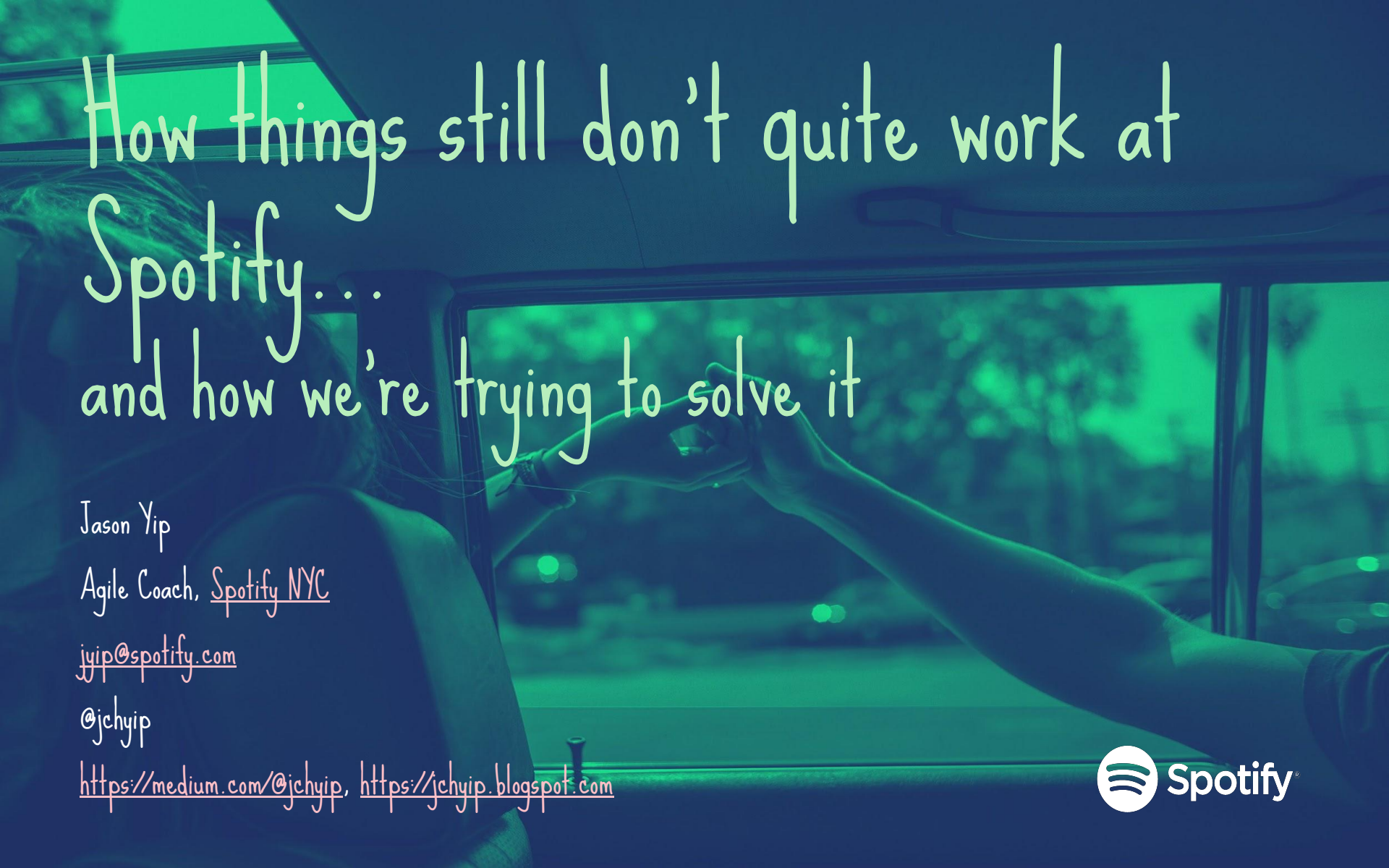


How things still don't quite work at Spotify...

and how we're trying to solve it

Jason Yip

Agile Coach, Spotify NYC

jyip@spotify.com

@jchyip

https://medium.com/@jchyip, https://jchyip.blogspot.com

Spotify

One of my favourite Toyota-isms

"No problem is a problem"

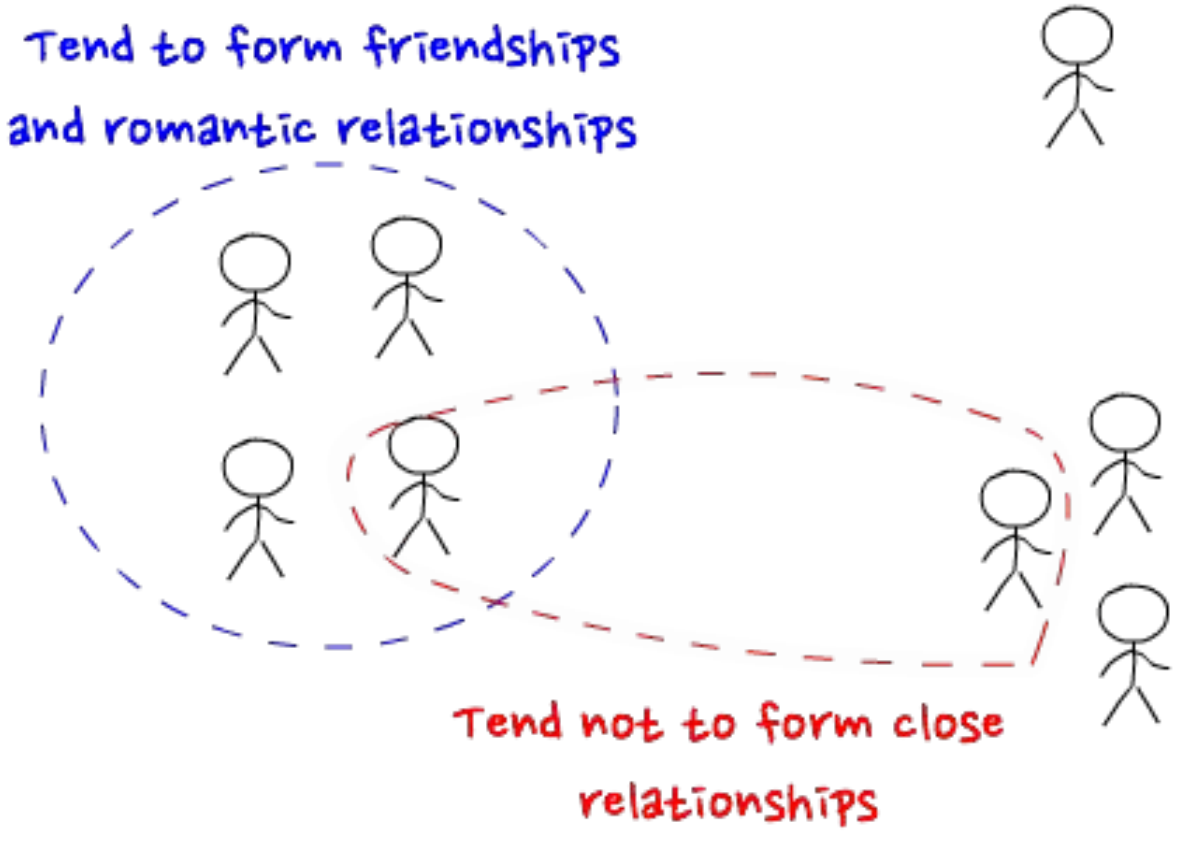
In essence, we won't improve unless we are willing to explore what isn't working.

