

# Pair and Mob Programming

Secret weapon for agile and continuous software development



Thomas Much  
 @thmuch  
#JAXLondon

# About...



Thomas Much

Freelancer, Hamburg

 @thmuch  
#JAXLondon

Agile Developer Coach

Software Developer (Java et al.)

A long time ago in a galaxy far,  
far away.....

*some coworking space*

The other day, in ~~a cubicle~~ next to you.....

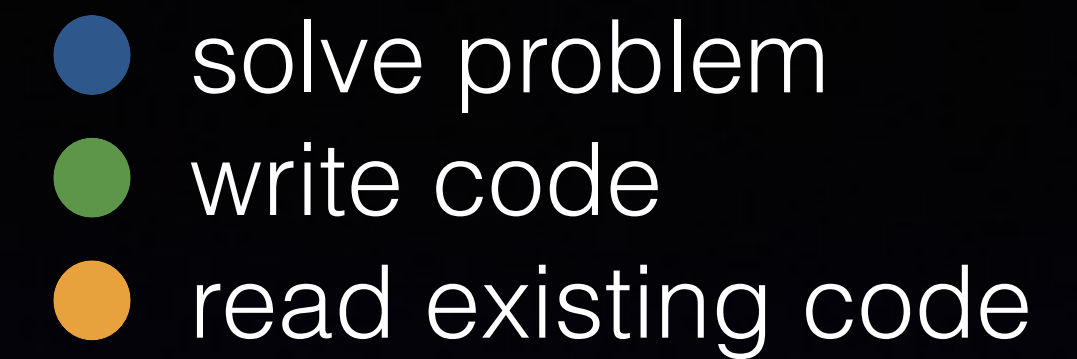
“Woah, who’s supposed to maintain this crap?”

“Who wrote that code?”

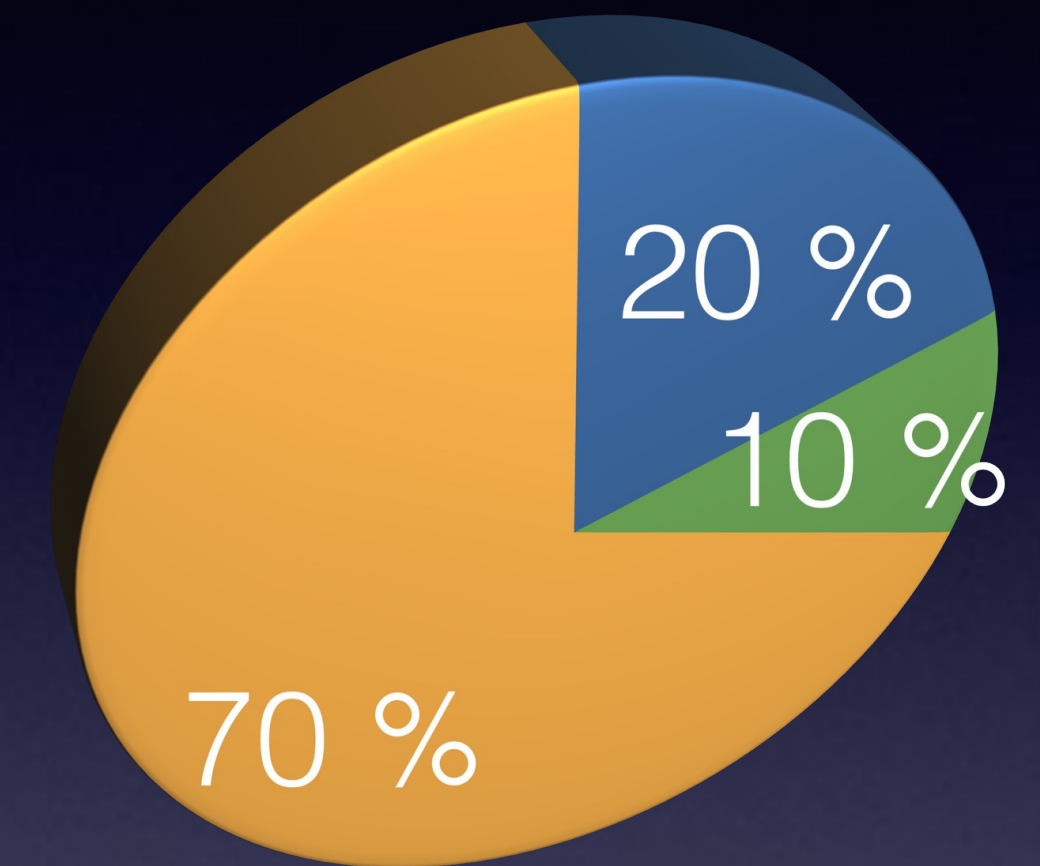
“Oh. That was me.”

“Leave that to <insert name here>,  
he wrote that in his #!@%&\$!?! coding style.”

# Problem: Readability



- We read code a lot more often than we write it
- Understanding code is essential for product care and maintenance!



<https://www.slideshare.net/cairolali/langlebige-architekturen>

- We developers tend to write sloppy code – or too “clever” code
- *Who’s going to give us feedback – before it’s too late?*

# Problem: Simplicity

“Everyone knows that debugging is twice as hard as writing a program in the first place.

So if you're as clever as you can be when you write it, how will you ever debug it?”

– **Brian Kernighan**

*Who protects us from being too “clever”?*



“We’ve got a mandatory code review process!”

# Code reviews?

Honesty of reviews questionable (for systemic reasons).

Wrong incentives.

Feedback too late.

*Who's really going to make major changes then?*

“Developer A is on vacation,  
we’ll get the urgent bugfix afterwards.”

“Developer B has left the company,  
we’ll have to rewrite his apps from scratch.”

“It will take months before newly-hired developer C  
fully understands our project and code.”

# Problem: Know-how transfer

*Missing know-how transfer.*

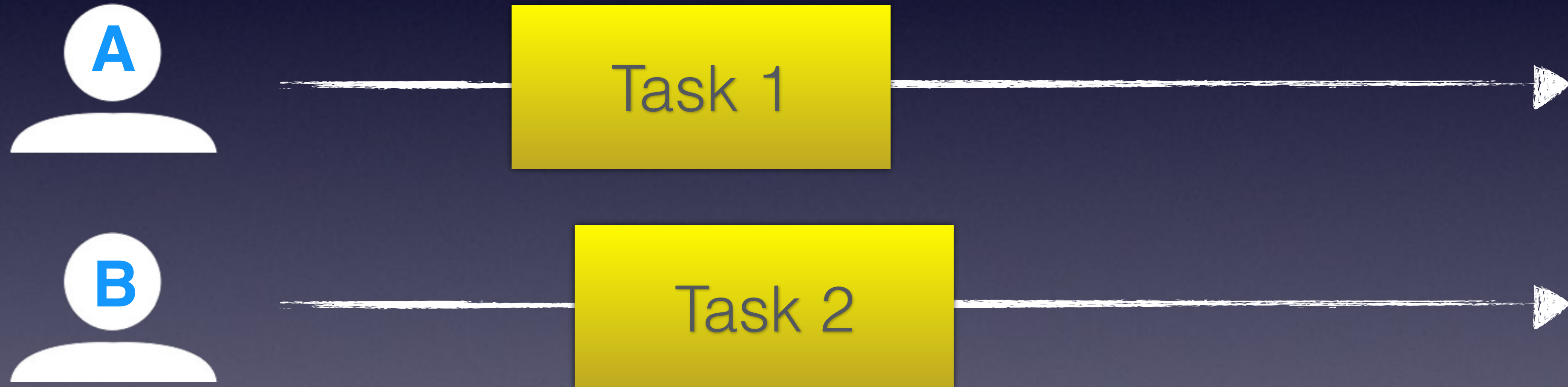
No collective ~~code~~ product ownership.

How? Documentation, workshops, trainings ...

*Are we working together as a team on our product / code?*

“But we *are* a team?!”

# “Team”work

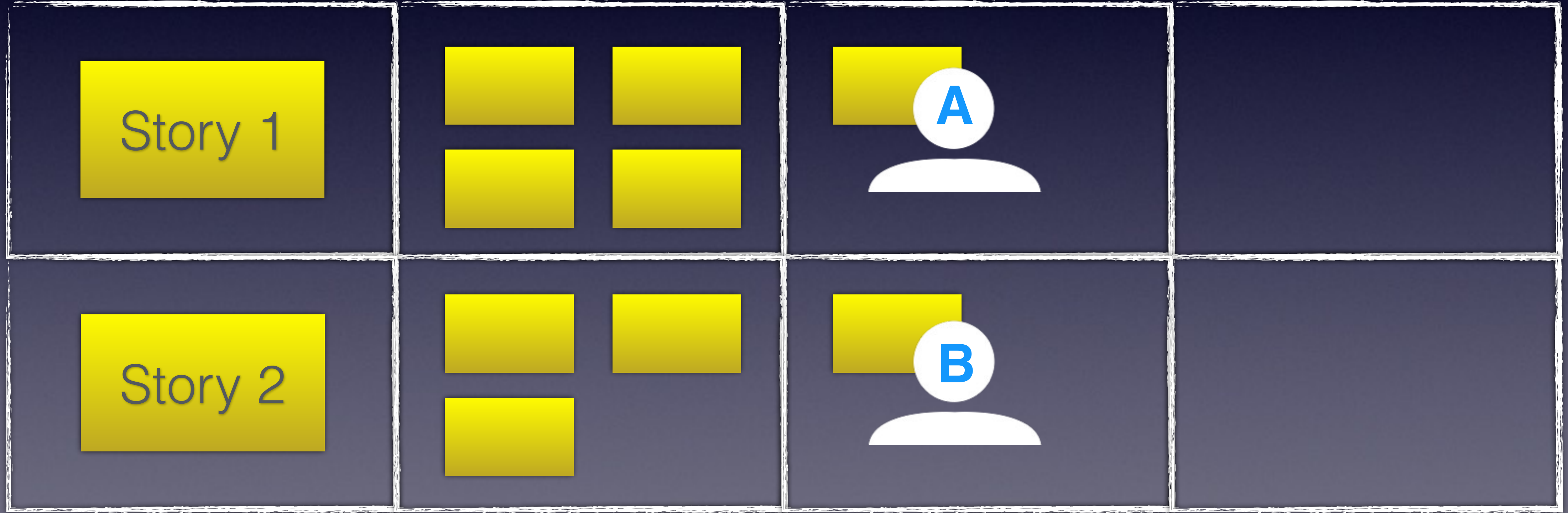


# Solution! “Let’s become agile.”

To Do

In Progress

Done

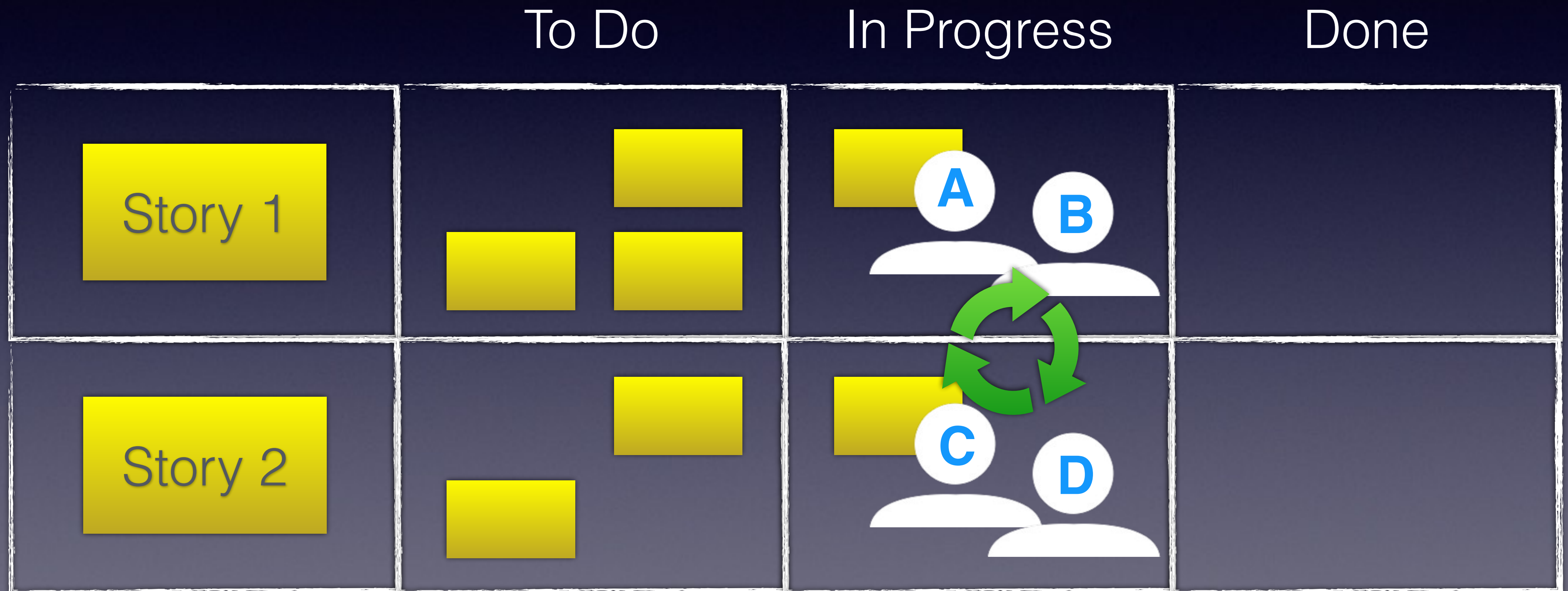


“If your agile Team has individual work assignments,  
I suspect it is neither agile nor team.”

