The Kiev Experiment

Evolving Agile Partnerships

Who are we?

- Simon Ogle
- Alexander Kikhtenko
- Peter Thomas

Where did we start?

- Existing monolithic mainframe application
- Quarterly deliveries
- 6 week testing cycles
- Offshore delivery teams
- Waterfall process
- Command and control

What did we want to do?

- Belief there was a better way to deliver software
- Incremental development to deliver business value quickly
- Address the rapidly changing business landscape with flexibility in delivery
- Build quality into the solutions
- Deliver the software rapidly, but in a cost effective manner
- Put the fun back into software delivery

- But the rest of the organisation very sceptical about our delivery approach

How did we start?

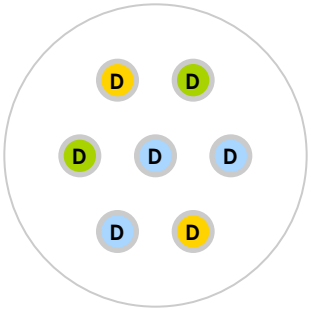
- Started with a single team in London
- Initially focused on building the tools, proving the processes and technologies
- Release 1.0 on Friday 13th October 2006
- Very soon we started to look how we could scale

Where are we now?

- 5 years into the delivery
- 2M trades per day
- 100 billions settling per day in all major currencies
- 40 exchanges across EMEA and APAC
- 15 scrum teams/120 people
- Teams in London, Kiev, Hyderabad, Hong Kong and New York
- 9 application components
- Production releases every 2 weeks