I'm not going to talk about what's great at Spotify

I'm going to talk about what still doesn't quite work

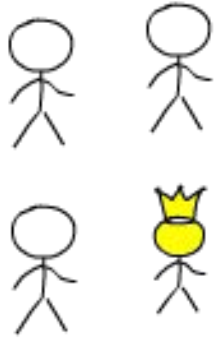
There is still too much distance between problems and problem-solvers.

Propinquity effect

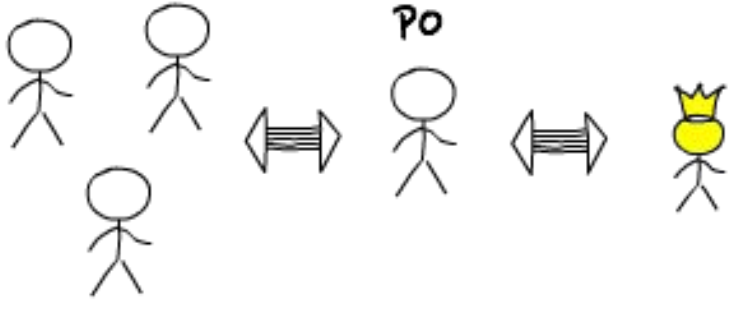


Propinquity effect and Agile

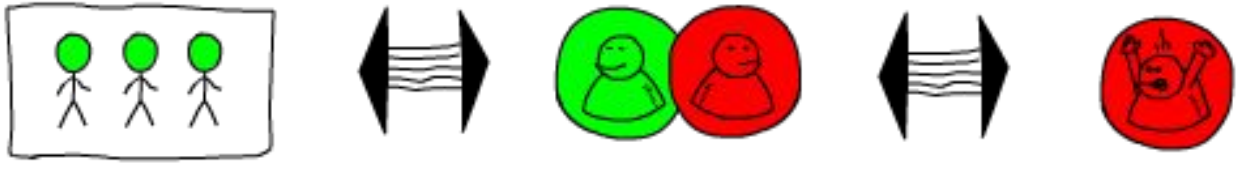
Tend to have good customer-team relationships



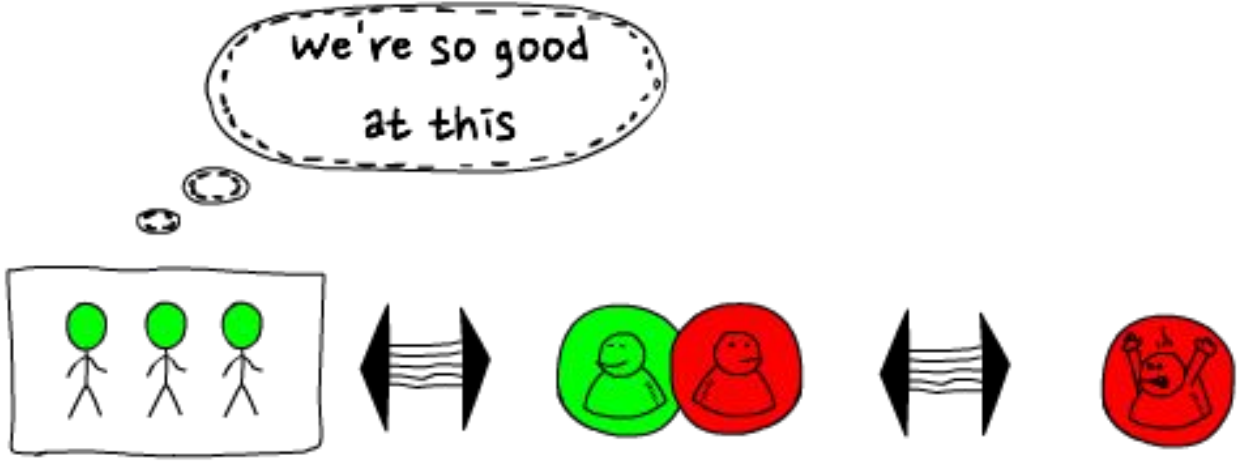
Tend not to have good customer-team relationships