– **Tim Ottinger**



# Real team collaboration



# Problem: Collaboration

How can we really work *together*  
instead of just next to each other?

# Problems!?

Readability / Simplicity / Intelligibility

Maintainability

Know-how transfer / Collaboration

# What do we want to achieve?

Getting things “done” quickly?

(“devil-may-care”, release & run)

Or rather develop maintainable software?

# Maintainable software

*In “my” projects:*

Clients have to / want to maintain software themselves.

Our goal:

Develop maintainable software.

Supported by pair programming.

# Pair programming coaching

Idea: *Actively*

Since 2013: N

E-commerce, E

**OTTO**  
**DEV.OTTO.DE**  
— DEV + OPS BEI OTTO.DE —

[About](#) [Meet Us](#) [GitHub](#) [Jobs](#) [Impressum](#)

## Continuous Delivery

Process Automation and Continuous Delivery at **OTTO.de**

Successful Mountaineering – Reaching the Summit of **LHOTSE** with Agile Software Development

**Continuous Everything: Fast Feedback Driven Development**

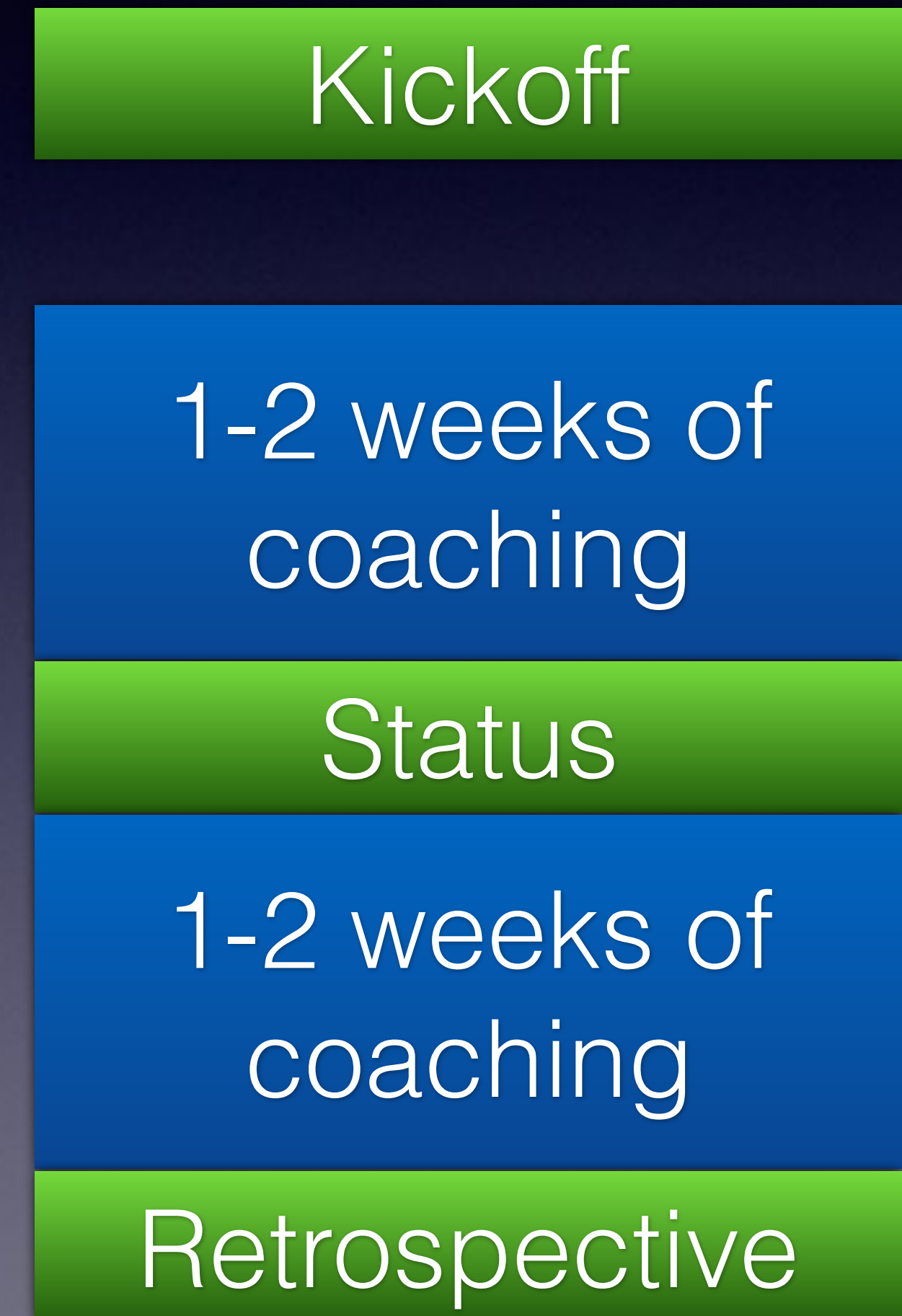
LHOTSE is the internal code name for our project

Fast feedback is a cornerstone of agile software development. When developing the **LHOTSE** project at Otto, we tried to be as agile as possible

The screenshot shows a website layout with a navigation bar containing links for 'About', 'Meet Us', 'GitHub', 'Jobs', and 'Impressum'. Below the navigation is a large heading 'Continuous Delivery'. Underneath, there are three columns of text. The first column is titled 'Process Automation and Continuous Delivery at OTTO.de' and includes a diagram with icons for 'new build', 'deploy', 'test server', and 'live'. The second column is titled 'Successful Mountaineering – Reaching the Summit of LHOTSE with Agile Software Development' and includes a sub-heading 'LHOTSE is the internal code name for our project'. The third column is titled 'Continuous Everything: Fast Feedback Driven Development' and includes a sub-heading 'Fast feedback is a cornerstone of agile software development. When developing the LHOTSE project at Otto, we tried to be as agile as possible'.

# Timetable

1/2 or 1  
sprint



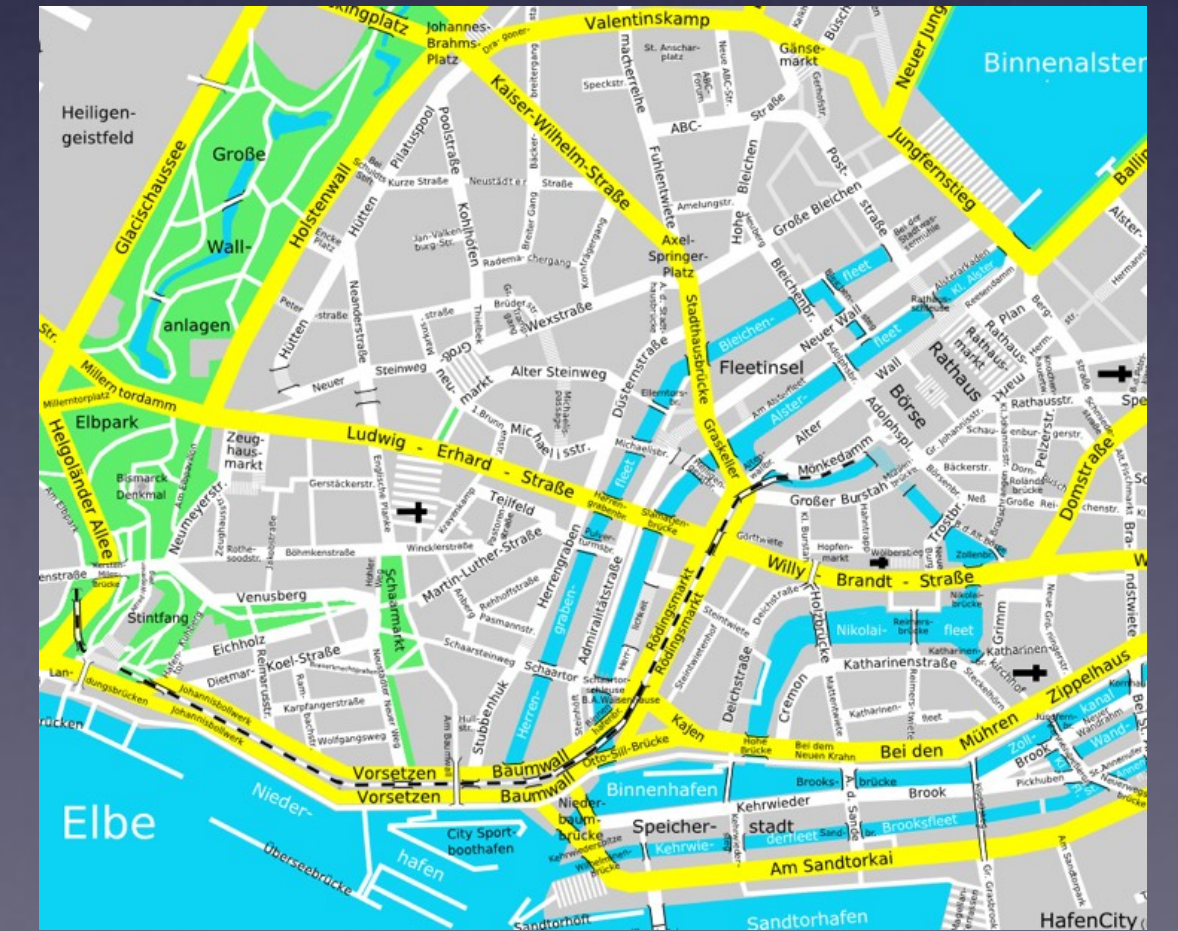
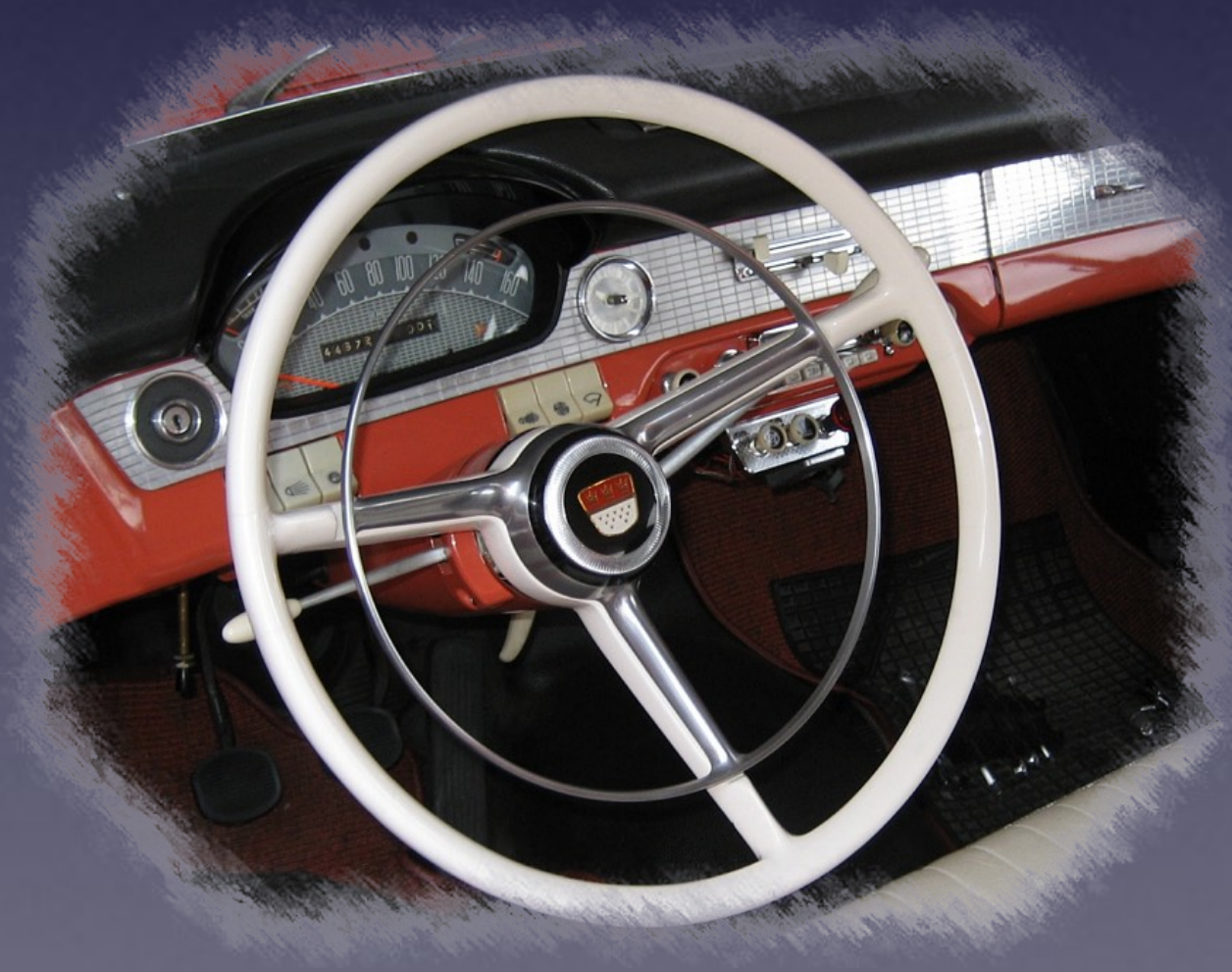
# Pair programming in a nutshell