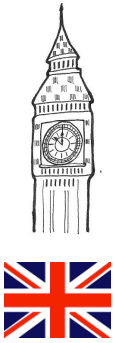
Evolving the team

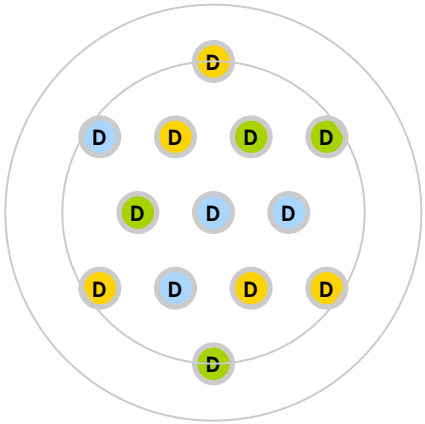
Evolving the relationship



2006

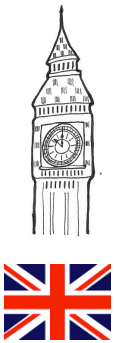
small co-located
team

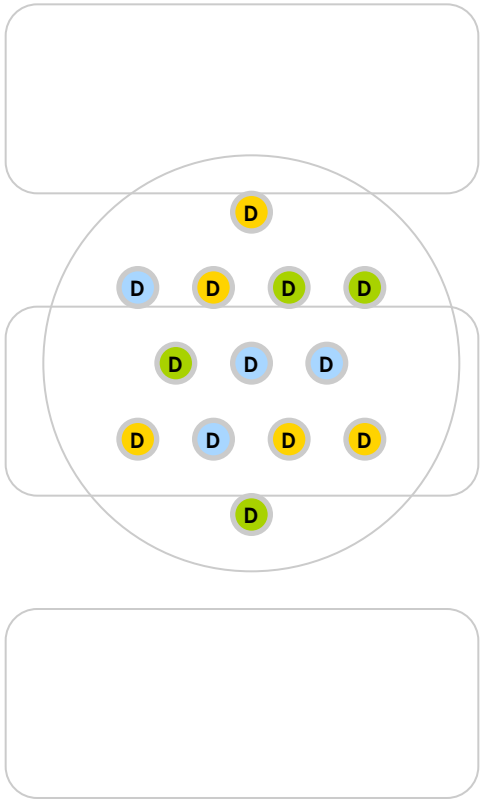




2006 2007

small co-located team





2006

small co-located team

2007

component teams

2006 2007 2008 2009

small co-located team

component teams

co-located feature teams

dispersed feature teams

2006 2007 2008 2009 2010

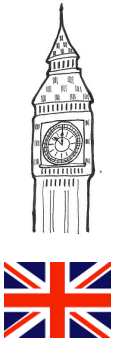
small co-located team

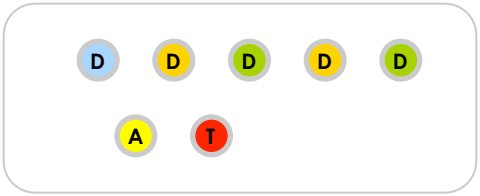
component teams

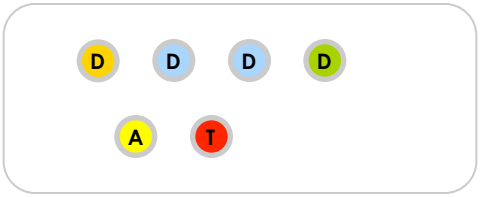
co-located feature teams

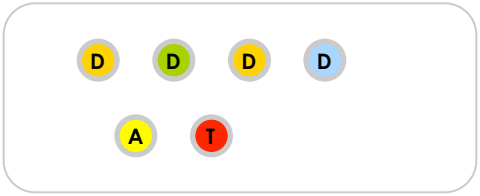
dispersed feature teams

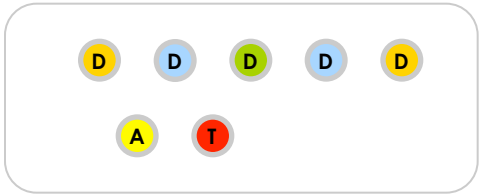
first distributed feature team

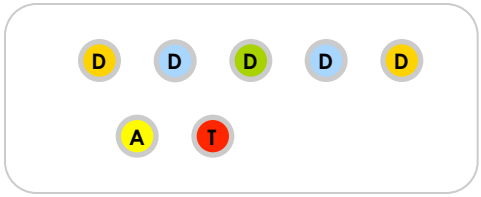


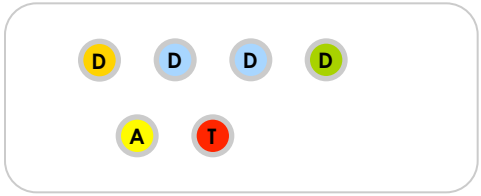










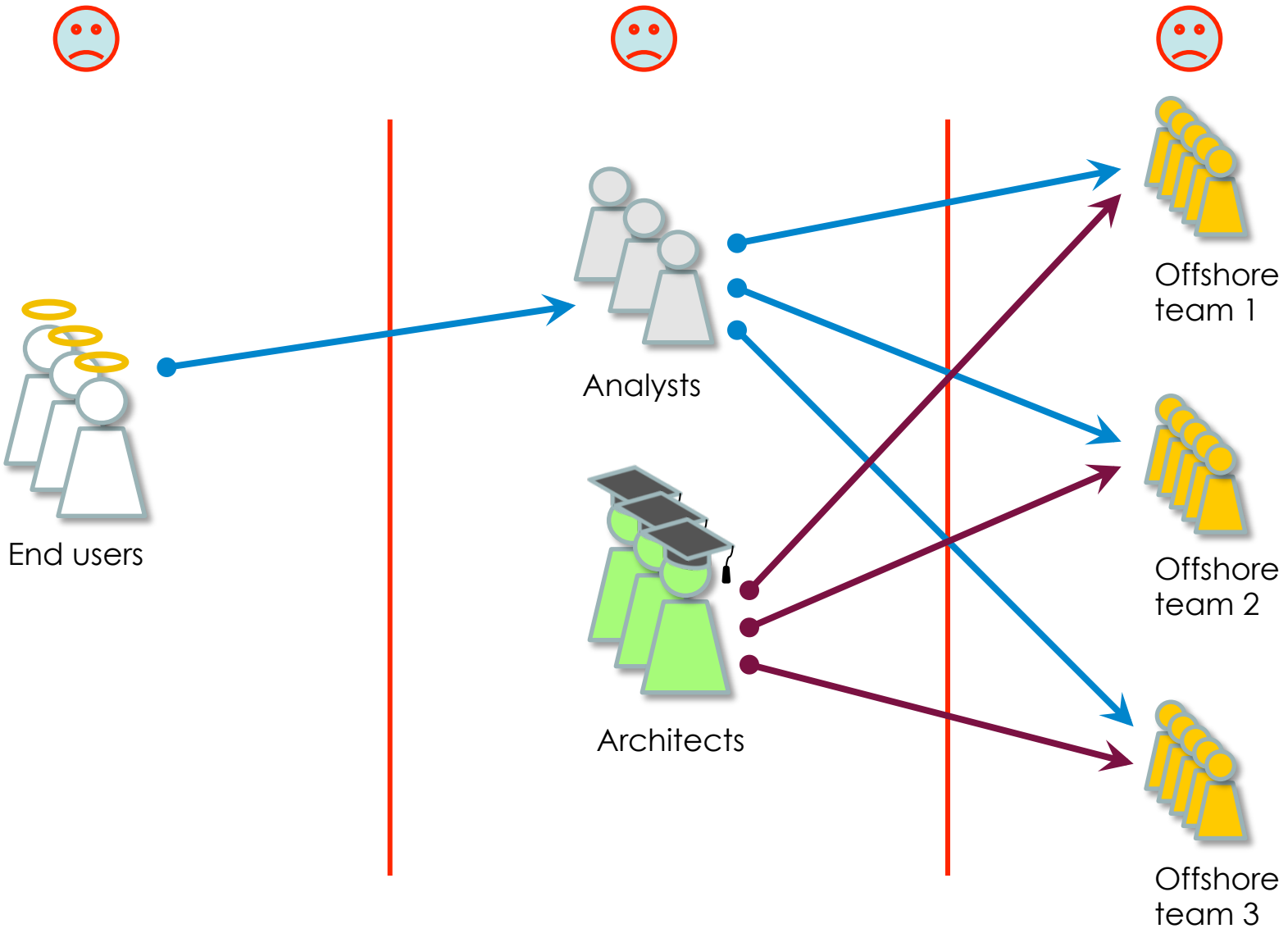


2006	2007	2008	2009	2010	2011
small co-located team	component teams	co-located feature teams	dispersed feature teams	first distributed feature team	many distributed feature teams