It is hard to have empathy when you are "protected" from contact

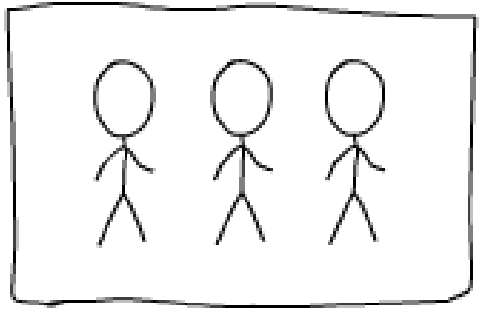


It is hard to build the right things if you don't have empathy

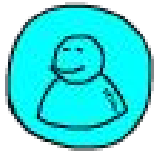


Gatekeeping context

- Low empathy
- Low engagement

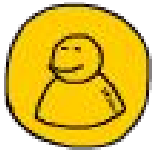


PO



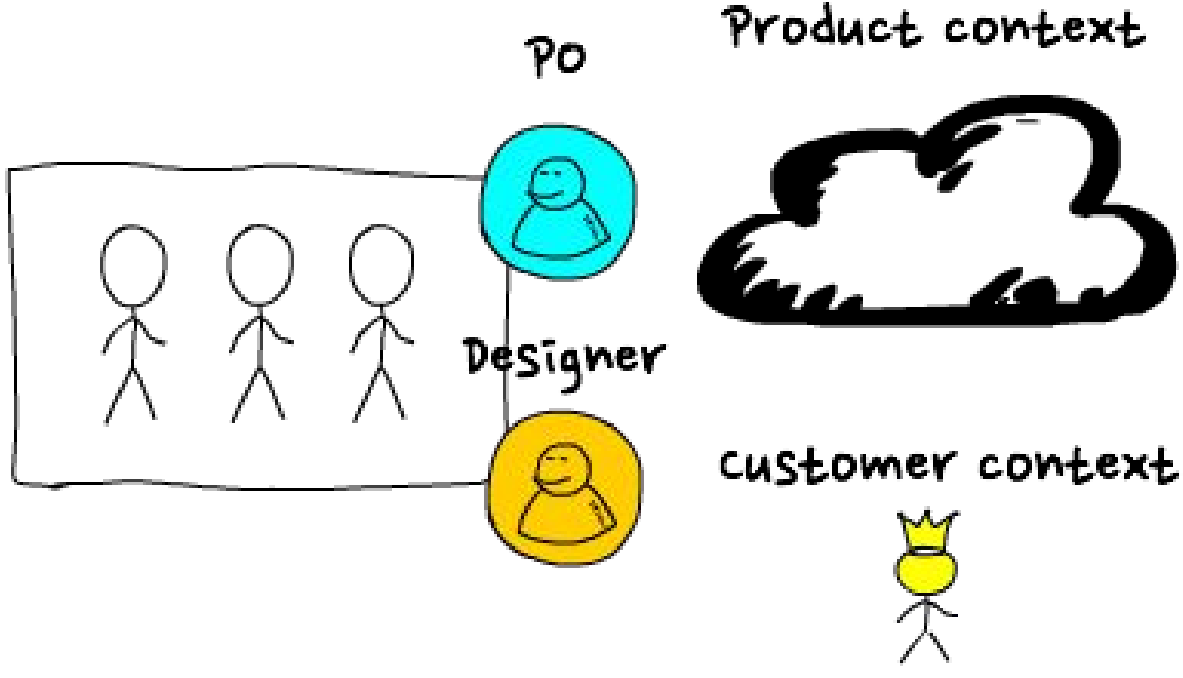
Product context

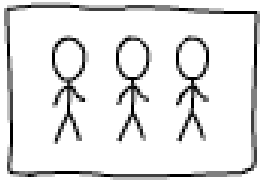
Designer



customer context

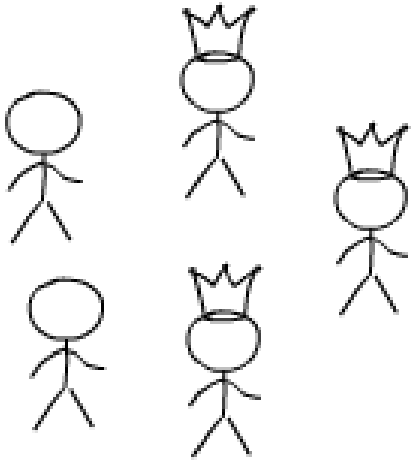
Facilitating context





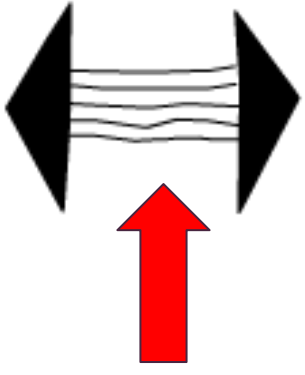
"sound check"

customer meetings



At Spotify, when we say "cross-functional" we mean "not just R&D"