1  
task

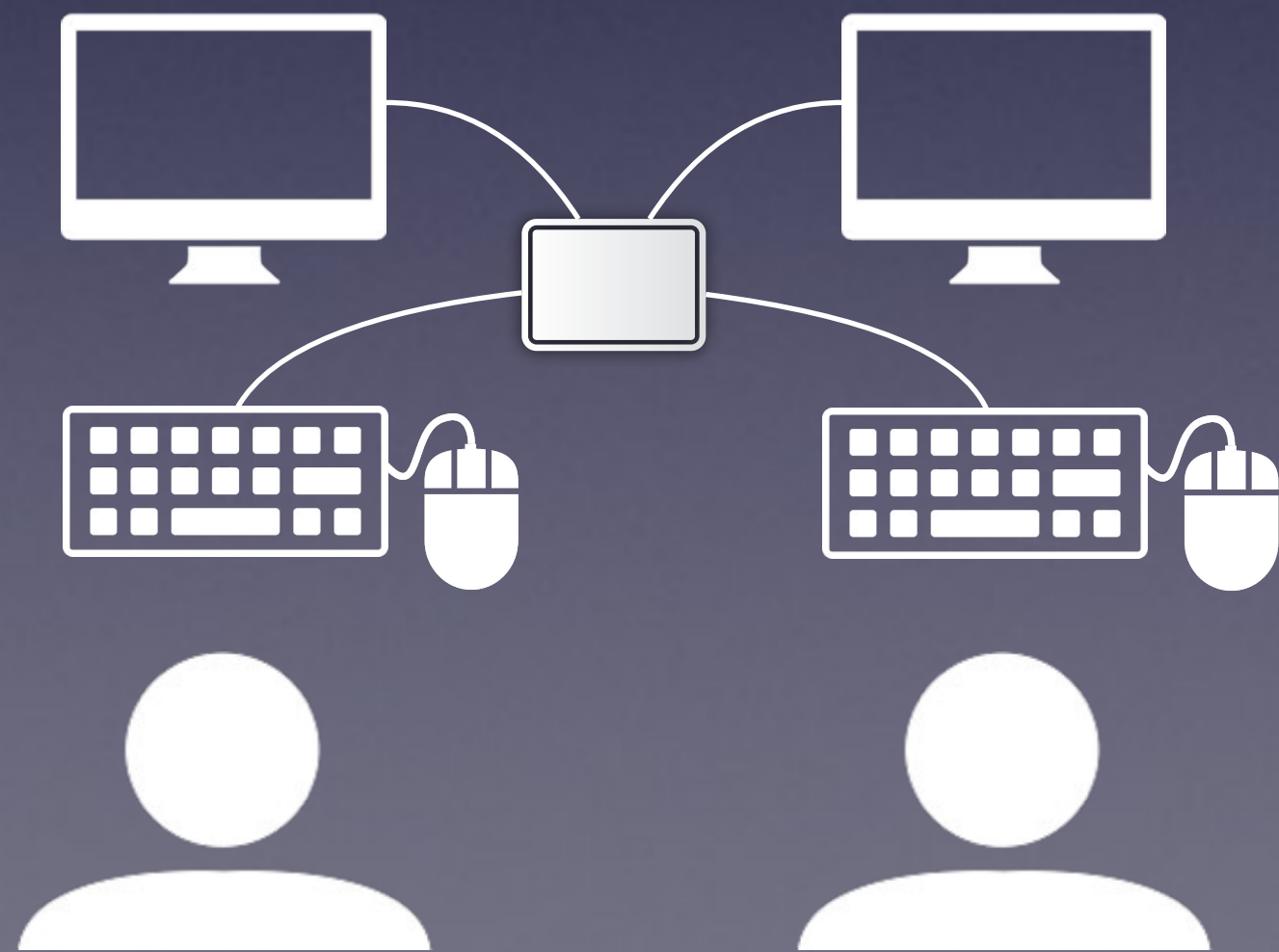
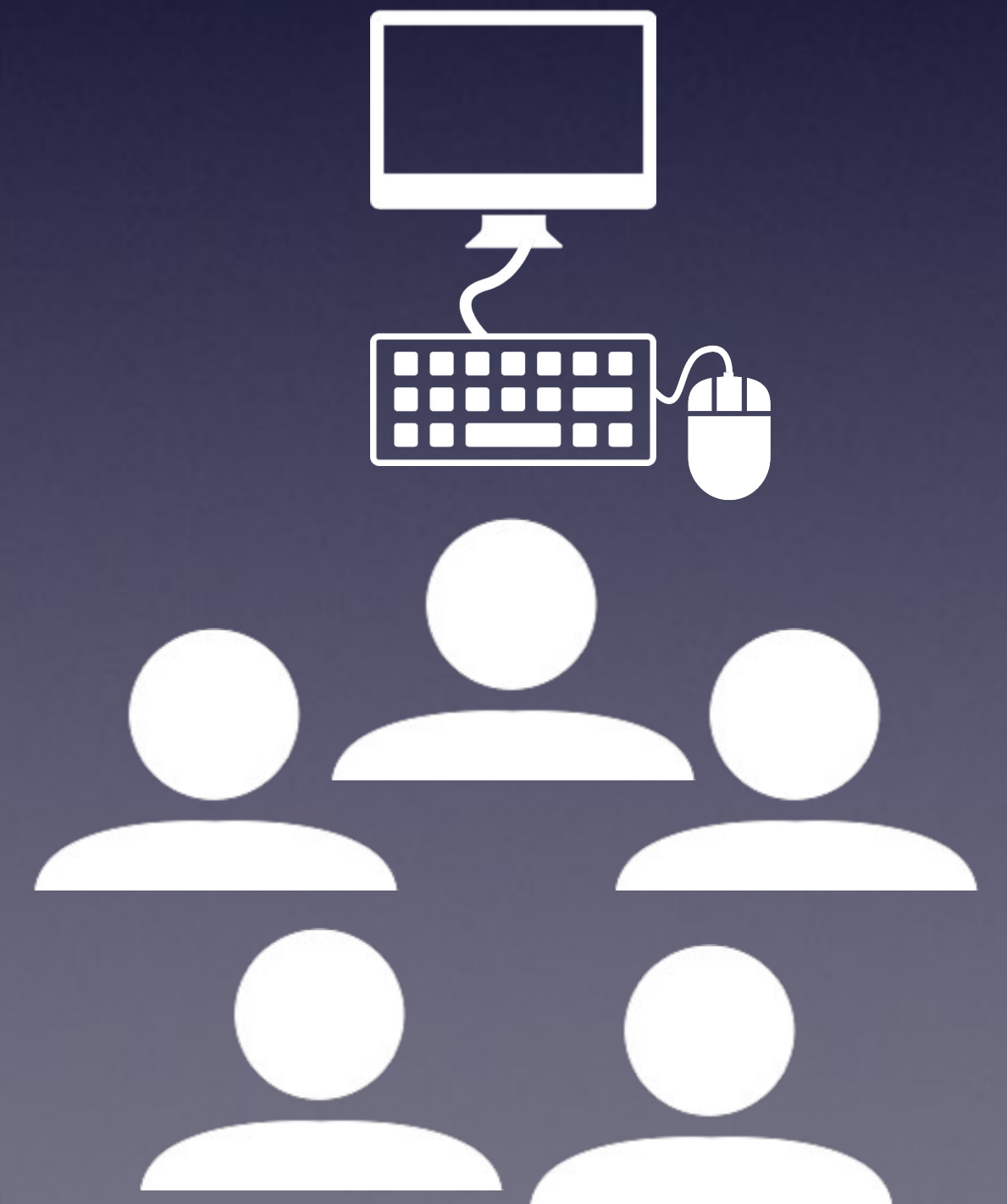




# Driver & navigator



# Variants



# Pair programming – our salvation

Know-how transfer

Collective ~~code~~ product ownership

Clean code

Maintainability

Quality



# Nothing new

Pair programming – ca. 1992? .. 2000 ...

Extreme programming (XP) – ca. 1996 .. 2000 ...

“Flaccid Scrum” (Fowler 2009): Scrum = XP - practices 🤔

# Pair programming is “in”

Boss:

“We’re doing pair programming now.  
You’ll sit in pairs in front of your computers!”

Developer A: “Finally!”

Developer B: “No. Not really. Not again.”

Developer C: “???”

“The other one’s way too fast.”

“The other one’s way too slow and just doesn’t get it.”

“I’m exhausted. Every. Single. Evening.”

“I’d rather work alone.”

# Anti-patterns

Fixed pair works a story.

That story takes 4 weeks or more.

Basically one developer owns the keyboard.

*Variation, relief & creativity are missing completely!*

Small print



# We can't do without exercises

appropriate communication

switching roles

taking breaks efficiently

pair rotation

how to deal with different levels of knowledge

preparation of stories & tasks

# Appropriate communication

silence ↔ too much talking

*As engineers we have to practice communicating with people...*

Driver explains “why”, not “how”.

Navigator does not criticise details.

# Proper pair programming



Proper pair programming is  
**communicating by writing down code.**

*Not just talking about hypothetical code.*

# Why pair programming helps us

*We are subject to certain “**brain patterns**”:*

interpretation

“how” vs. “why”

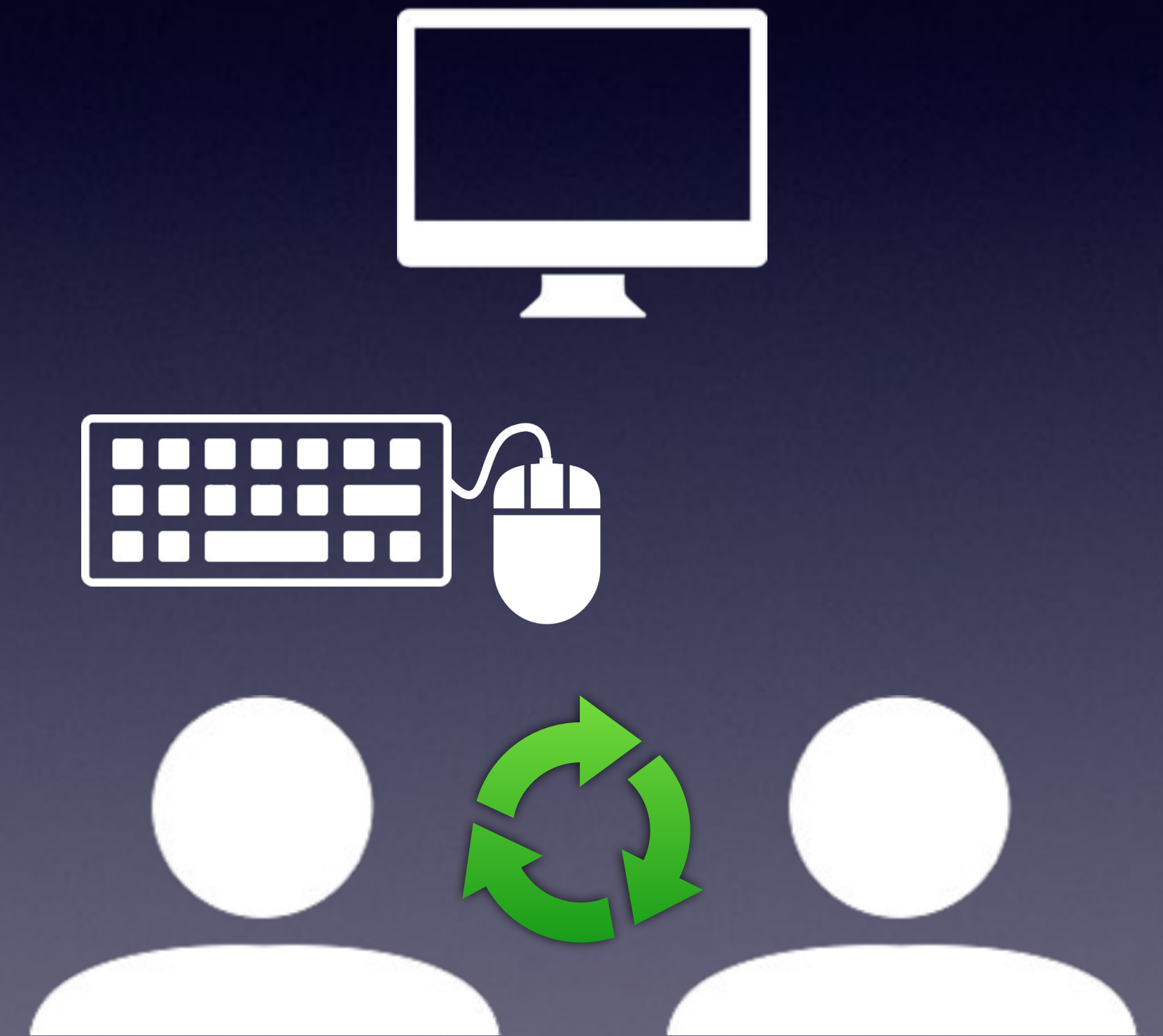
...

