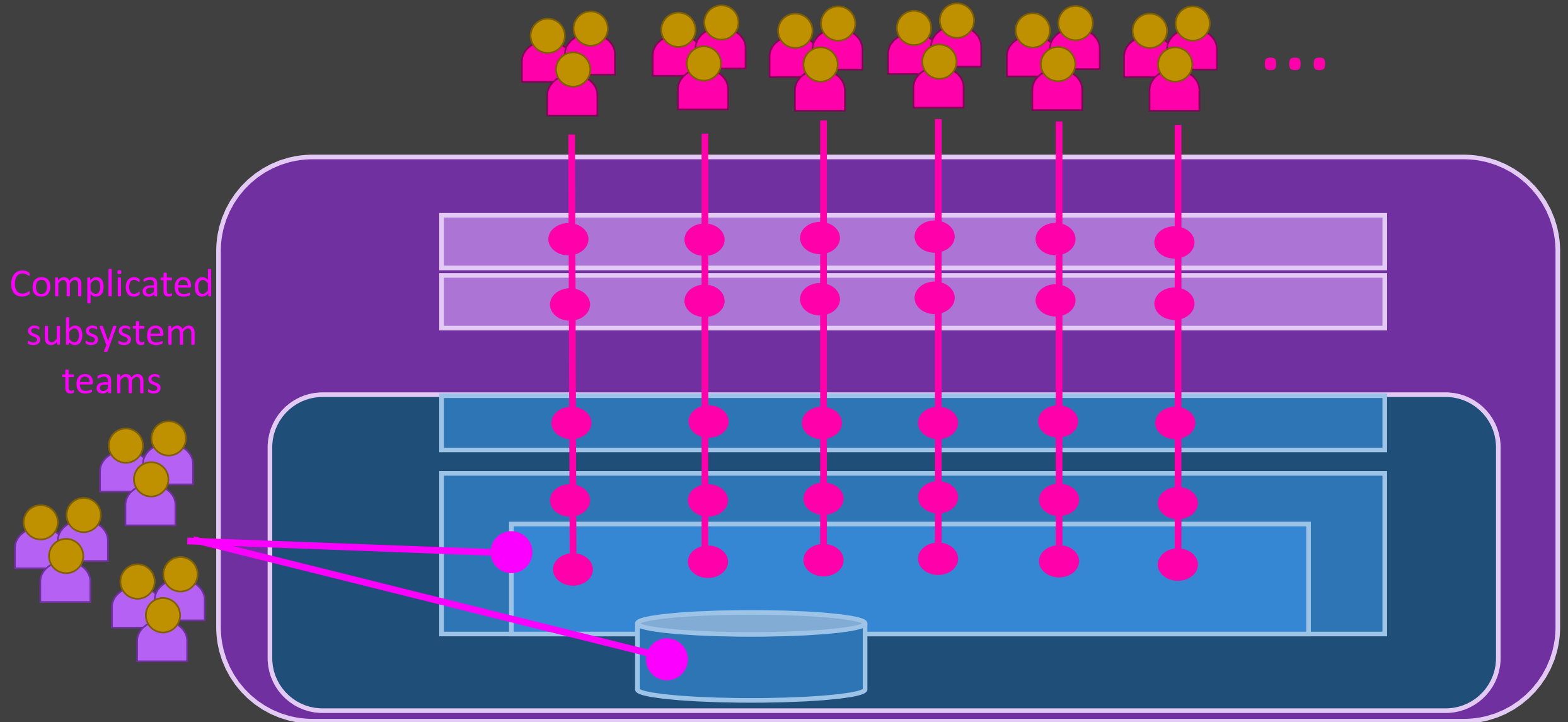
RELEASED



IV. The aftermath

Flipping the axis

Stream-aligned teams



Stream-aligned team: aligned to a flow of work from (usually) a segment of the business domain.

Complicated Subsystem team: where significant mathematics/calculation/technical expertise is needed.

Platform team: a grouping of other team types that provide a compelling internal product to accelerate delivery by Stream-aligned teams.