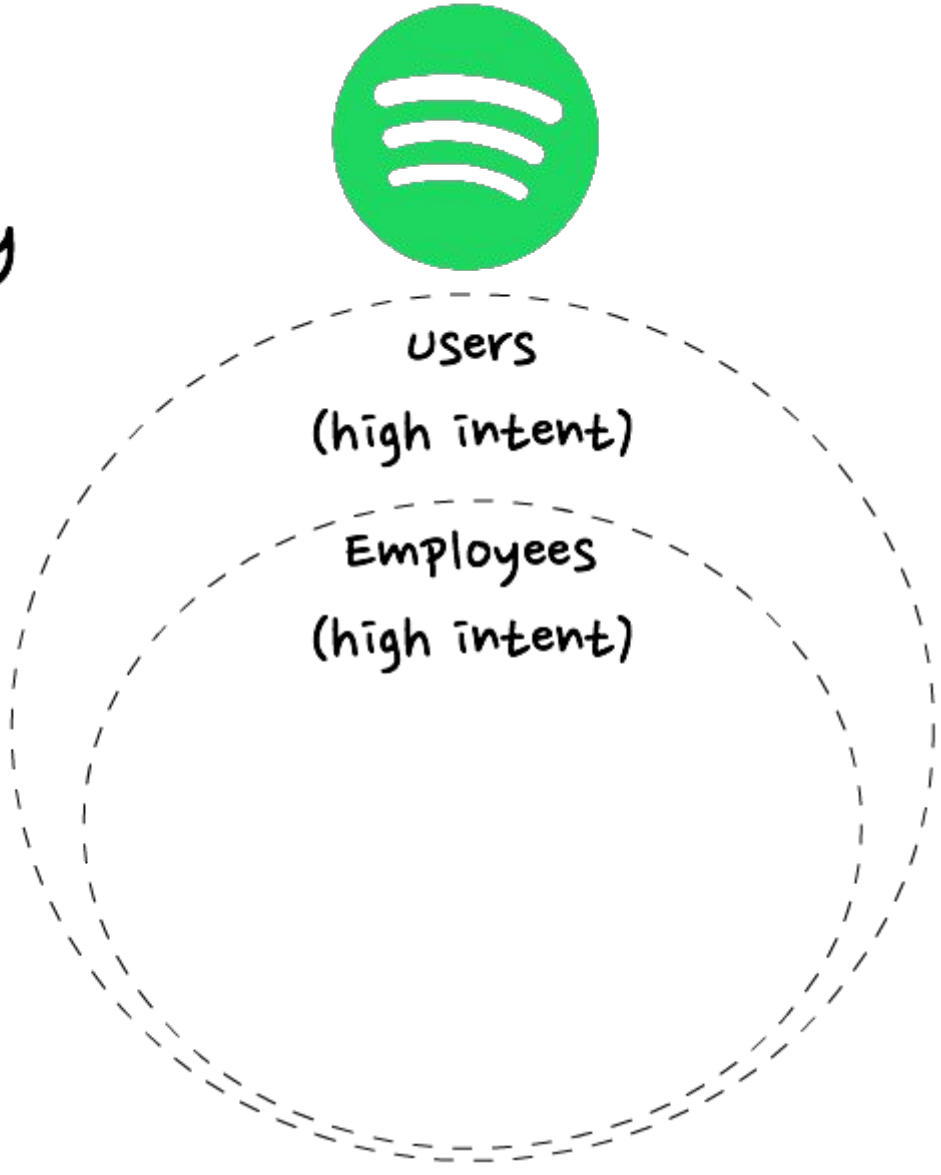
R&D



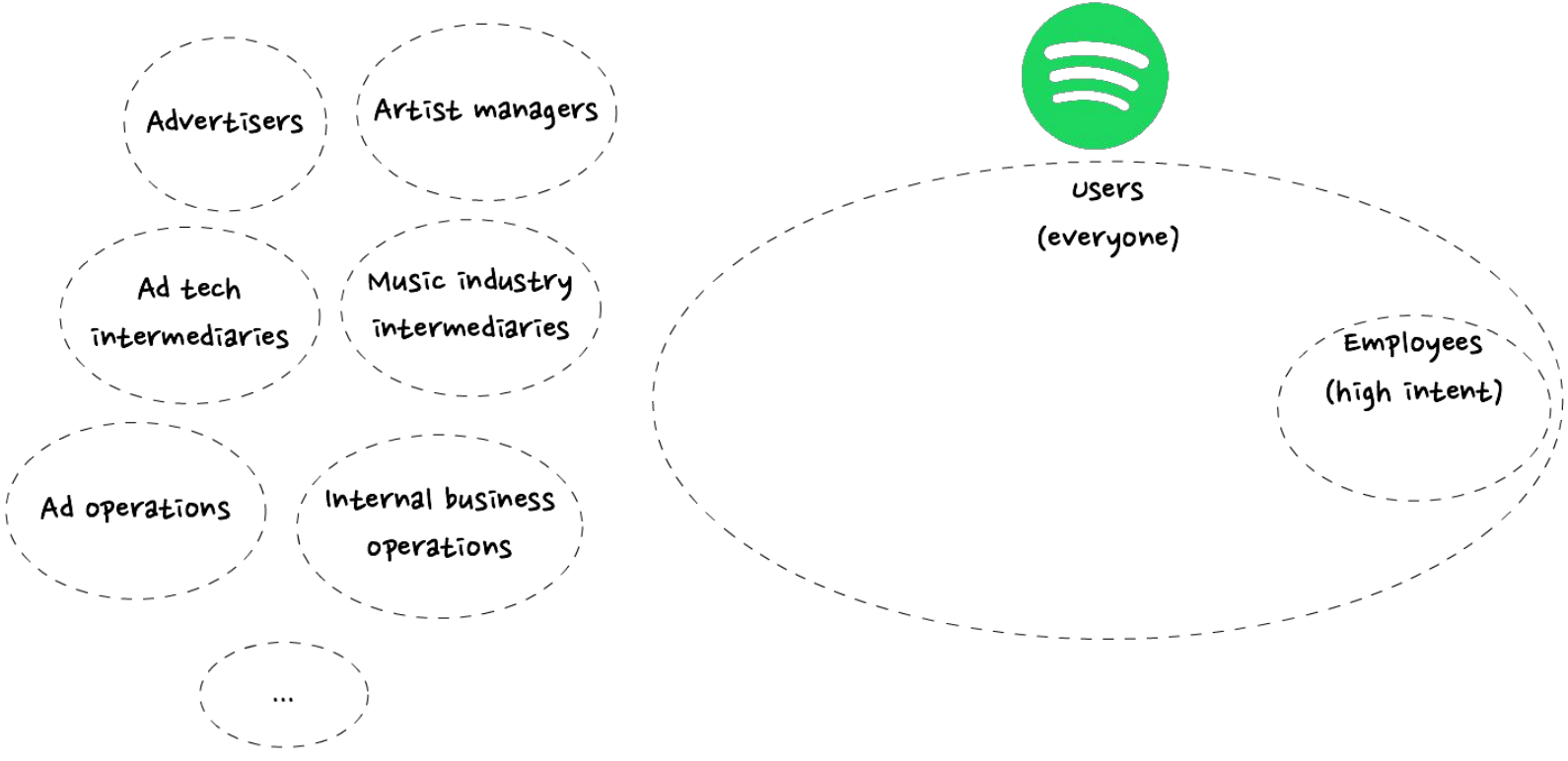
AKA distance

"Business"

I suspect this is why we got away with it

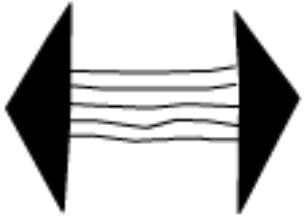


We can't really get
away with it...



Does this structure need to fundamentally change?

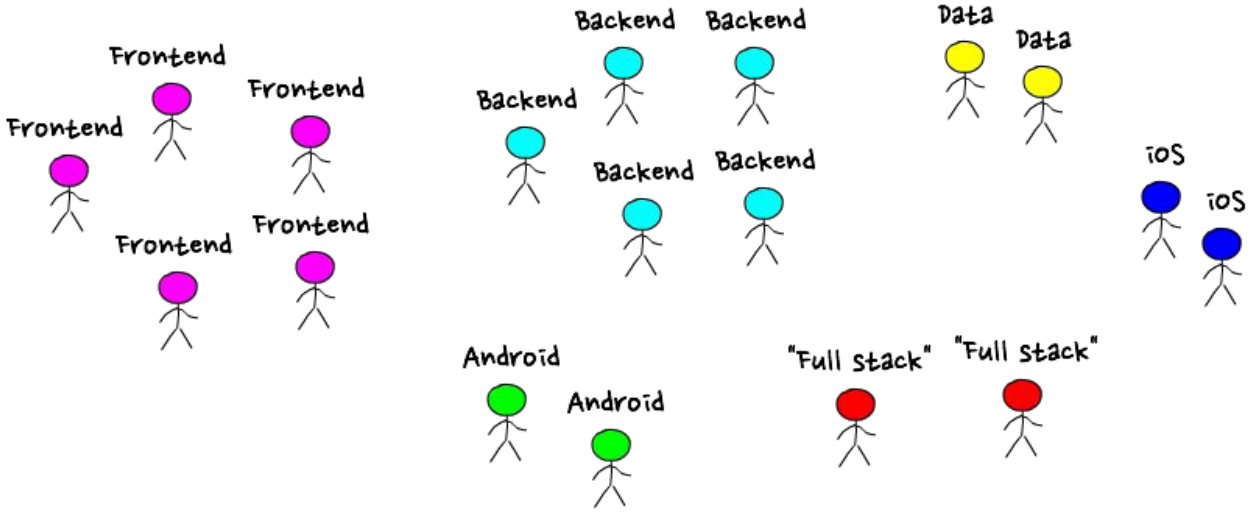
R&D



"Business"

There is still too much specialisation.

Another thing I noticed when joining...

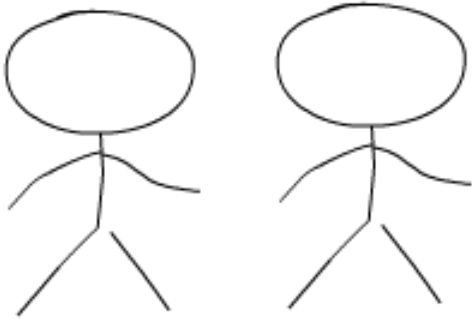


"There's not enough backend work to do."

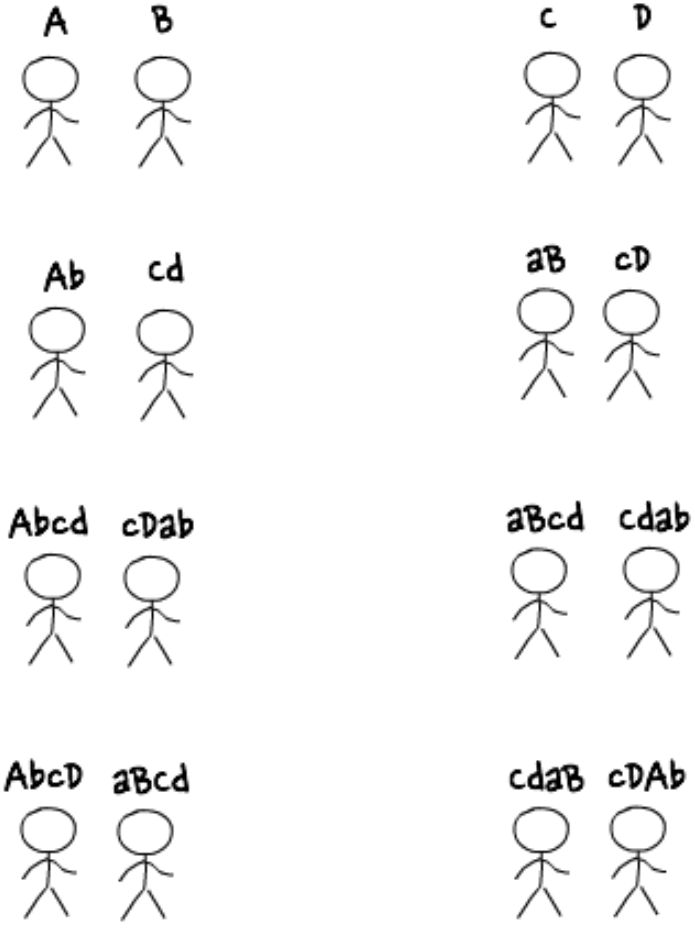
"We might as well work on this until we get another iOS developer."