<https://www.smidig.de/2015/12/brain-patterns-for-software-development/>  
<https://javabarista.blogspot.de/2016/06/pair-programming-das-gehirn.html>

# Switching roles

- Frequently!
- Every few minutes?!
- *Keeps attentiveness & creativity alive.*

*ping-pong programming*  
*red-green-refactor*  
*TDD*



# Code reviews: ongoing & implicit

*Pair programming = software peer review.*

Timely feedback.

Even for major changes.

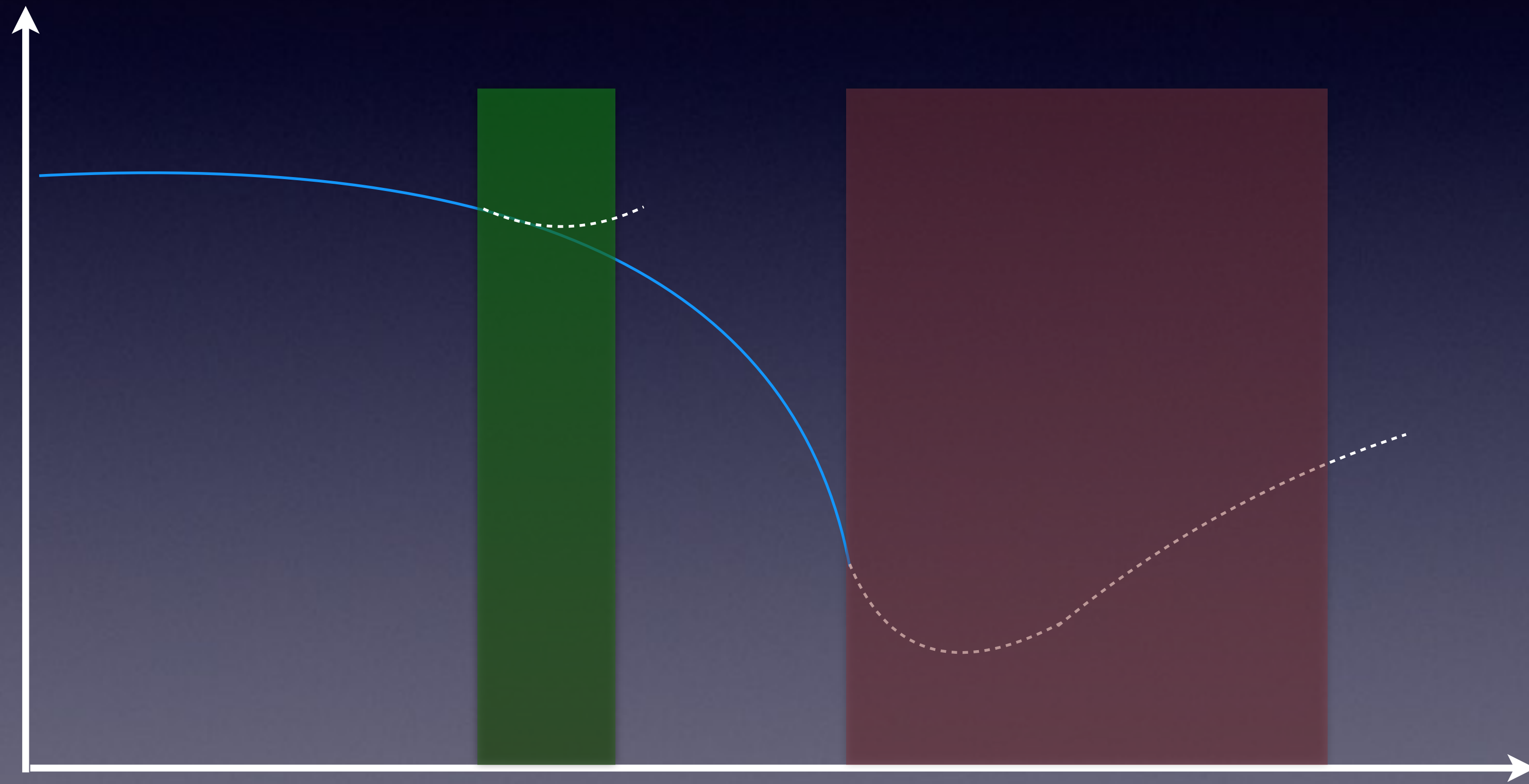
# Explicit code reviews: optional

*No mandatory code reviews when working in pairs.*

(But you can request them if you need another “senior” view.)



# Attentiveness & creativity



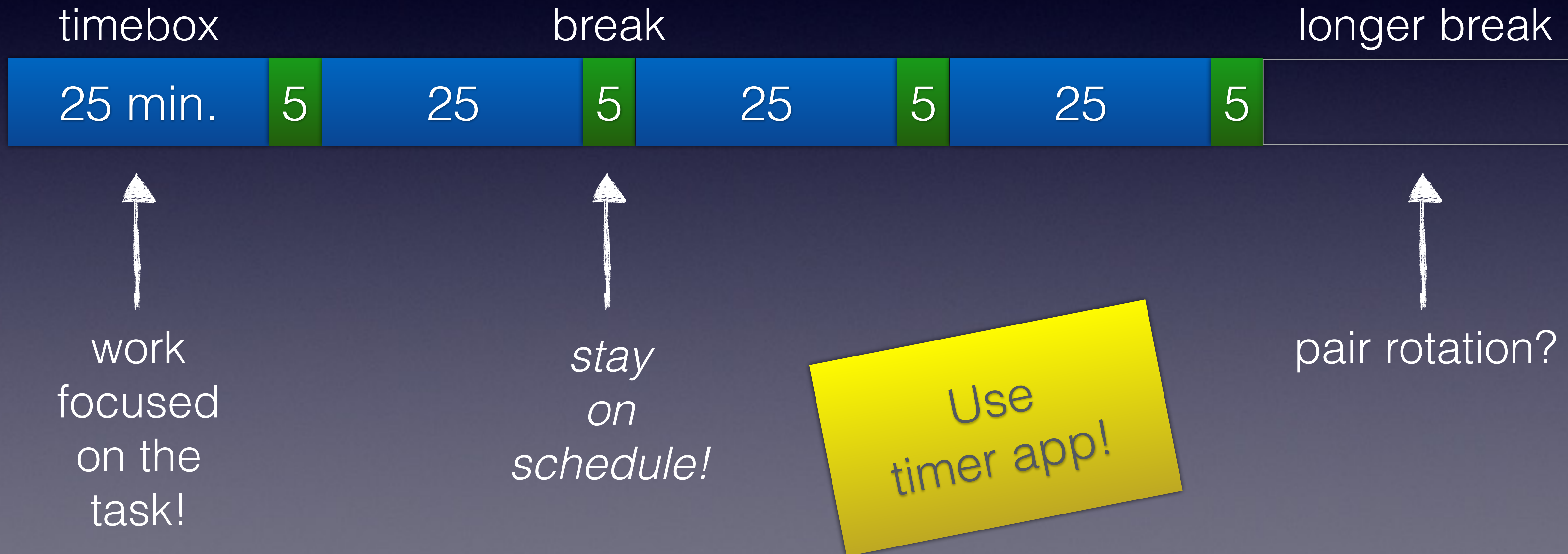
# Taking breaks efficiently

*Before attentiveness decreases too much.*

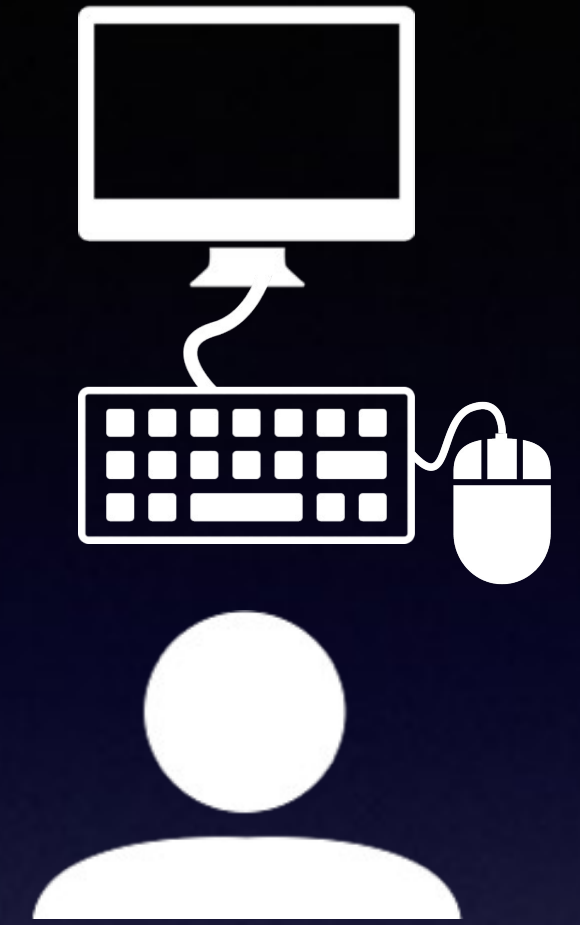
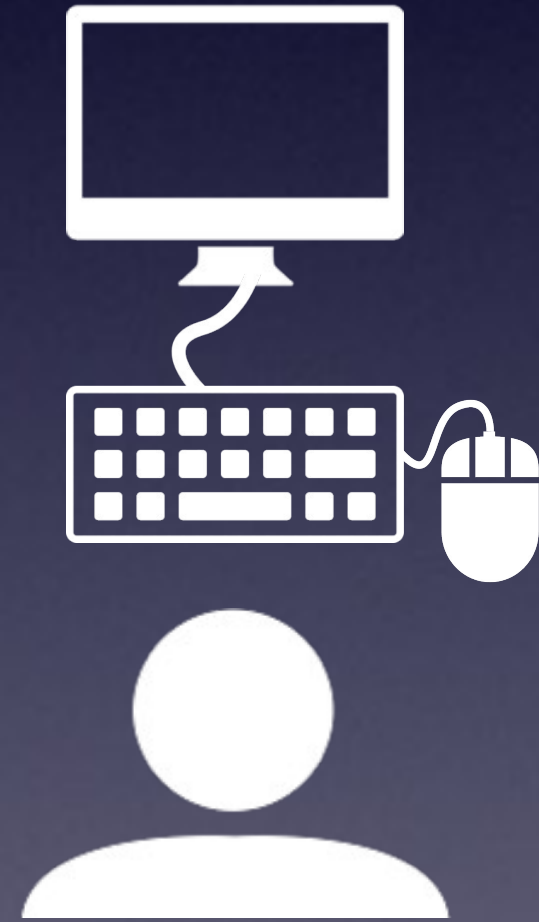
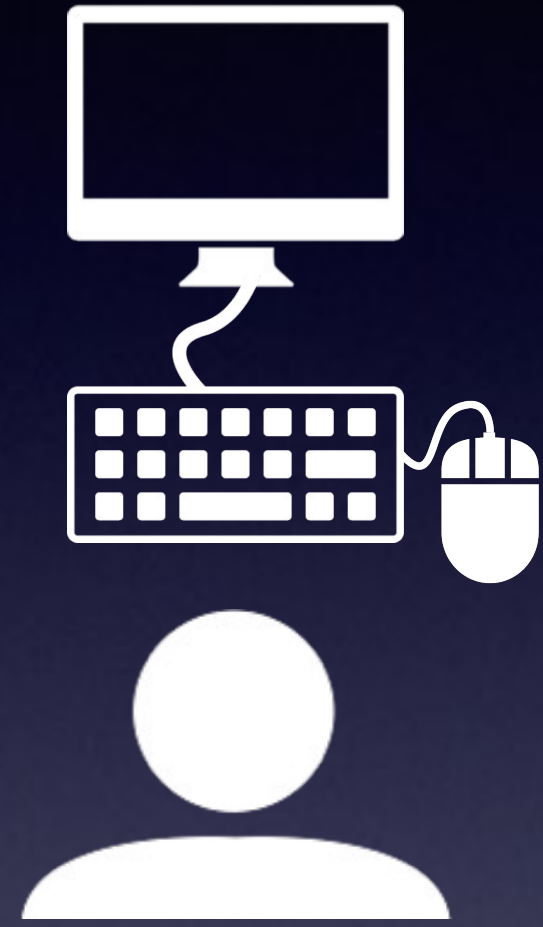
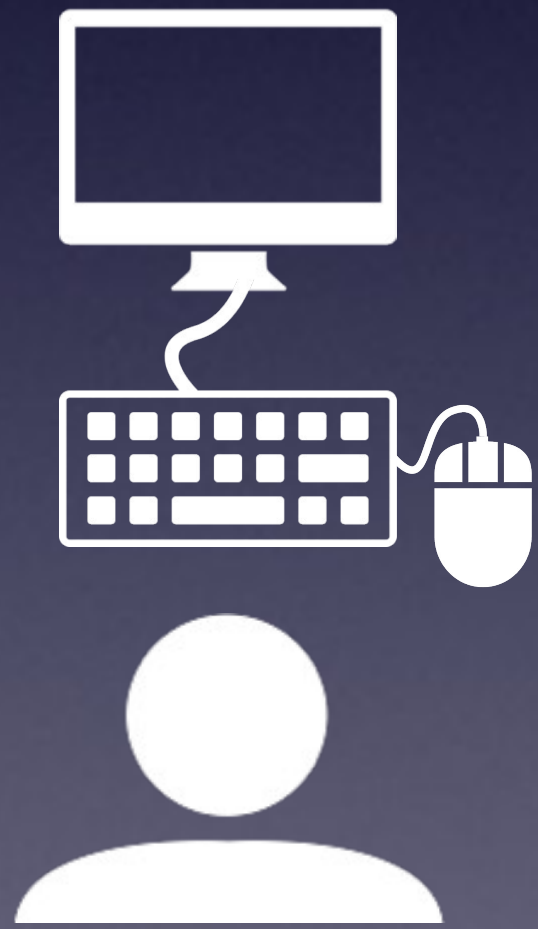
Life hack of choice: “Pomodoro”  
time management method