New York
London
Kiev
Hyderabad
Hong Kong

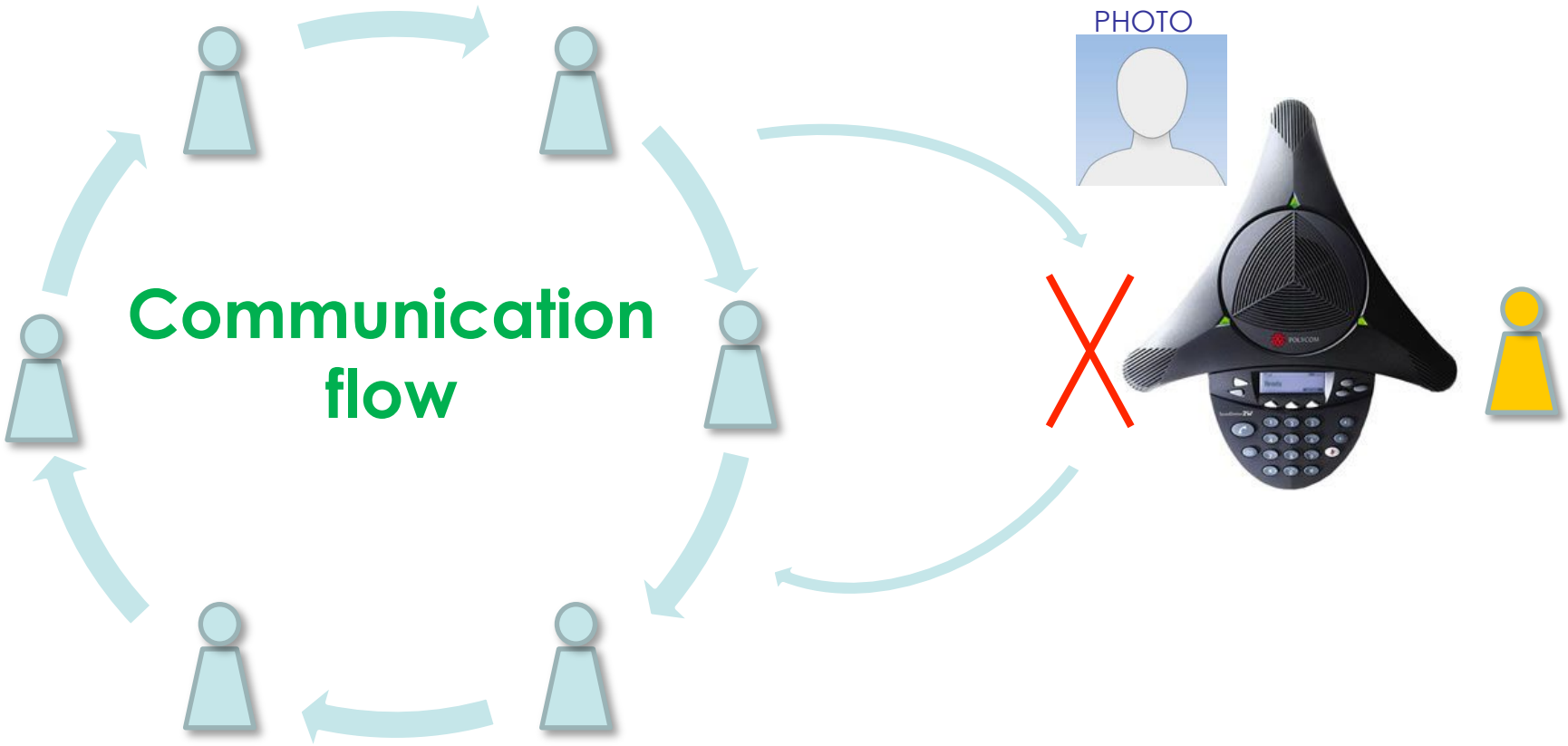
Evolving the communication

Communication issue

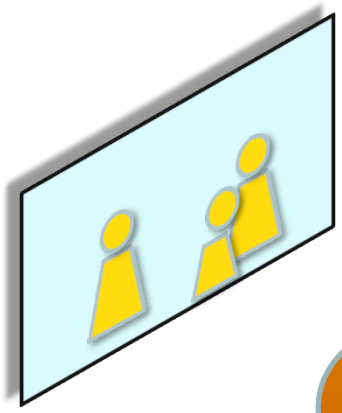


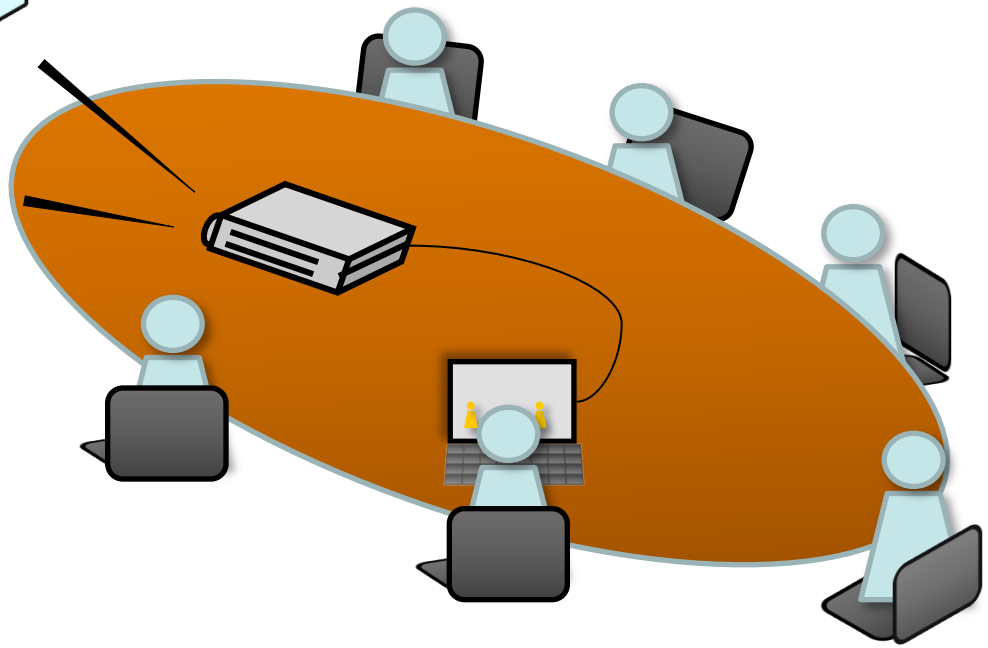
Broadening communication bandwidth