"We're stuck until someone can start on the frontend."

There are some interesting, non-obvious dynamics that occur with pair programming.



key person dependency goes away fairly quickly

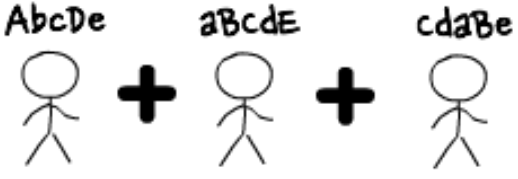


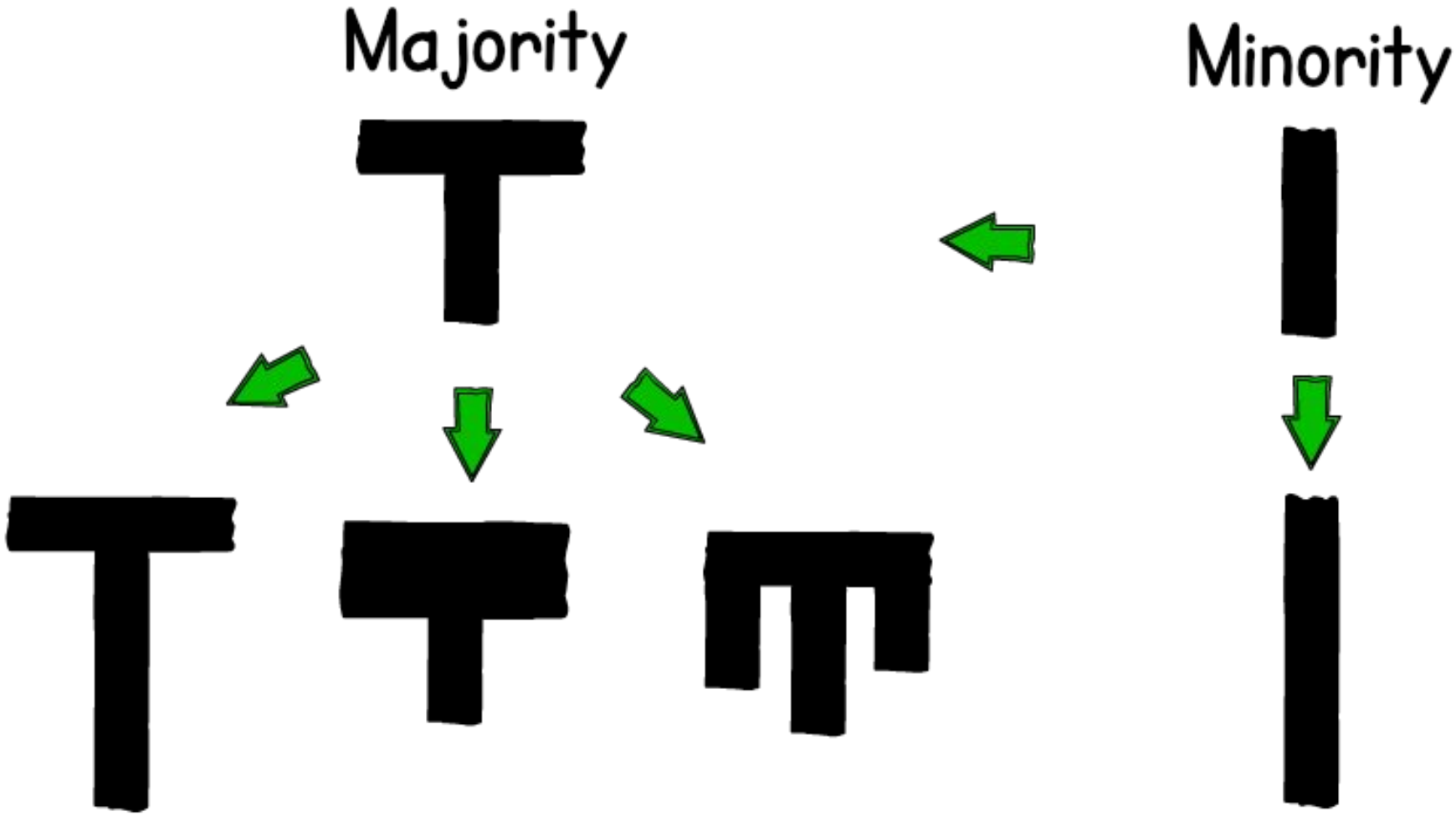
You can do things with less people

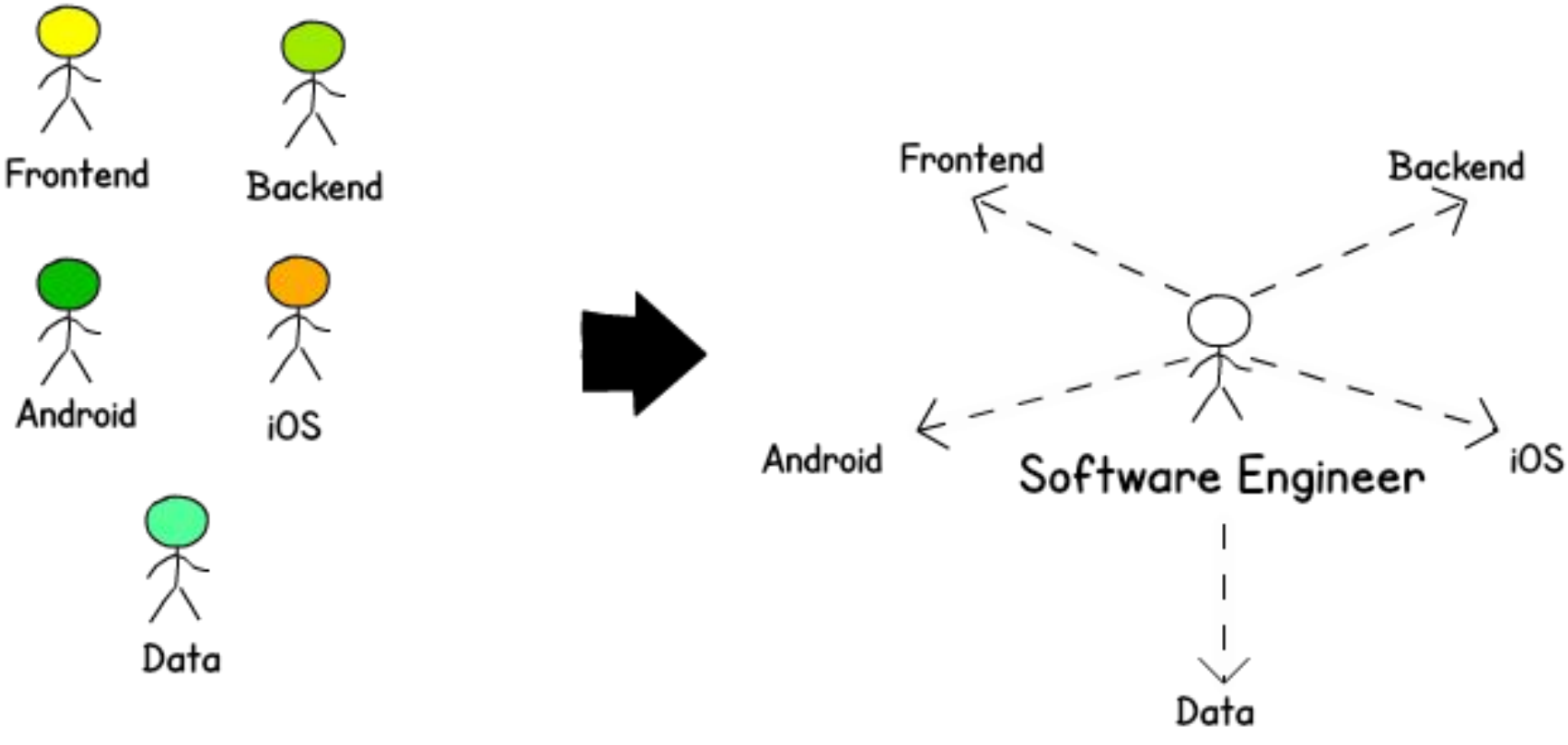
Manager

(to help coordinate all these people)