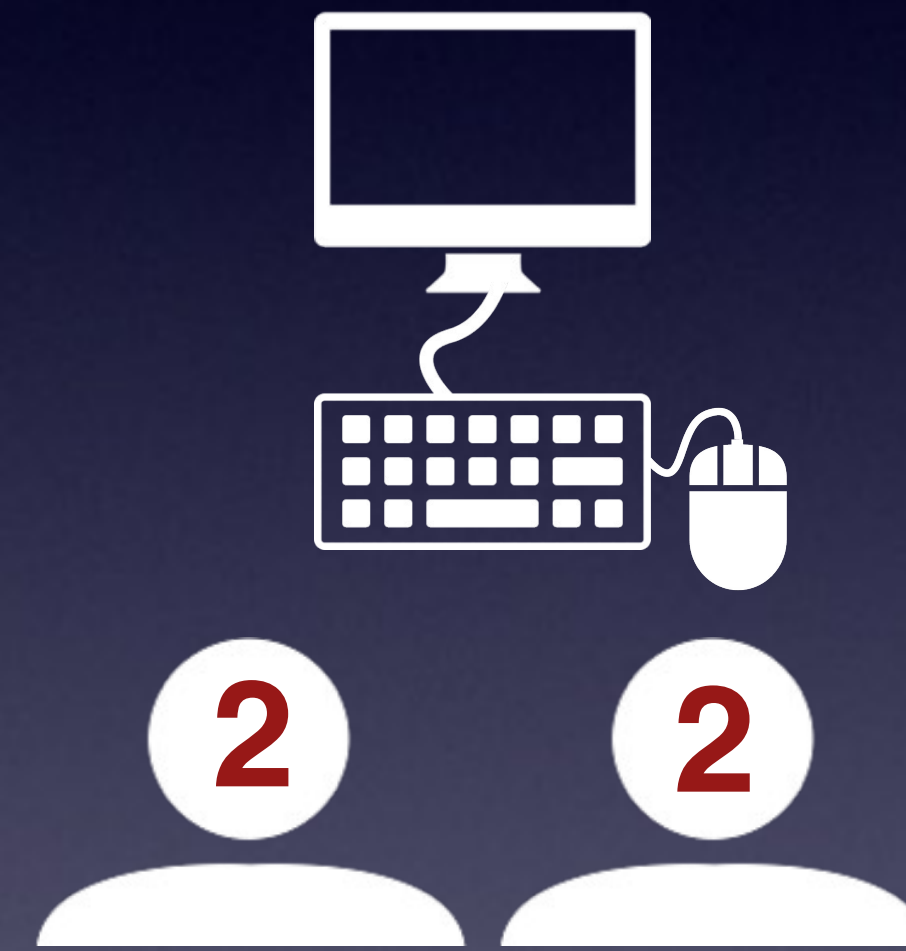
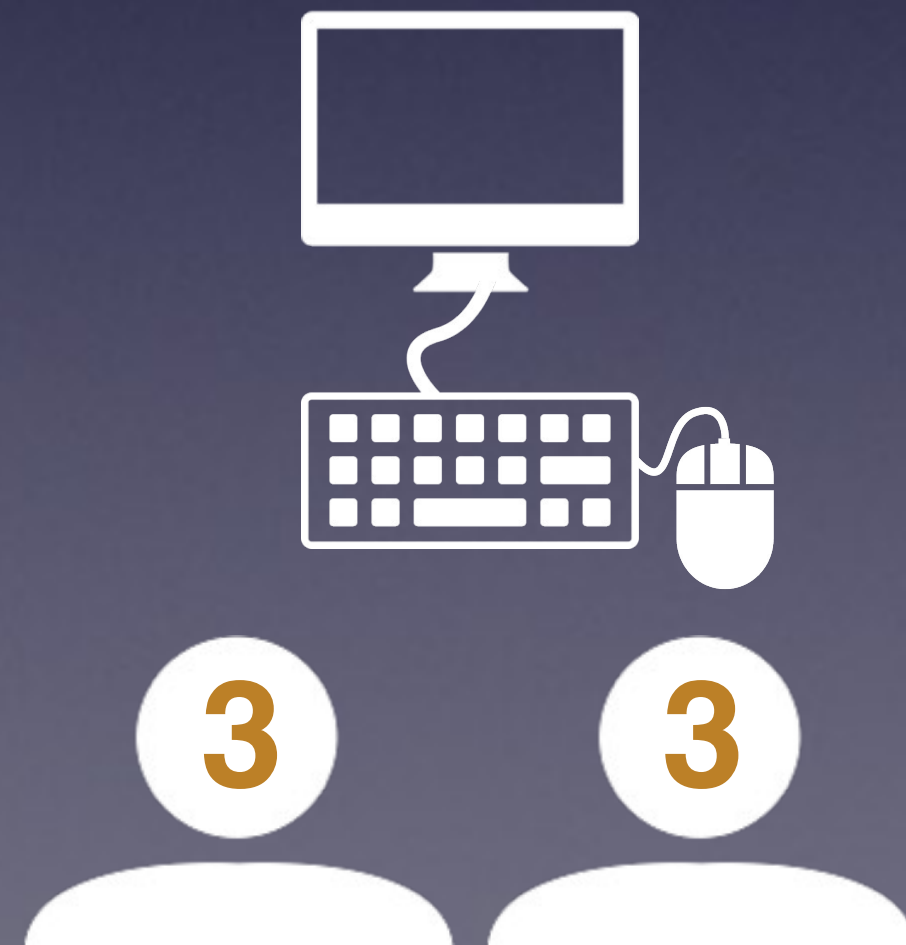
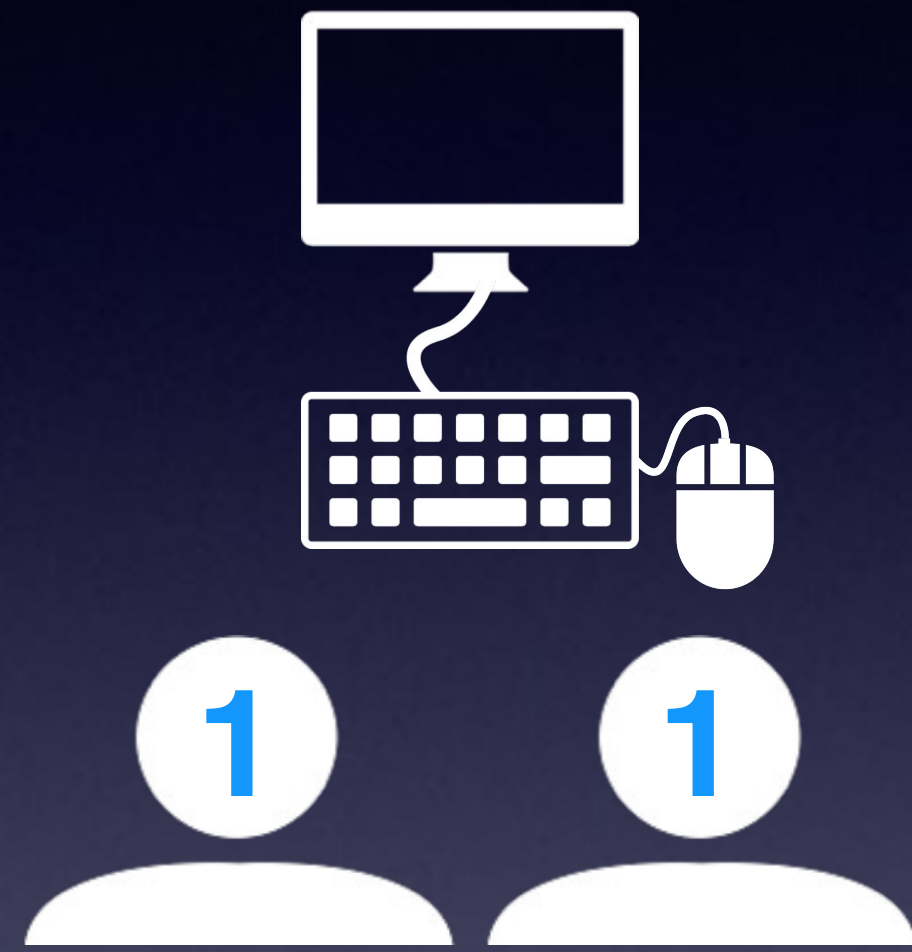
# Taking breaks efficiently



# Isolated knowledge

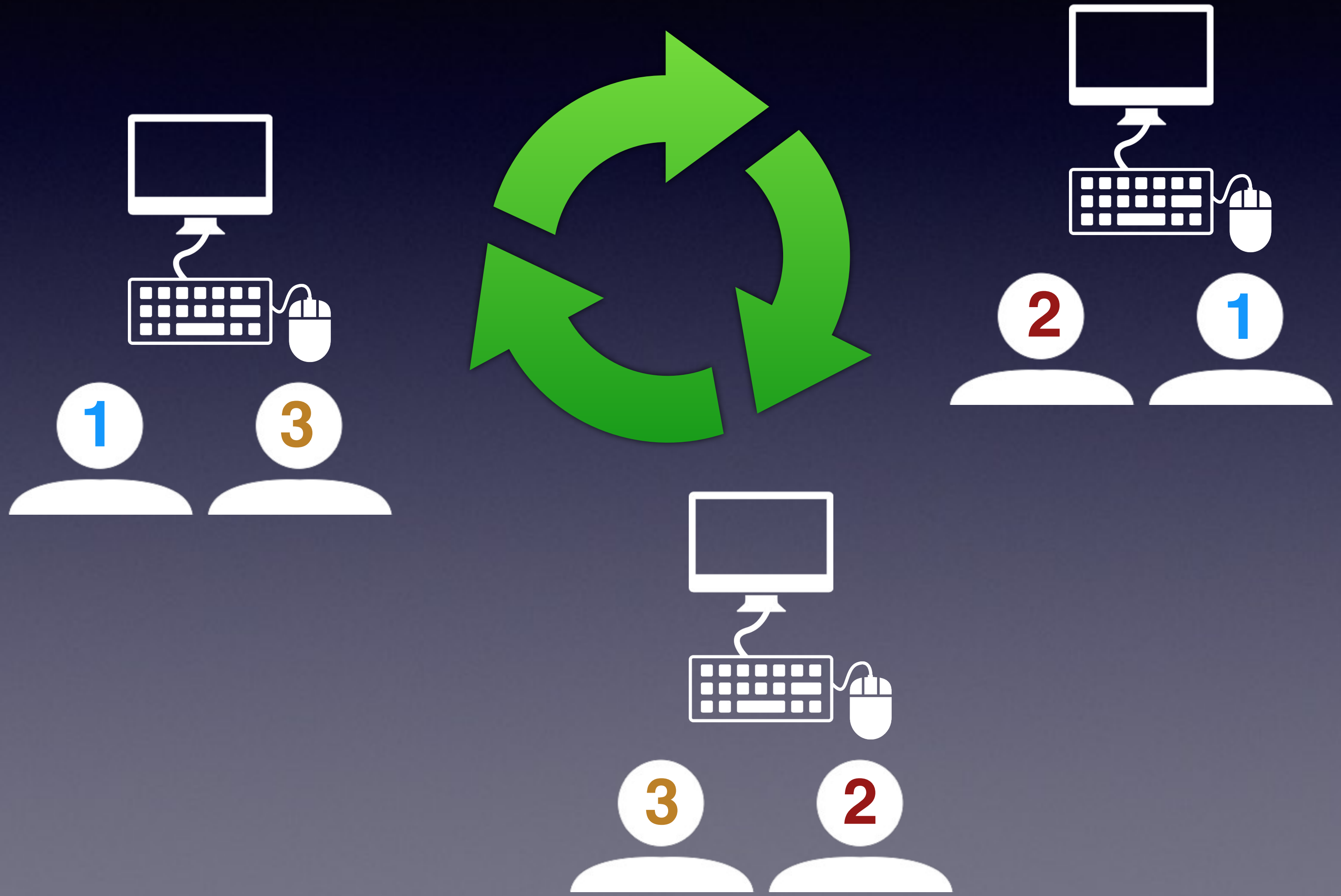


# Isolated knowledge 2.0



At least once a day

# Pair rotation!



# Who with whom?

**All together!**

Sparring partner

expert & expert

Know-how transfer.  
Beginner's mind!

expert & beginner

beginner & beginner

Discover project.  
Reveal weak spots.