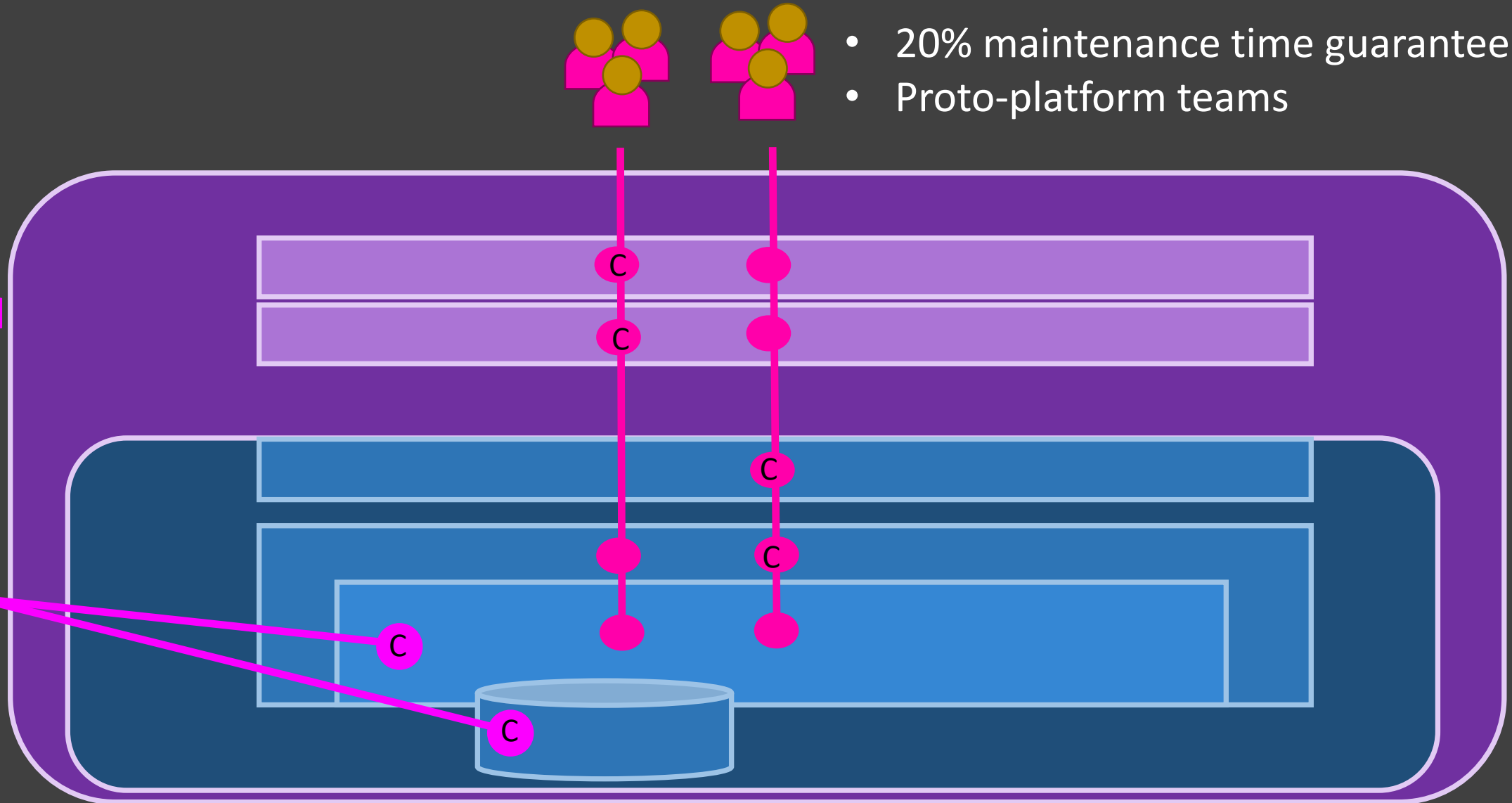
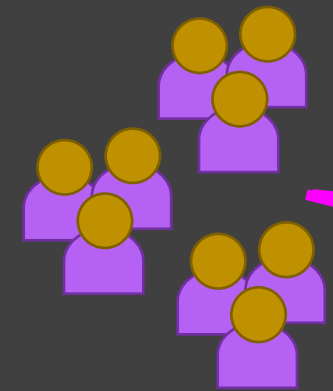
Matthew Skelton Manuel Pais
“Team Topologies” (2019)