versus

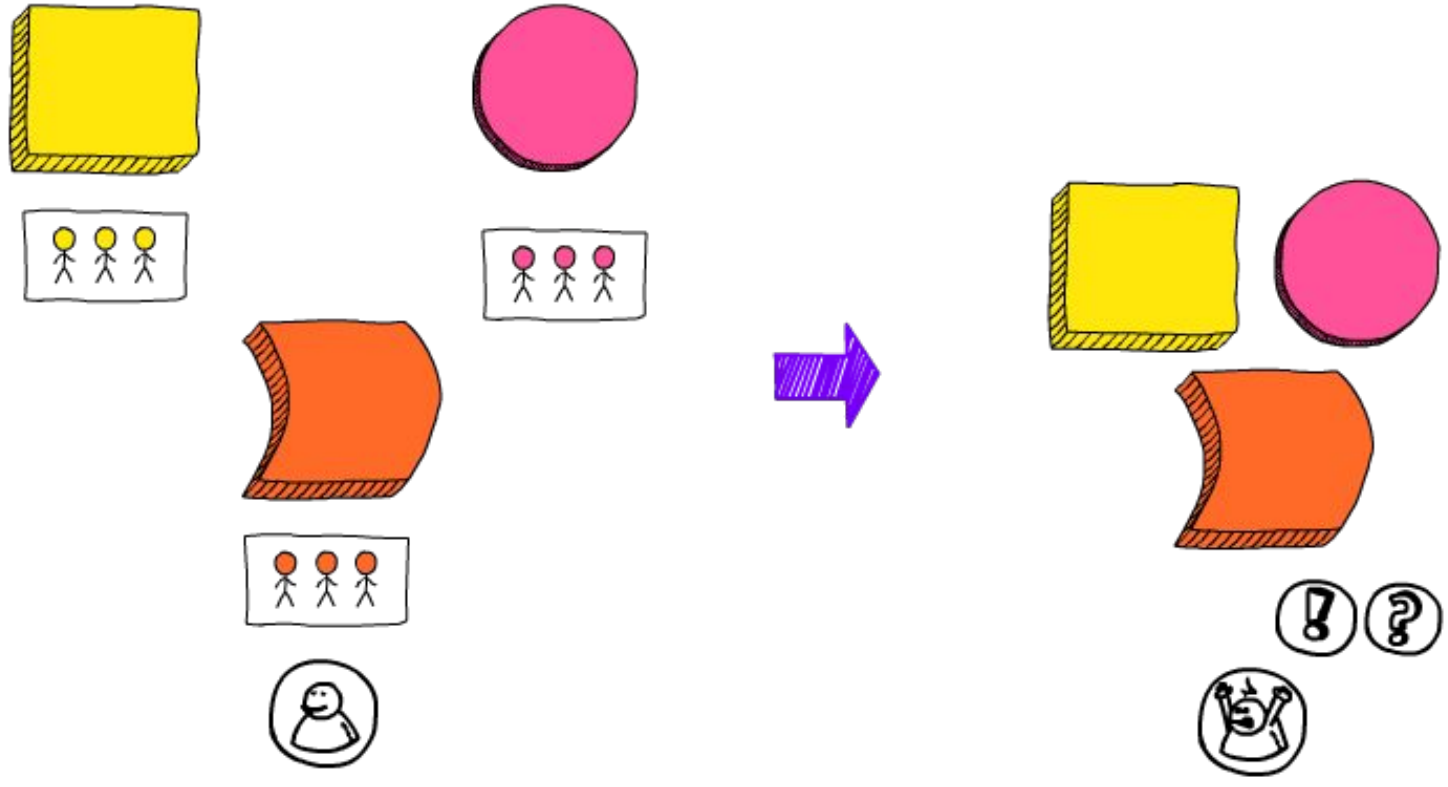



Majority
Minority

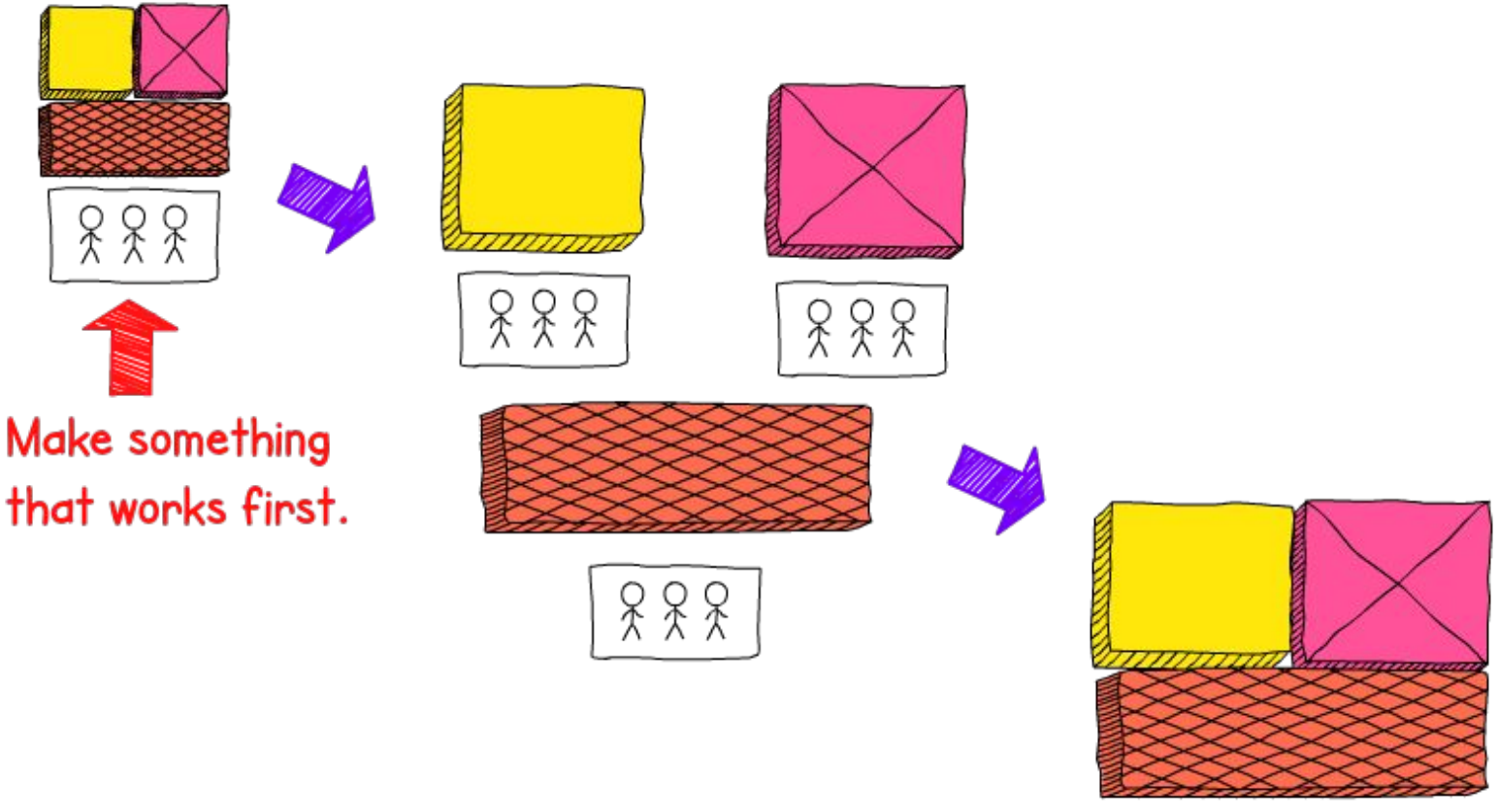

Frontend
Backend
Android
iOS
Data
Frontend
Backend
Android
Software Engineer
iOS
Data

Larger cross-Tribe / Mission initiatives are still not done well.

Divide and conquer leads to integration problems



Instead, conquer and divide



There is still a tendency to divide and conquer for large cross-Tribe initiatives.