Polycom experiment

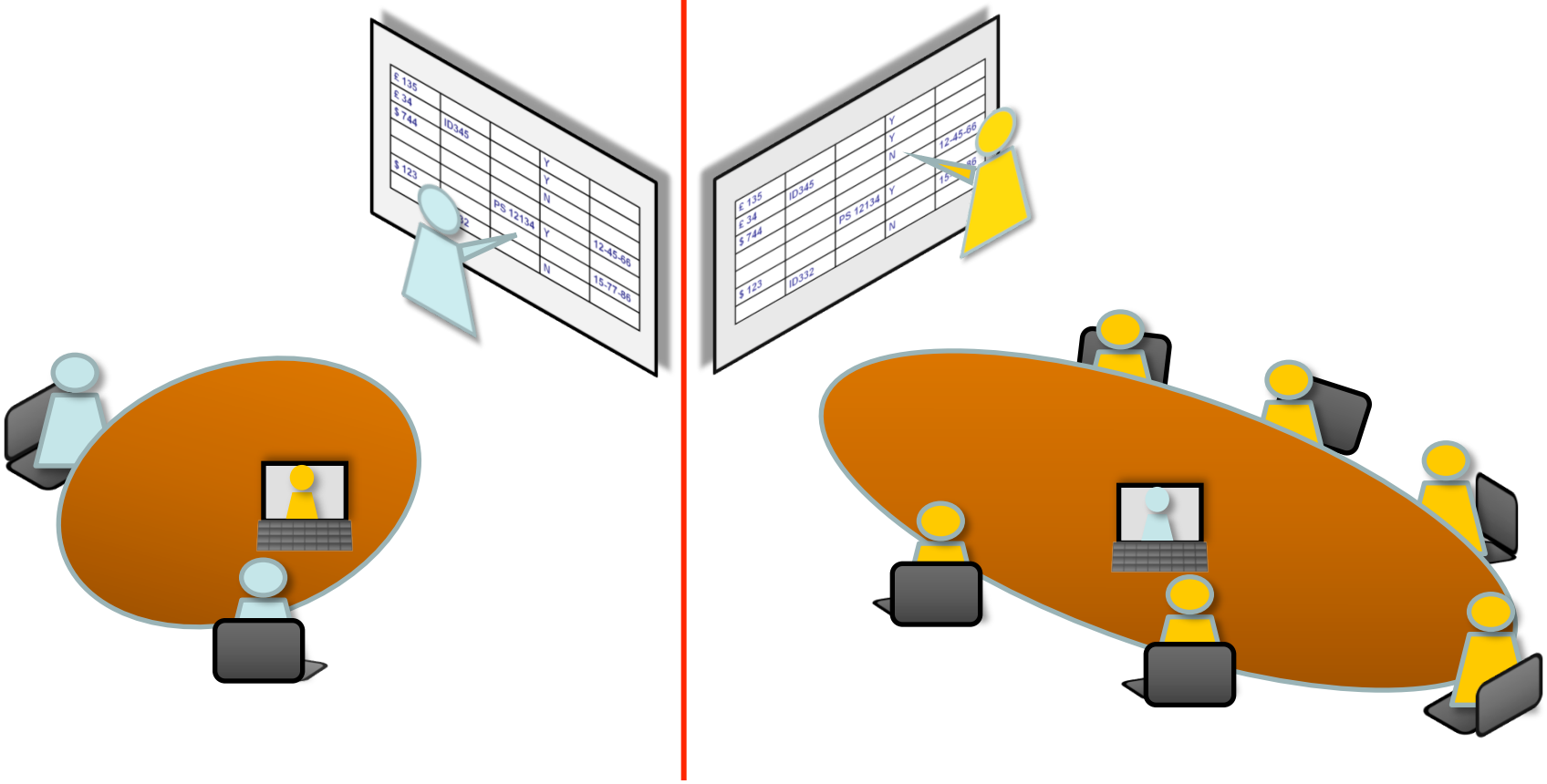


Skype and projector

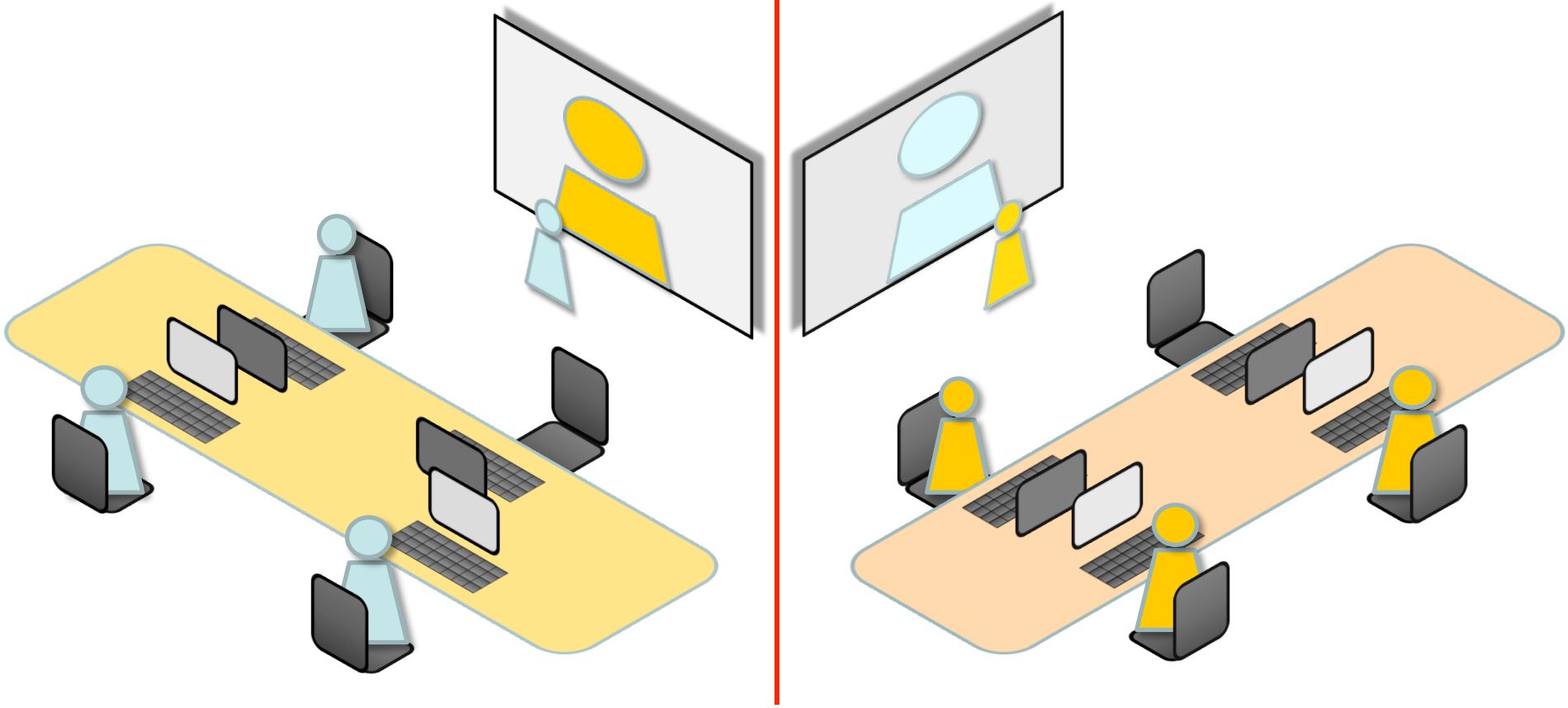




Interactive whiteboards + Skype

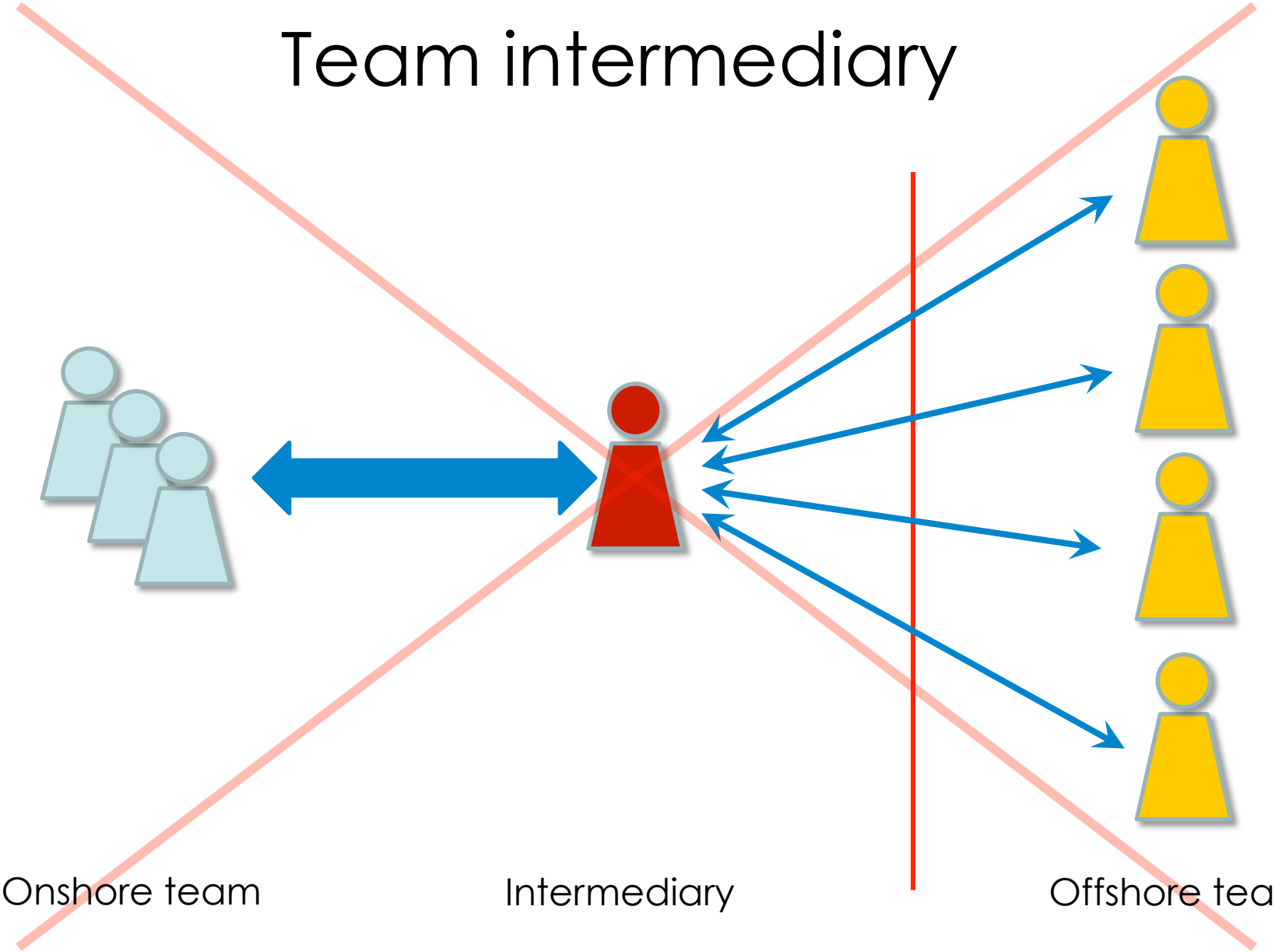


TelePresence?