Custodianship

Stream-aligned custodians

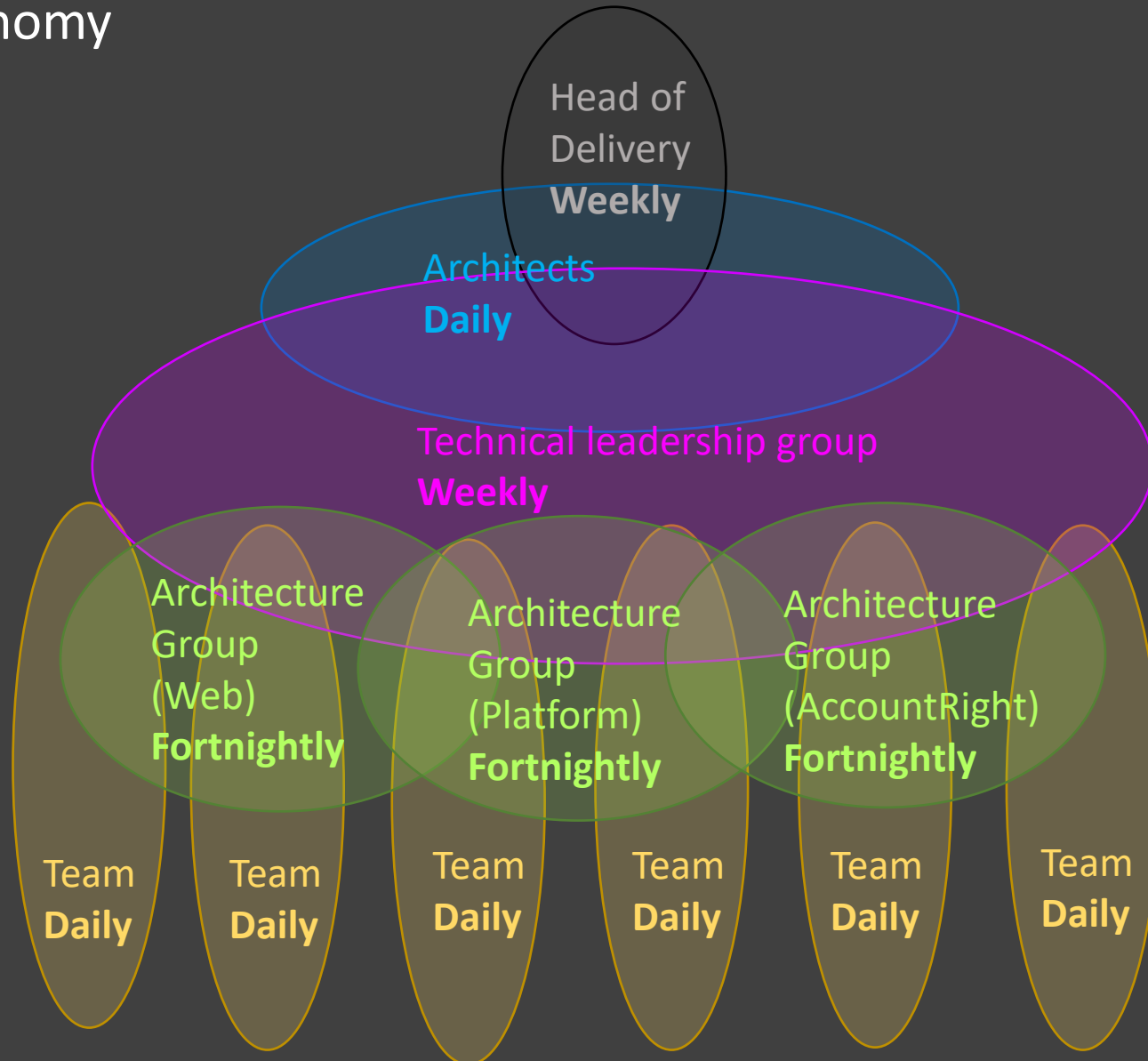
- 20% maintenance time guaranteed
- Proto-platform teams