# What about the coach?

Coach is an expert (methodically, sometimes technically)

Coach is a beginner (functionally, often technically)

**Realistic collaboration!**

*Acceptance*



# The coach ...

is a pairing partner



watches other pairs

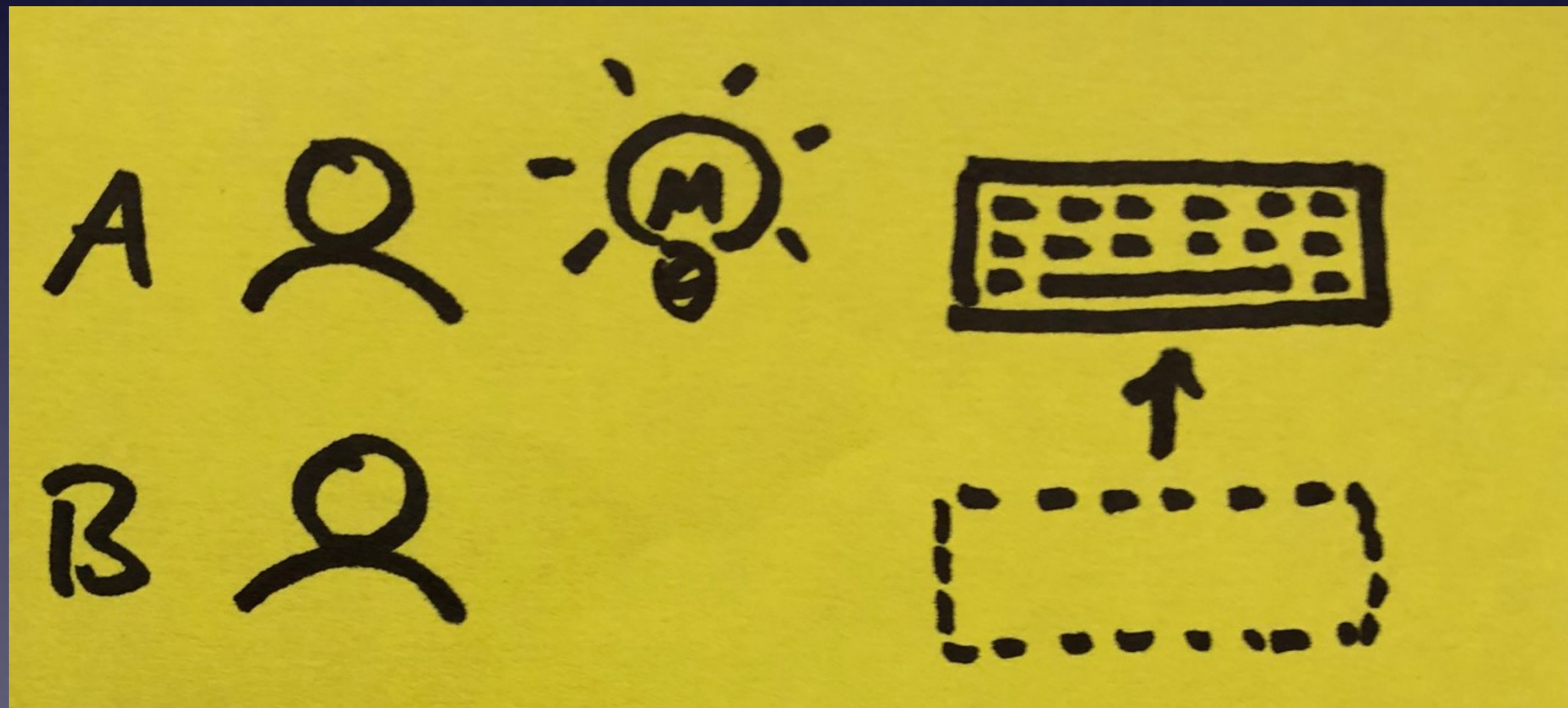


practises together with the team:

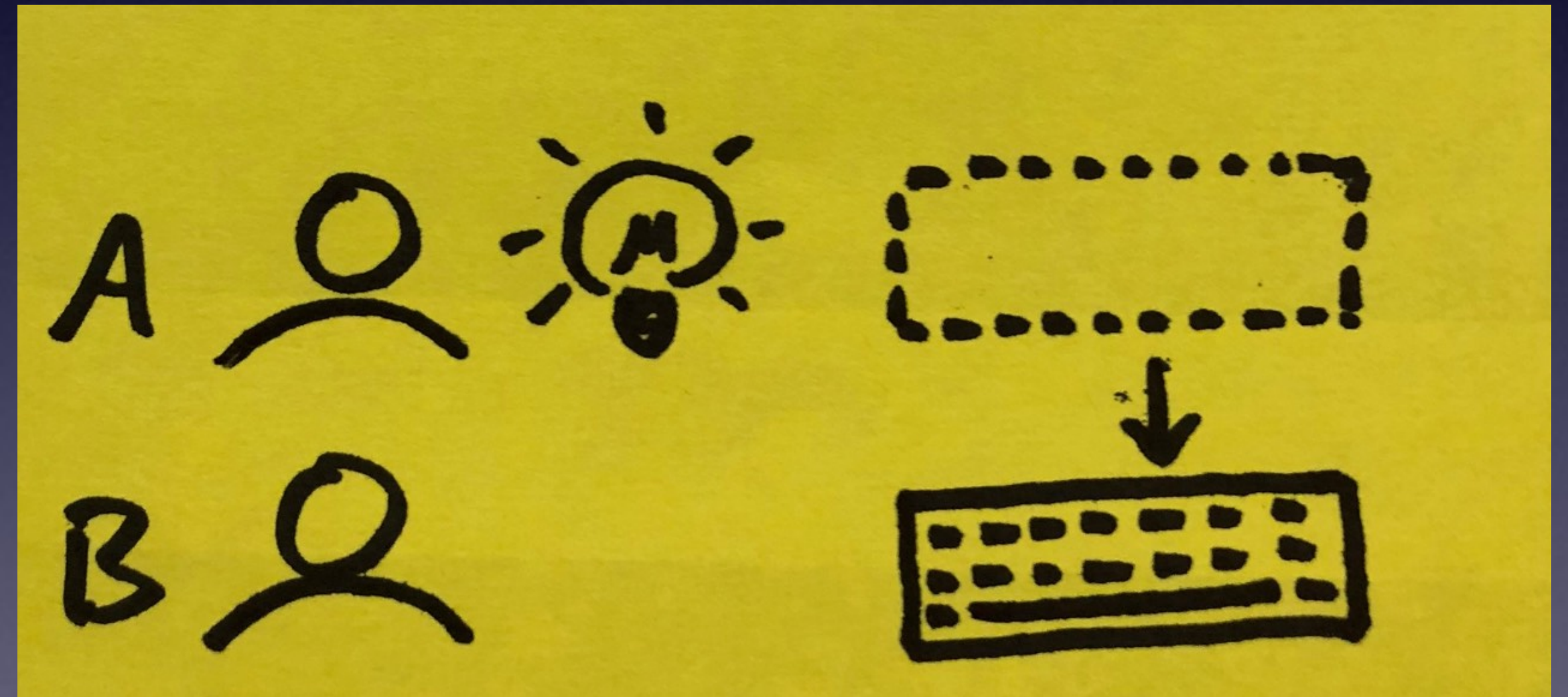
*Switching roles. Pair rotation. Taking breaks. Variants of pair programming.*

# Variants of pair programming

“classic”



“strong style”



@LlewellynFalco



Be an experienced offline (co-located) pair programmer first!

Tools:

*Floobits editor IDE plug-in, AWS Cloud 9 etc.*

*TeamViewer, appear.in, Tuple.app etc.*

Give it a try. Depends a lot on your network (proxies etc.).

# Thorough preparation a must

Joint preparation of suitable, small stories & tasks.

Discovery, planning, ...

*Often, teams see room for improvement  
when doing pair programming.*

# Comprehensive collaboration

Across roles:  
Dev, QA, UX, ...

Pair Doing – “Pair on Everything”