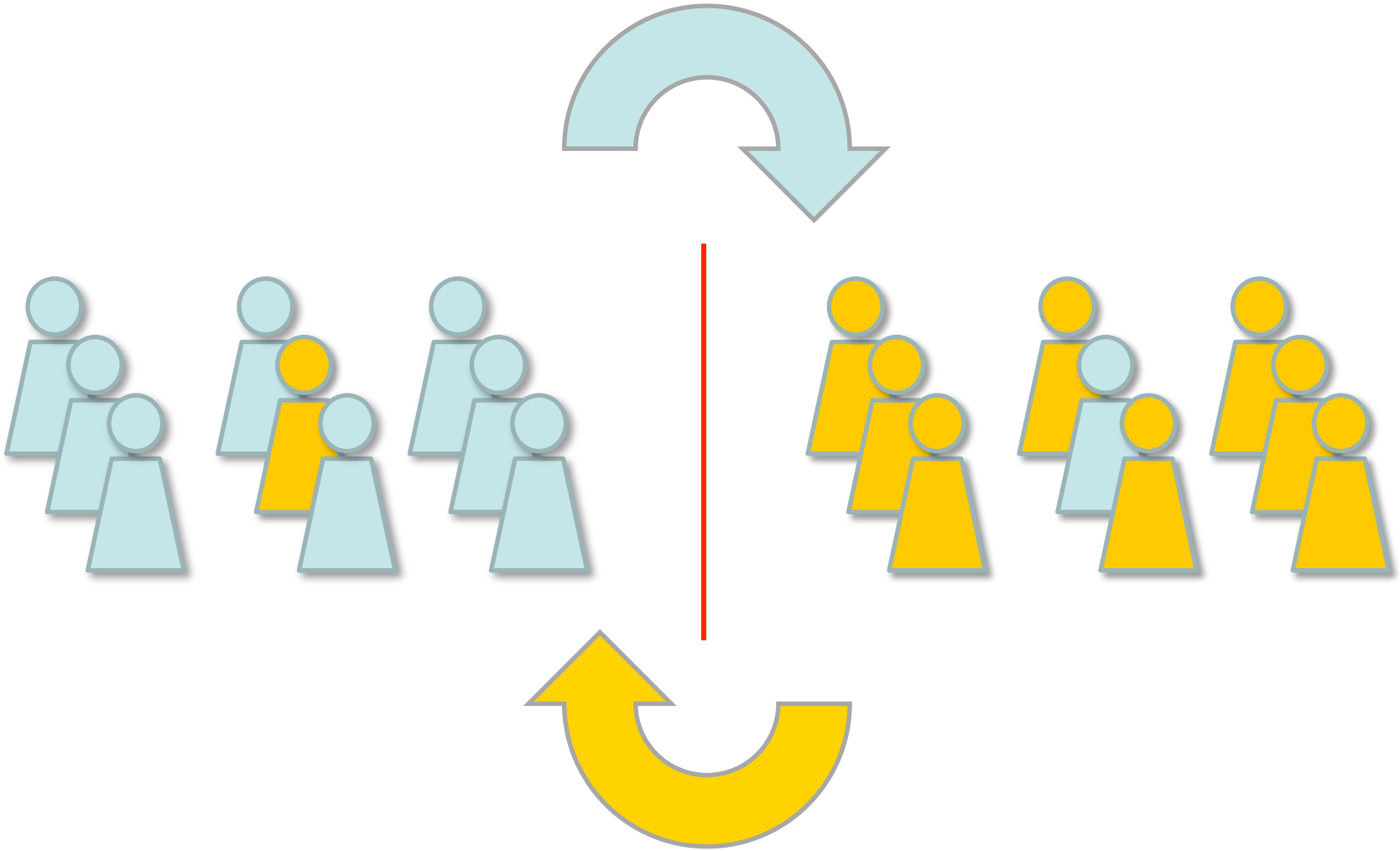


Bridging the communication gap

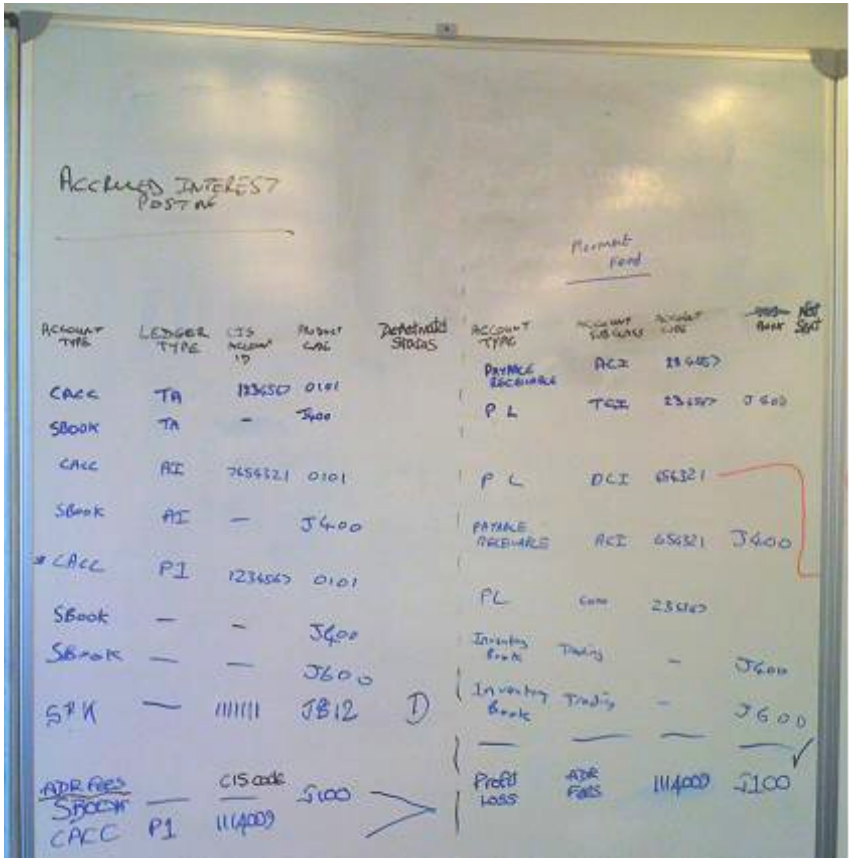
Team intermediary

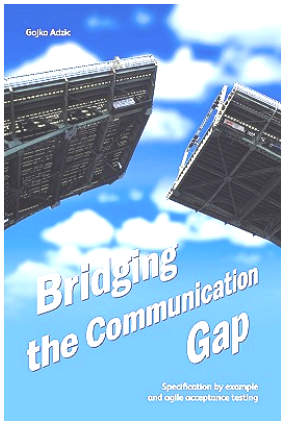