We know this is a problem and even how to solve it.
I'm not sure yet how we'll cross this knowing-doing gap...

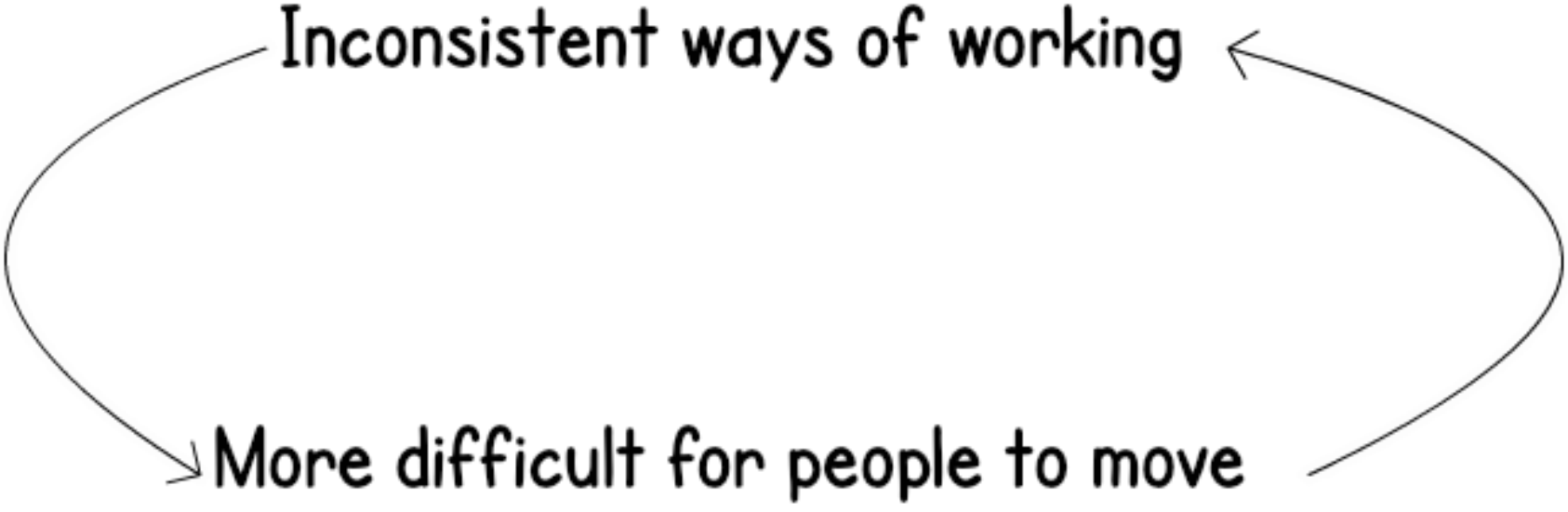
There is not enough cross-pollination of ways of working.

Useful variation

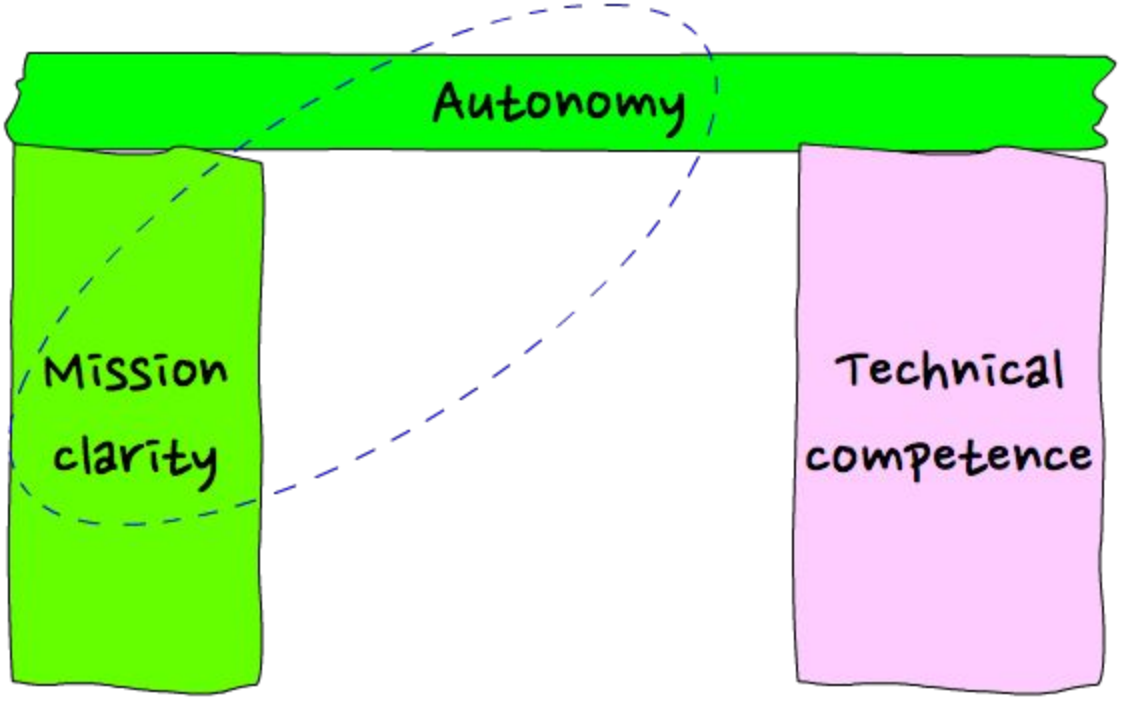
- Context-specific forces
- Experimentation
- Reinforces sense of autonomy

Non-useful variation

- Makes onboarding more complicated
- More difficult to move between Tribes / Missions