***Change of perspective.***

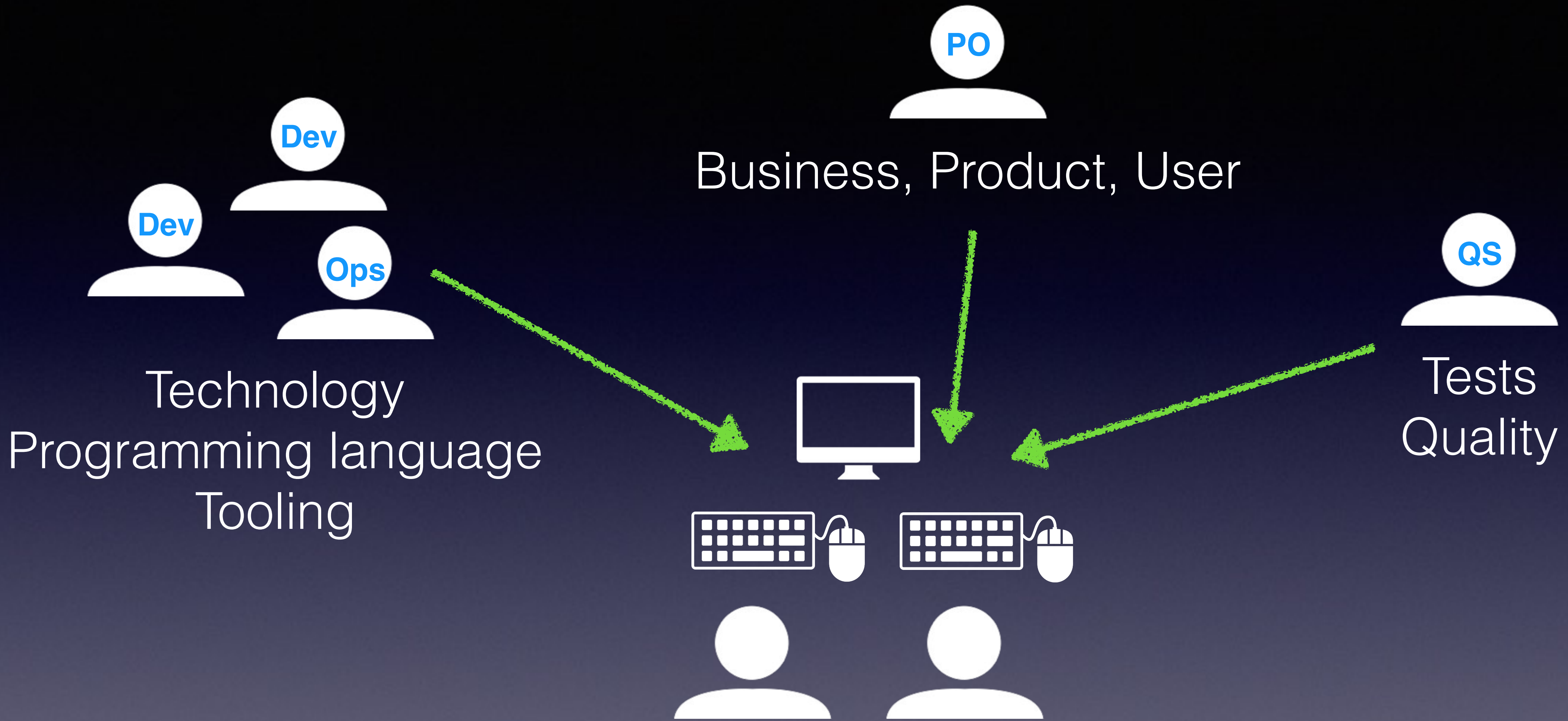


Business  
Product  
User

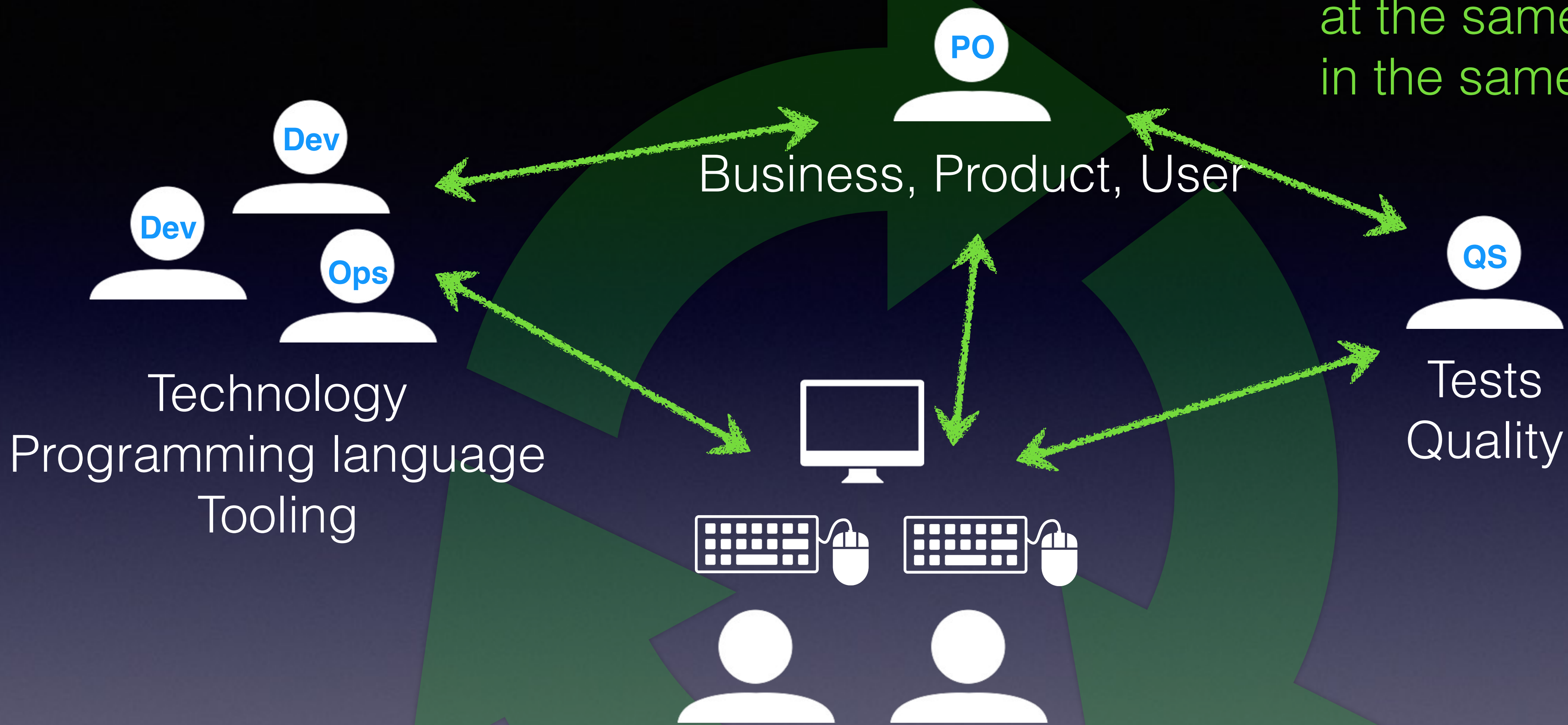
Tests  
Quality

Technology  
Programming language  
Tooling





at the same time,  
in the same space!



# Mob programming



# Mob programming

“It’s about getting the **BEST** (not the **most**) from your team.”

– Llewellyn Falco

“All the brilliant minds working on the same thing,  
at the same time, on the same computer.”

“Continuous Integration of Ideas”

– Woody Zuill

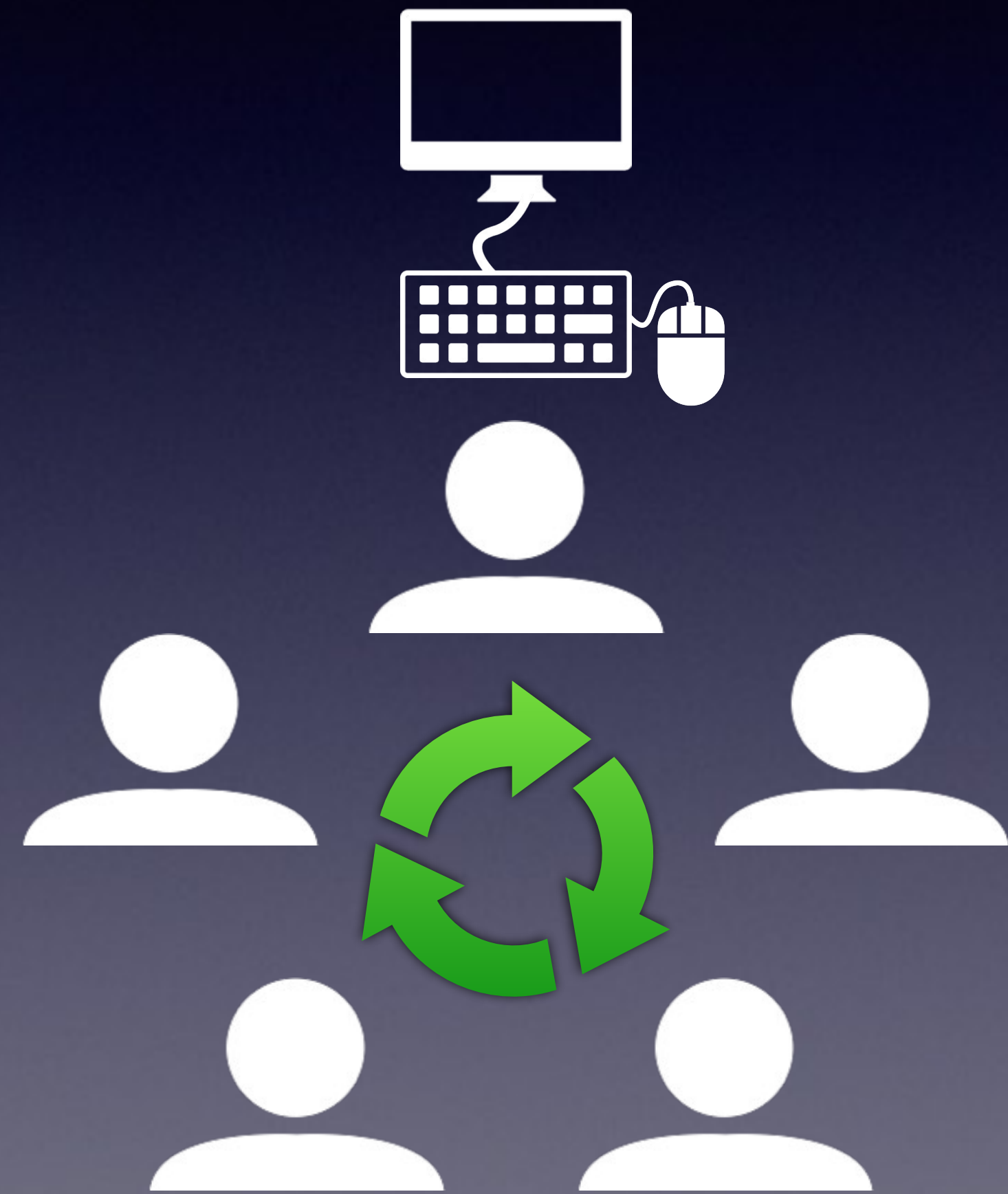
# Mob programming

*Switch roles!*

Fixed timebox  
(every 5-10 min.), <http://mobster.cc>

*Dynamic mob:  
coming and going.*

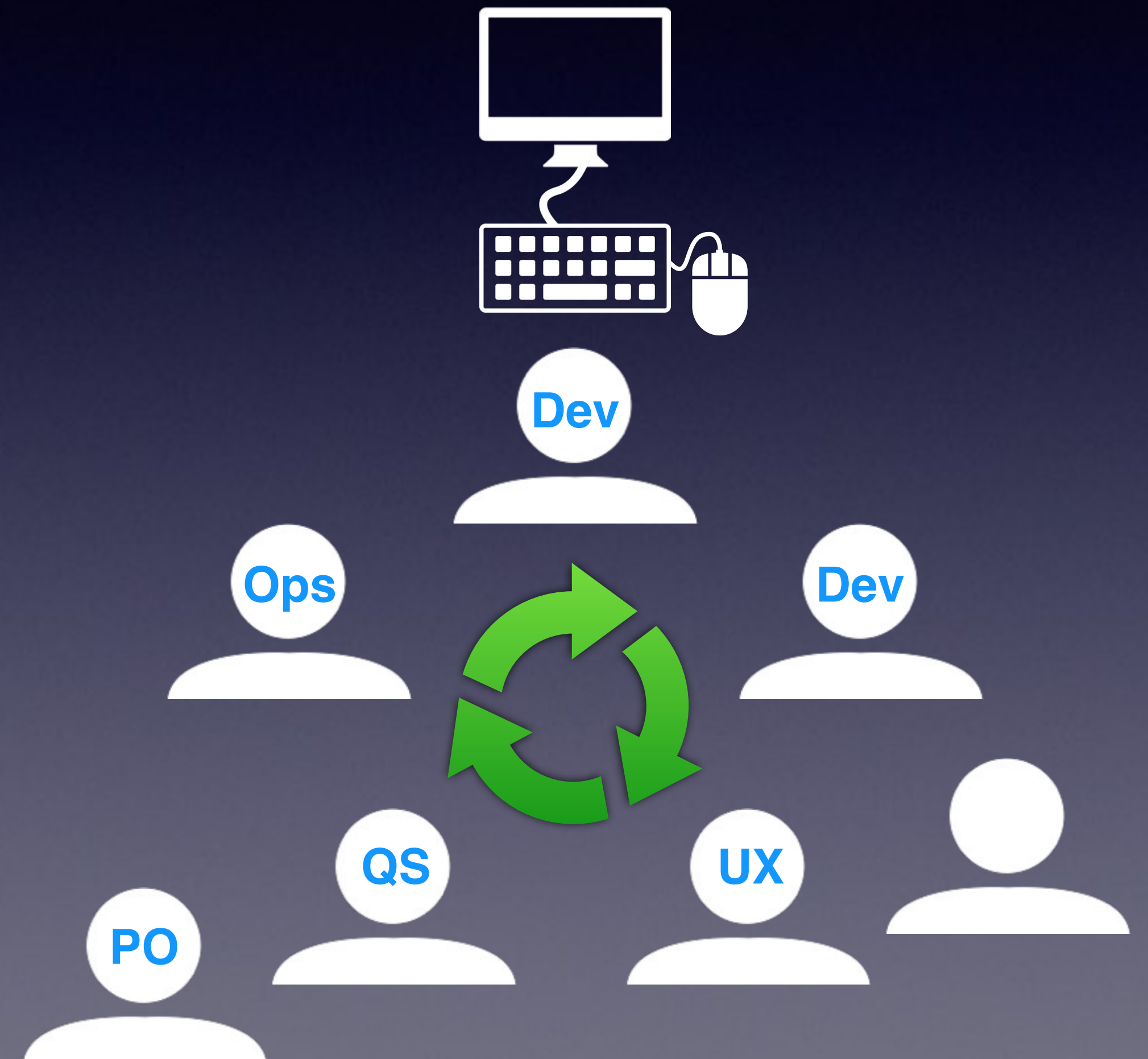
*Feels less cramped  
compared to pair programming.*



# Mob programming

Across team roles!

Getting the **most important** task done first.



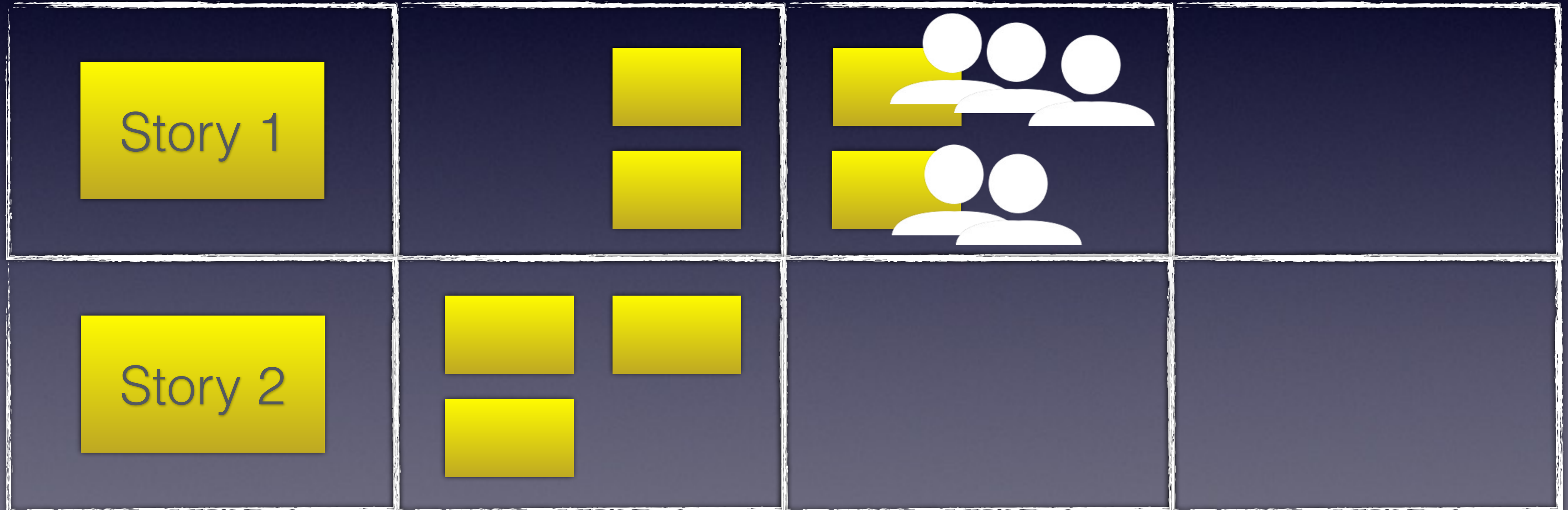
# Highest priority first!