Bilateral rotation

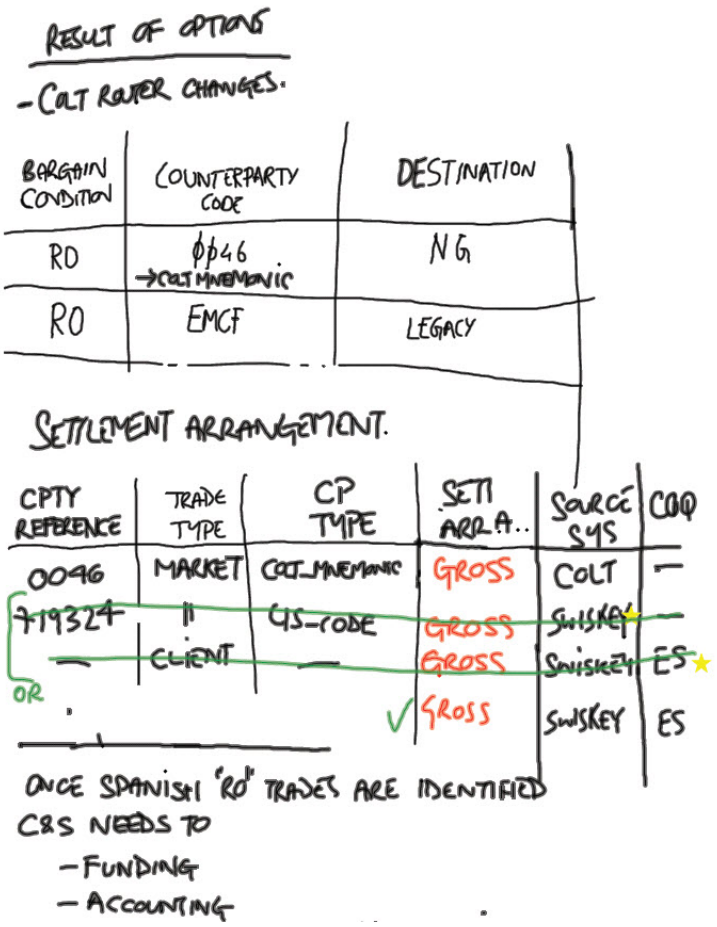


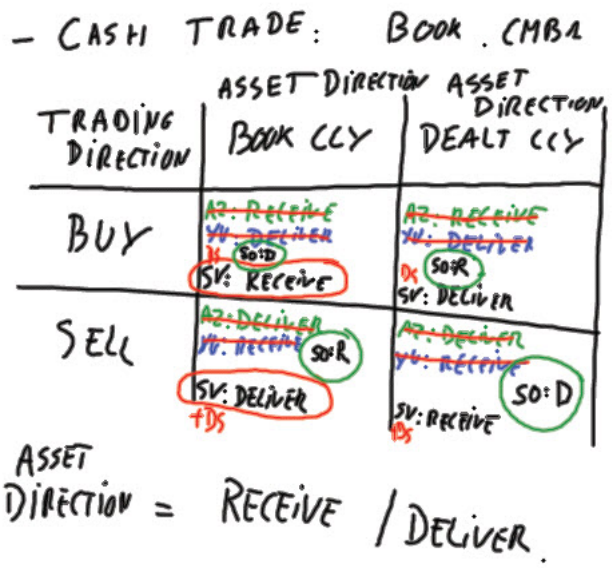
Specification by example



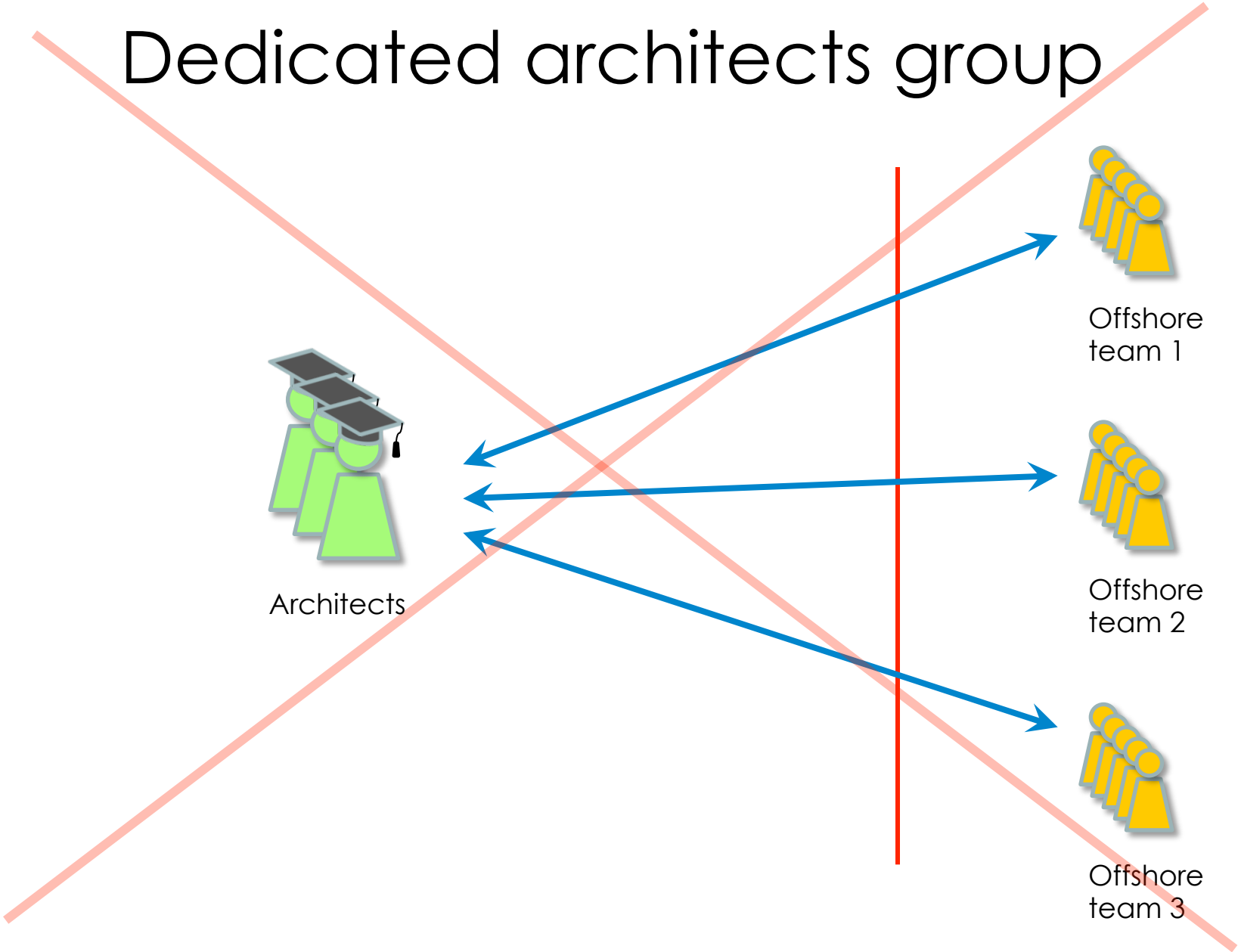


SBE over Smartboard