Inconsistent ways of working
More difficult for people to move

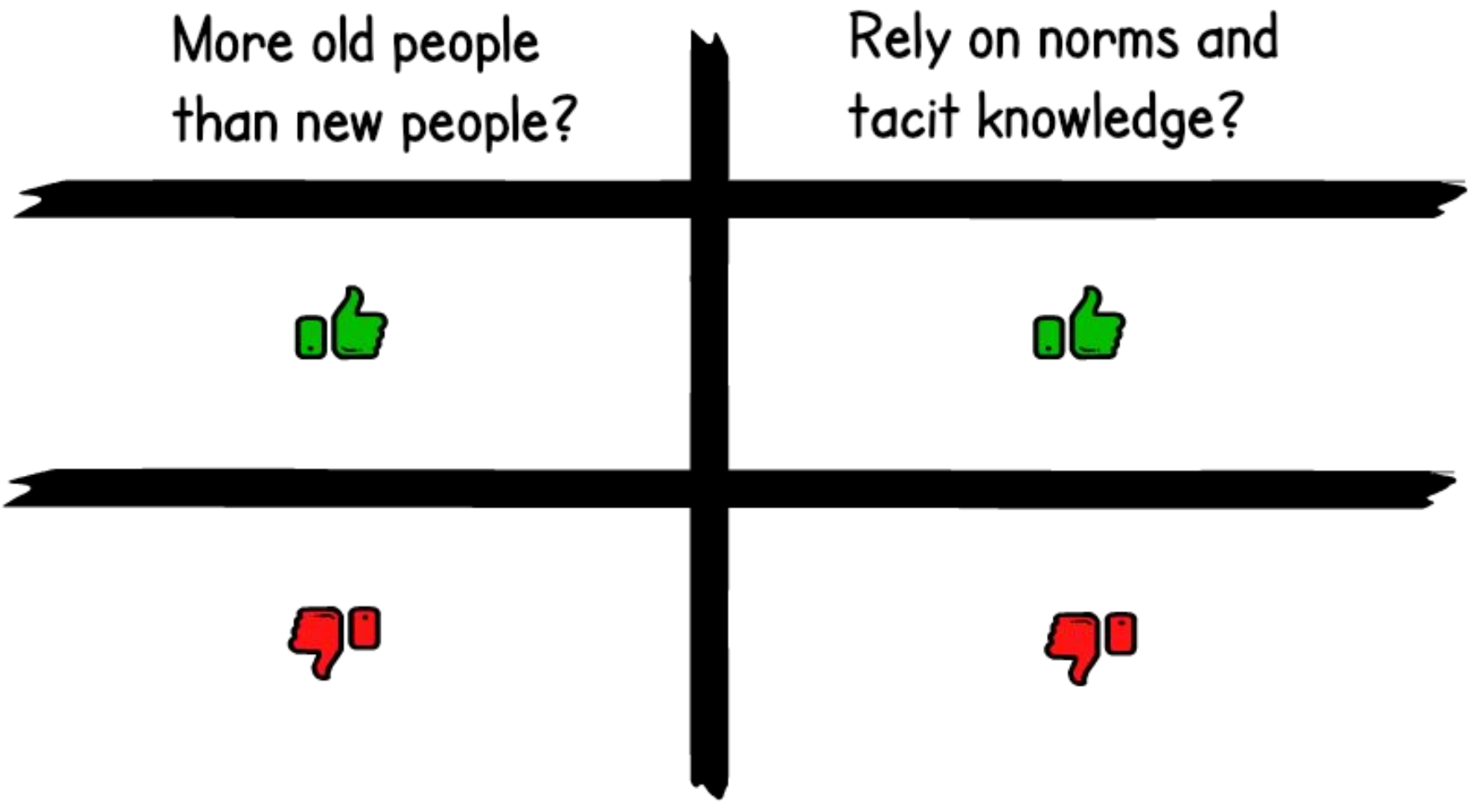
"Aligned autonomy" is not enough



See L. David Marquet and "Intent-based Leadership"

Habits difficult to derive on your own

- Test Driven Development
- Conquer and Divide
- Evolutionary Architecture
- Pair programming


More old people than new people?
Rely on norms and tacit knowledge?

There is no real mechanism to deal with larger organisational architecture issues.

Socio-technical architecture problems

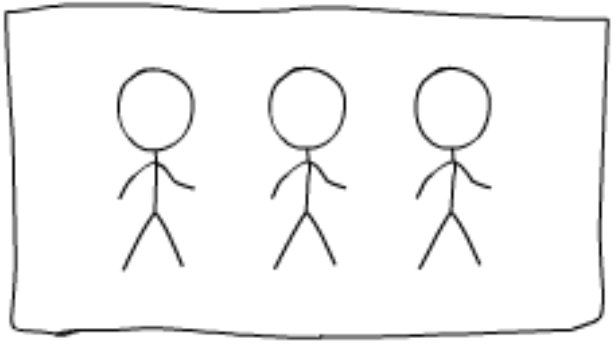
How might we setup an experiment with general Software Engineers with specialisations?

How might we propose an alternate structure for how Squads are typically setup?

How might we establish more consistent, effective practice with larger cross-Tribe initiatives?

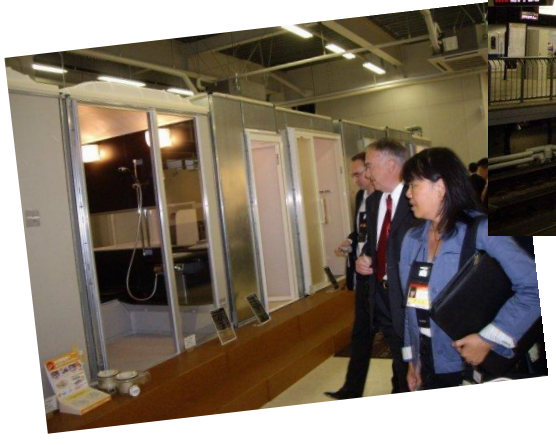
Where might we discuss cross-platform, cross-Tribe, technical architecture patterns and improvements?

Organisational Architecture Guild?



How should one think about all this?

In 2008, I did a Lean study tour in Japan.



http://jchyip.blogspot.com/2008/12/japan-lean-study-mission-summary.html

On the 4th day, we met Takeshi Kawabe (ex-Showa Manufacturing) who recalled a lesson from Taiichi Ohno.

"Stop trying to borrow wisdom and think for yourself. Face your difficulties and think and think and think and solve your problems yourself. Suffering and difficulties provide opportunities to become better. Success is never giving up."

Taiichi Ohno

If there is anything fundamental to Spotify Engineering Culture, it's probably autonomy.

Autonomy means you are free to act.

Autonomy also means you have to face your difficulties and think and think and think and think and solve problems yourself.

Autonomy means you succeed by never giving up.

What doesn't quite work at Cerner and how are you trying to solve it?

Thank you for your attention

First encountered Extreme Programming in 1999.

Joined ThoughtWorks in Feb 2001 (Buildmaster; Java developer; Agile / Lean consultant) (mostly Australia)

CruiseControl committer (retired)

Joined Spotify in Feb 2015 (Agile Coach)

@jchyip (twitter; medium, slideshare)
https://www.linkedin.com/in/jasonyip/
https://jchyip.blogspot.com (old blog)
jyip@spotify.com
https://www.spotifyjobs.com/