**WIP limit 1**

To do

In progress

Done



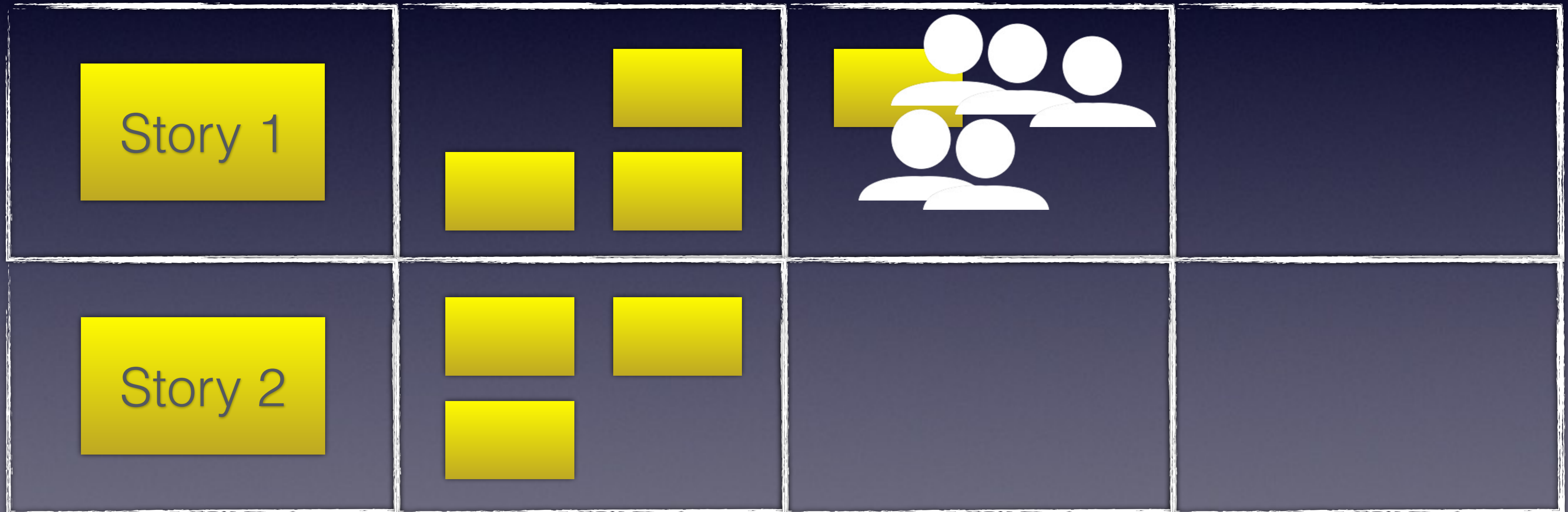
# Highest priority first!

**WIP limit 1**

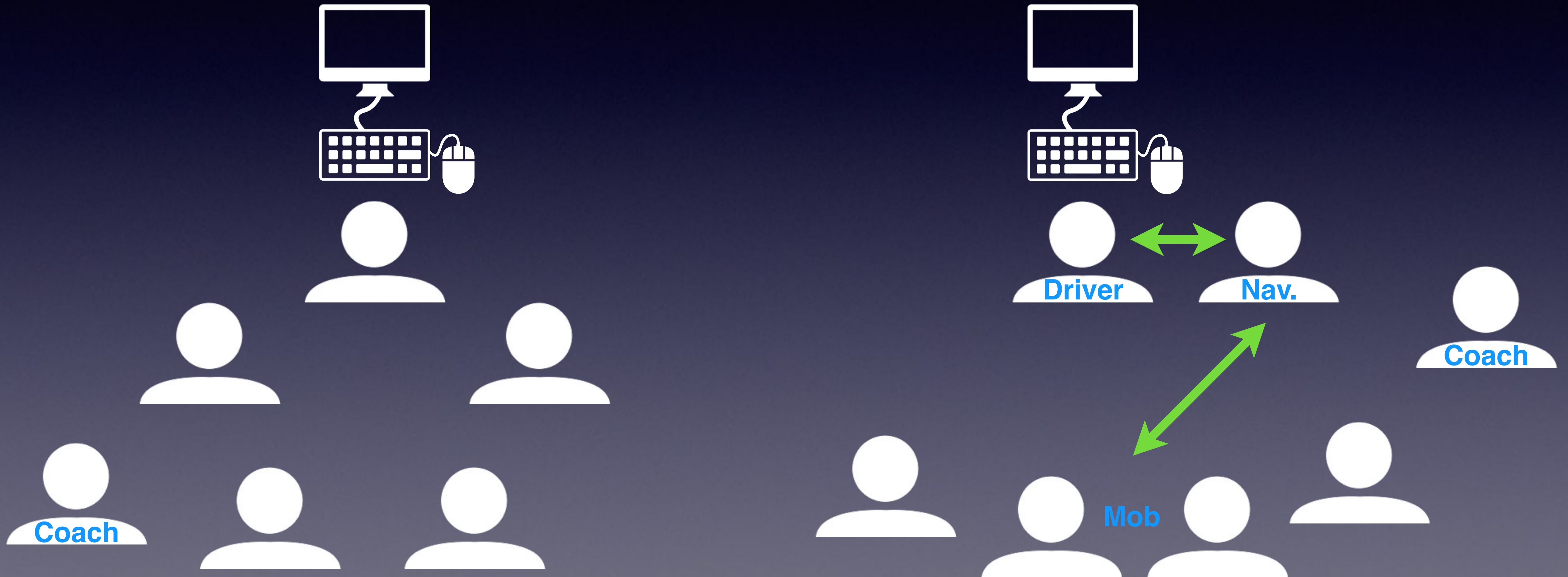
To do

In progress

Done



# Mob programming – setups

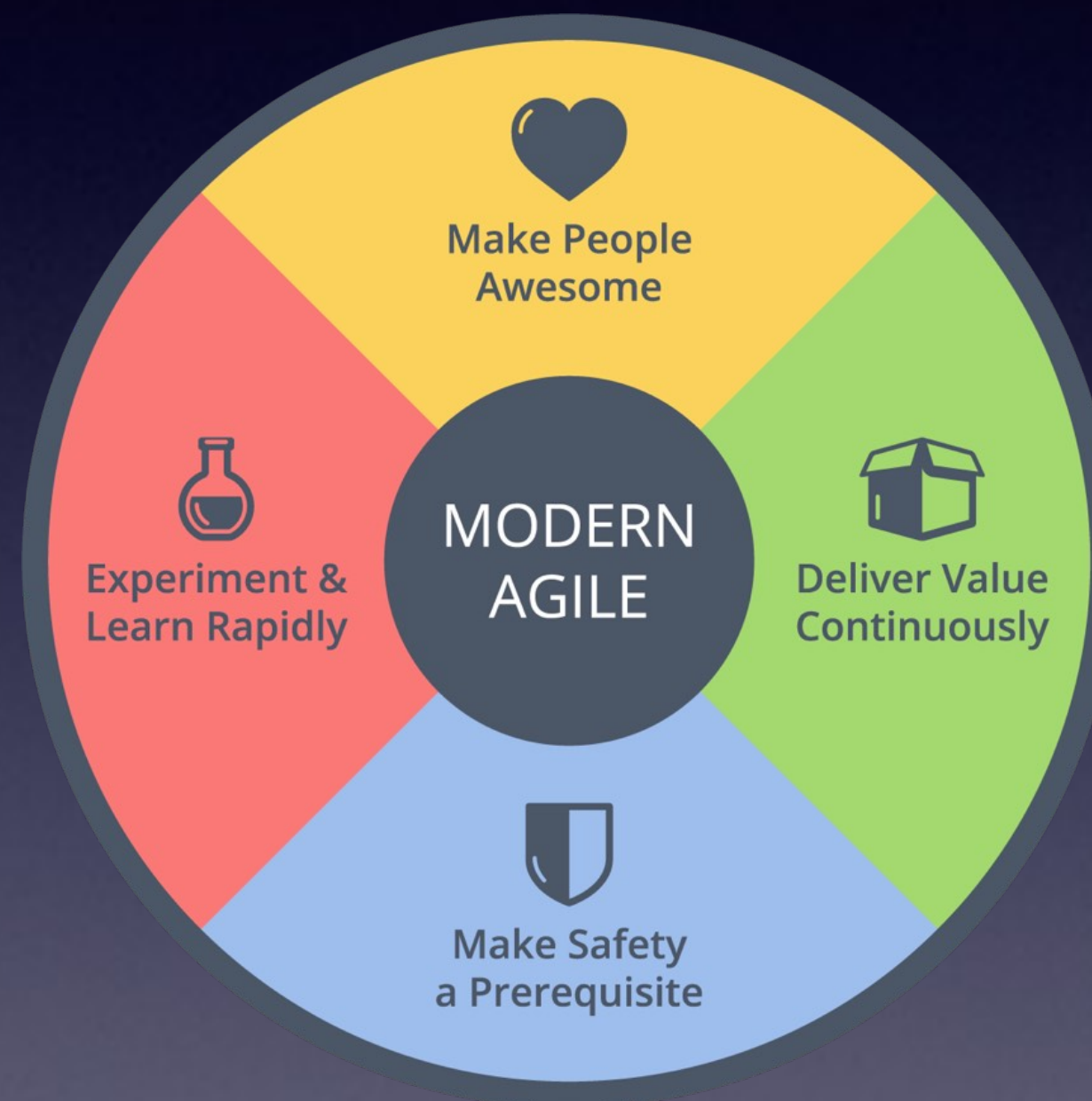


“Mob Programming ... is the most important improvement  
I've seen the last couple of years.”

– **Marcus Hammarberg**

<https://twitter.com/marcusoftnet/status/1042708243544514560>

# Modern Agile



Pair & mob programming are part of it, simple as that.

<http://modernagile.org/>



And still ...

“I’m faster alone.”

# Raise awareness

“If you want to go **fast**, go **alone**.

If you want to go **far**, go **together**.”

– African proverb

# Take care of the details

Many reasons for rejection...

Proponents and opponents must compromise.

**Fix clear agreements.**

*“Short-time pair programming”, for instance.*

*One small step for a developer, one giant leap for a team!*

“Don't think of pair programming  
as 2 people doing the work of one.

Think of it as 2 people avoiding the rework of 7.”

– **Jason Gorman**

# Speed... velocity... pace...

*We follow these principles:*

...

Agile processes promote **sustainable development**.  
The sponsors, developers, and users should be able  
to maintain a **constant pace** indefinitely.

...

*<http://agilemanifesto.org/principles.html>*

# 100% pair programming?

Probably not. But:

**Should be standard programming practice!**

No excuses for not working in pairs!

How much % per day do you pair?

*Much of the real coding time*

Allow for solo time!

For learning something new,  
reading, doing research etc.

*...not working in pairs!*

# Recap

Pair & mob programming strengthen agile processes.

Focus on developer skills & programming practices.

Coaching helps establishing pair & mob programming long-term.

Developers experience benefits hands-on.



Methodical agile coaching – important!

But:

**Don't forget coaching of programming practices.**

**Mob Programming**

**Pair Programming**

**Know-How Transfer**

**Coaching**

**Pomodoro**

**XP**

**Questions?**

**Readability**

**Modern Agile**

**Simplicity**

**Strong Style Pairing**

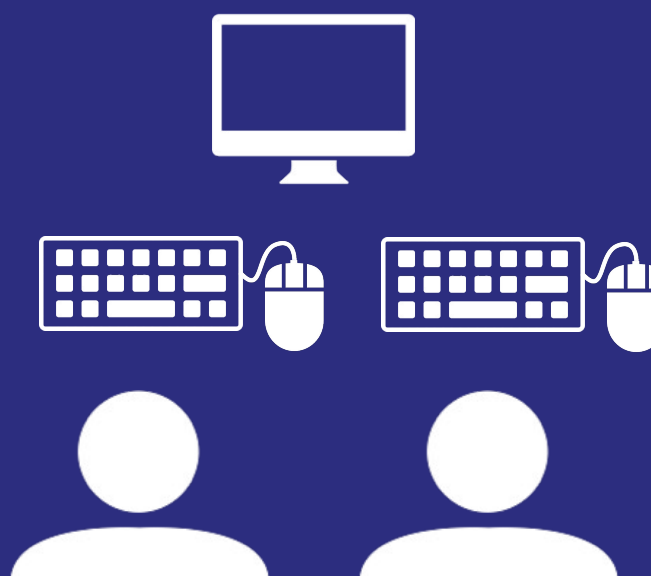
**Velocity**

**Speed**

**TDD**

**Collective Product Ownership**

**Jax** *LONDON*



Thank you!

[thomas@muchsoft.com](mailto:thomas@muchsoft.com)

[www.javabarista.de](http://www.javabarista.de)

 @thmuch