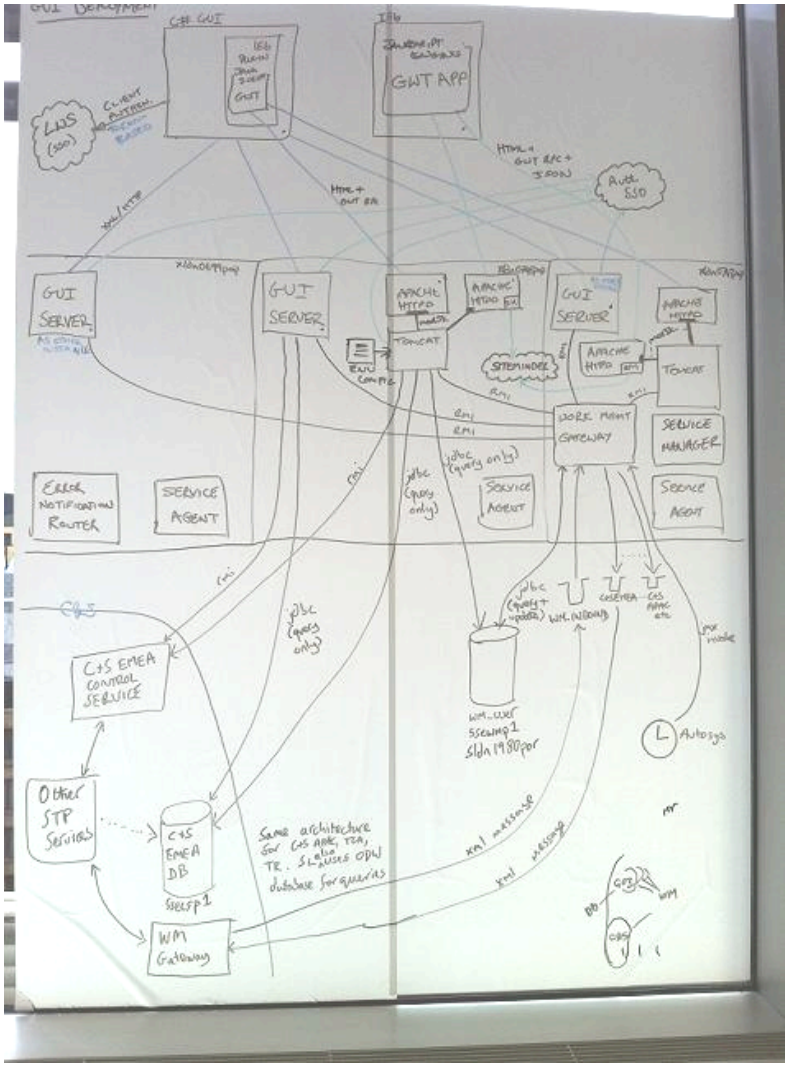


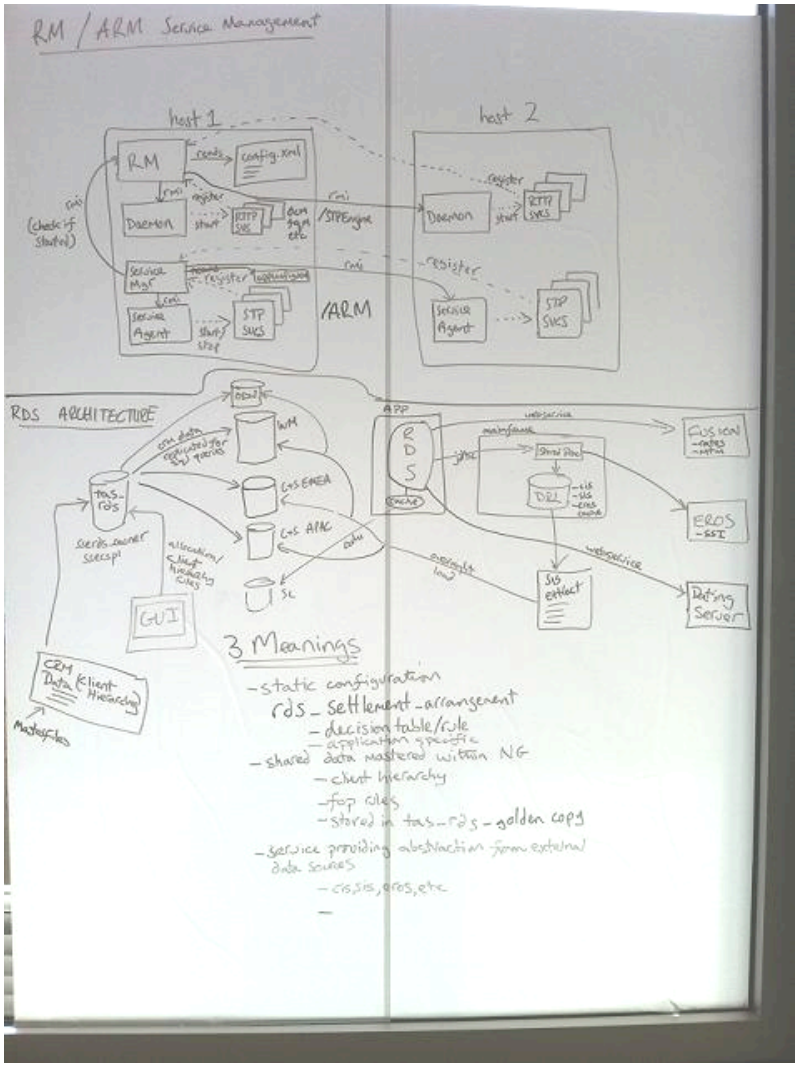
Dedicated architects group



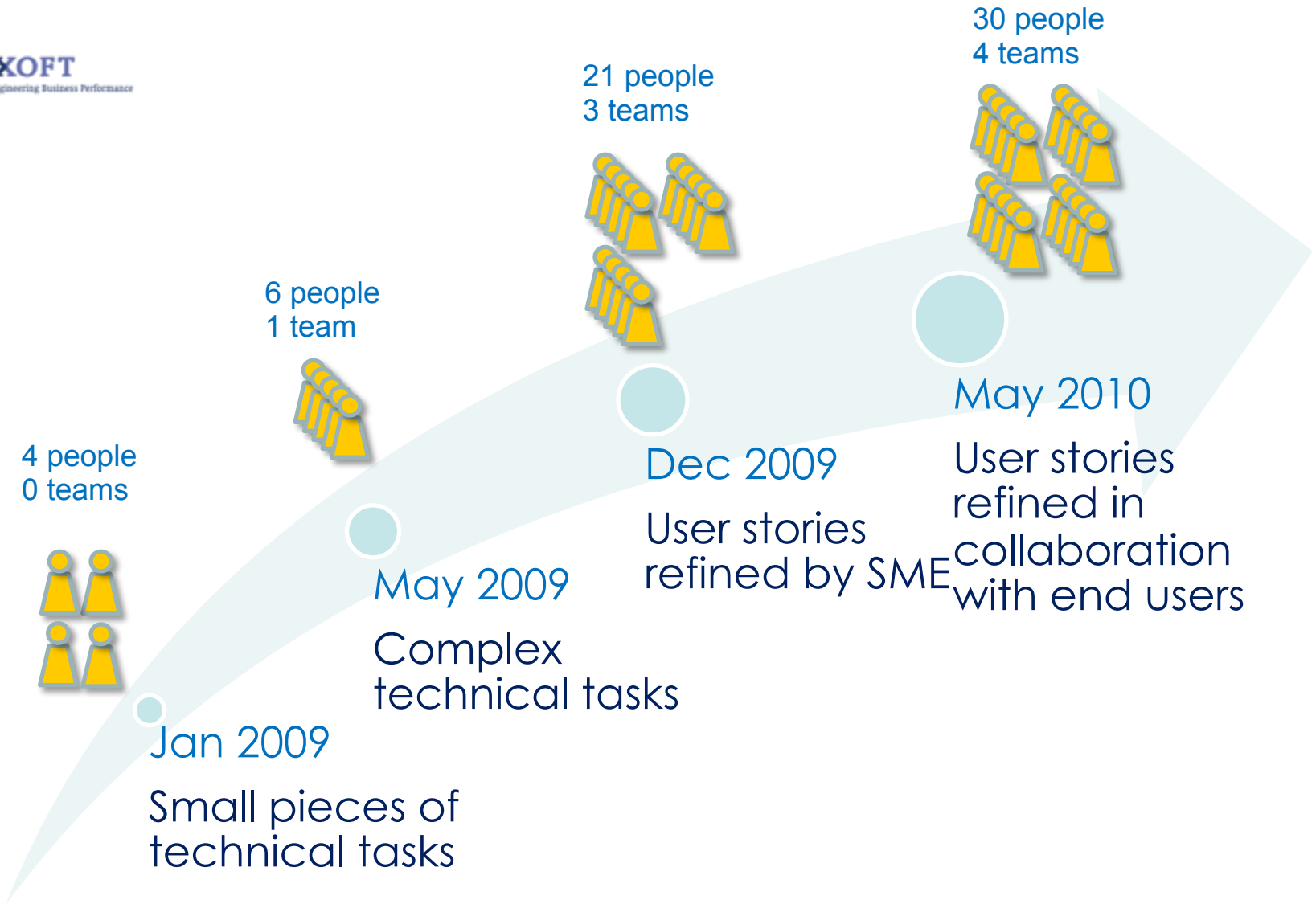
Joint architecture workshops