Complicated
subsystem
teams



Overlapping communication structures

Aligned autonomy

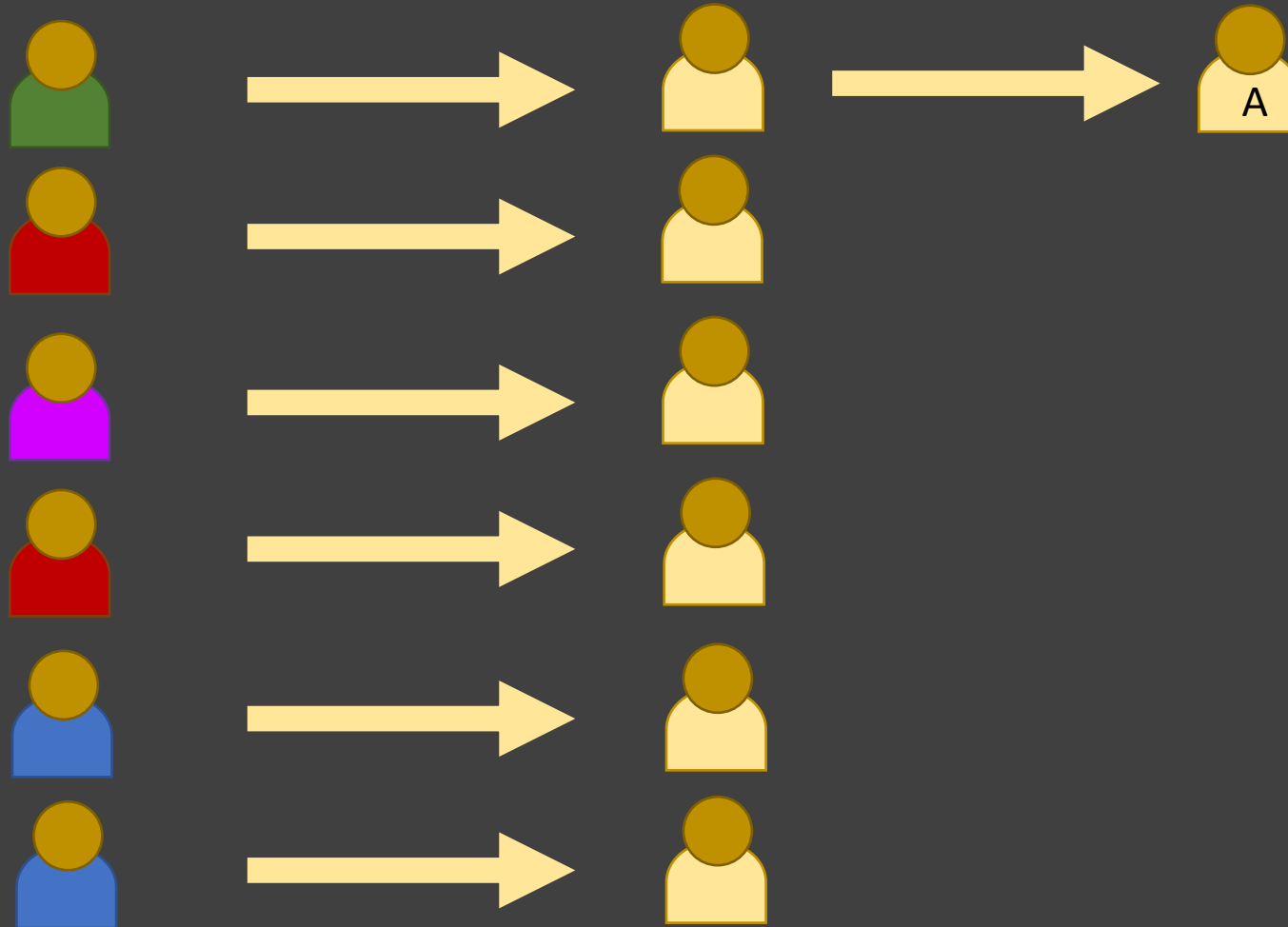


Architecture Groups

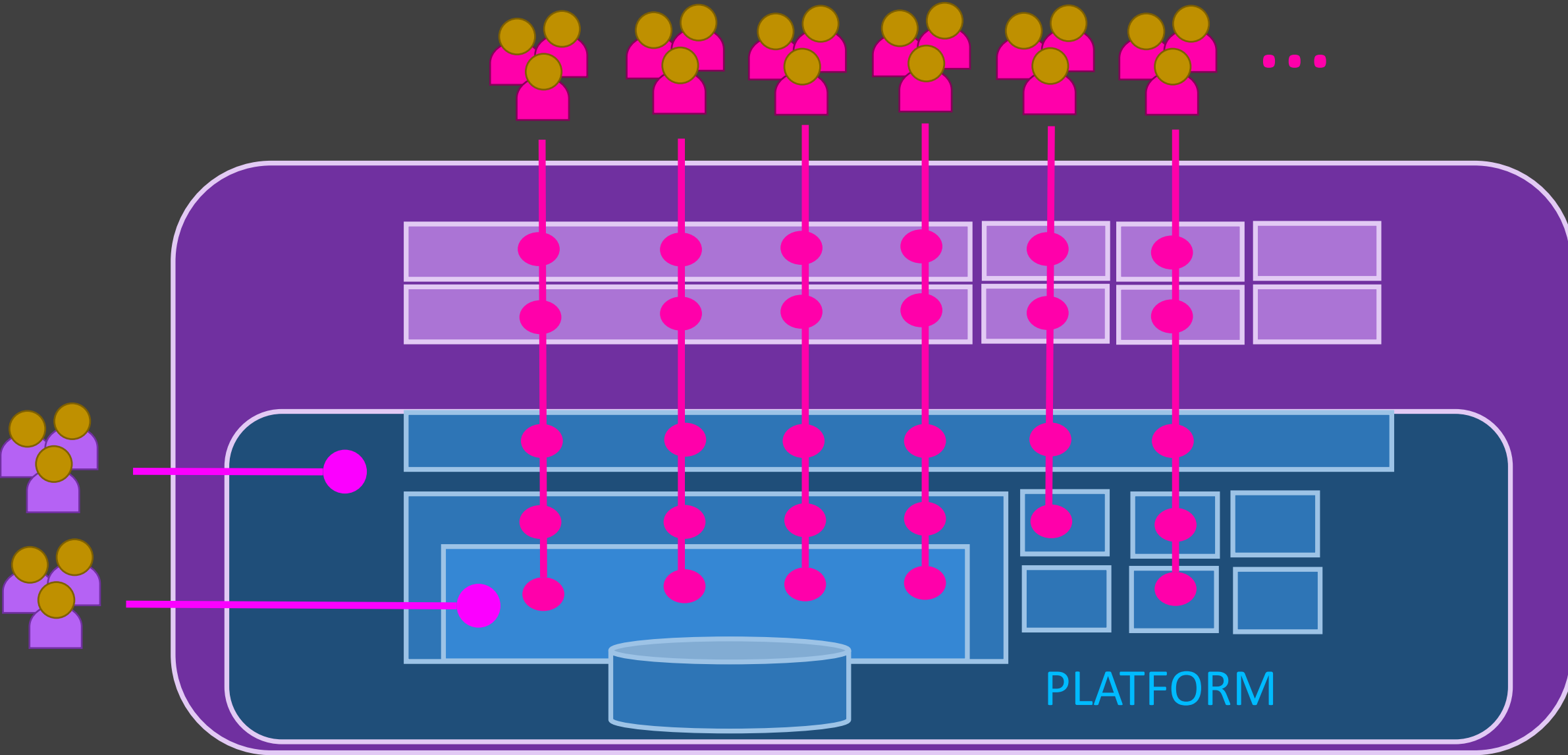
- Broad membership
- Alignment on codebase conventions

Internal tech leadership

By mid 2020: 6 promotions to Principal Developer or Architect



From a project to a platform



Winning!



Winning?



High stakes don't mean overtime

Instead:

- Data-driven tuning
- Data-driven prioritization
- Dropping scope
- Tasteful technical compromises