Joint architecture workshops



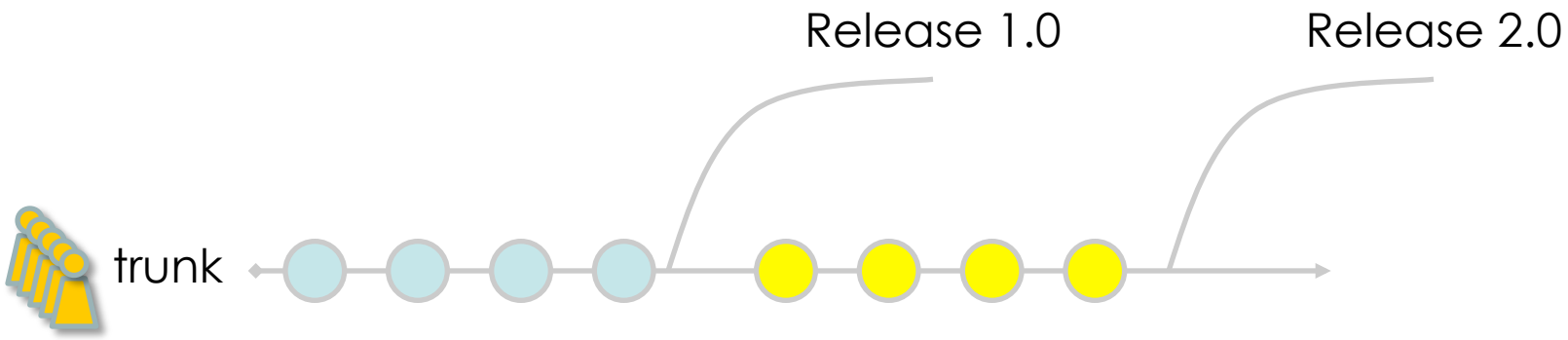


Luxoft teams evolution

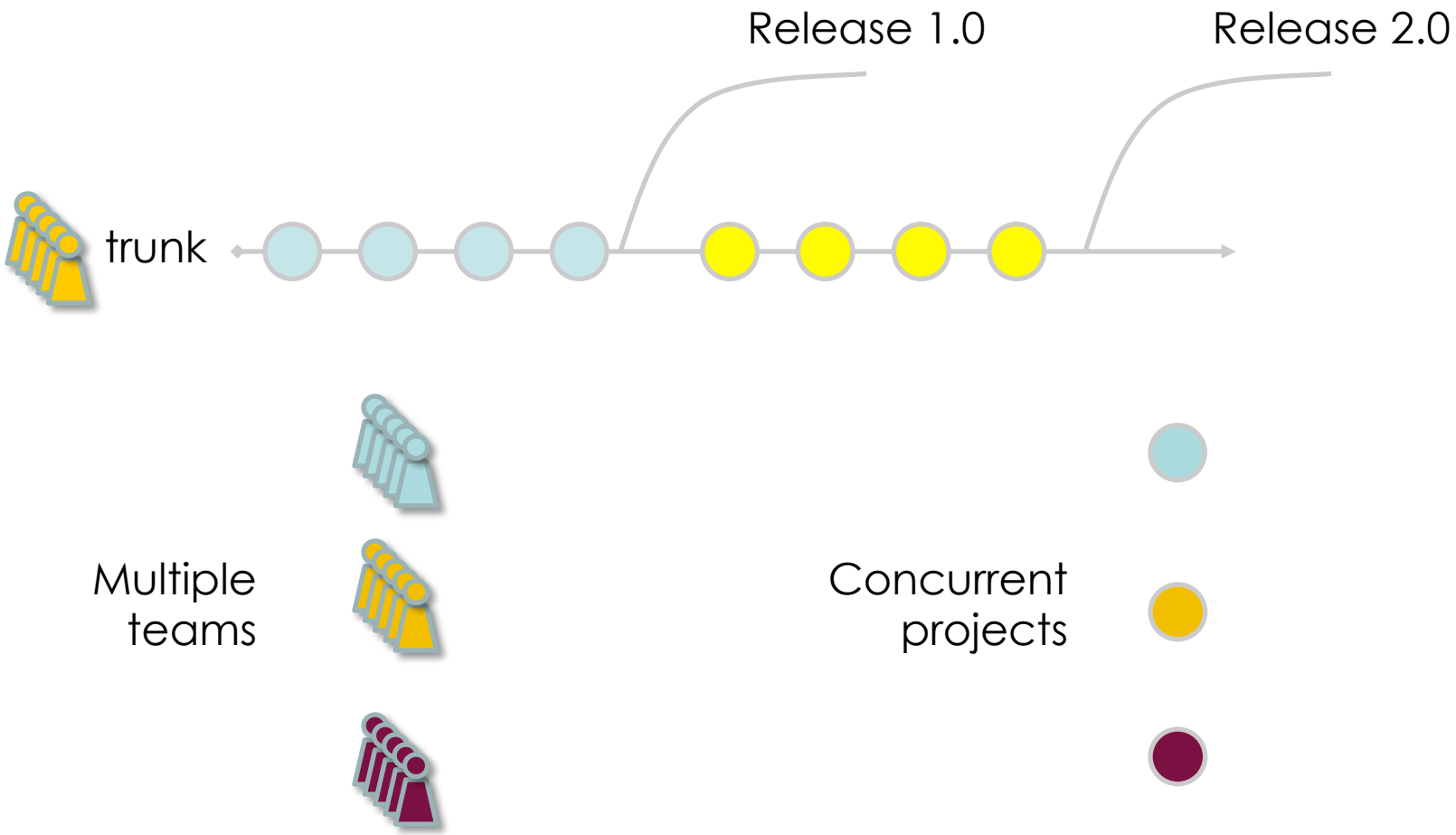


Tackling technical challenges

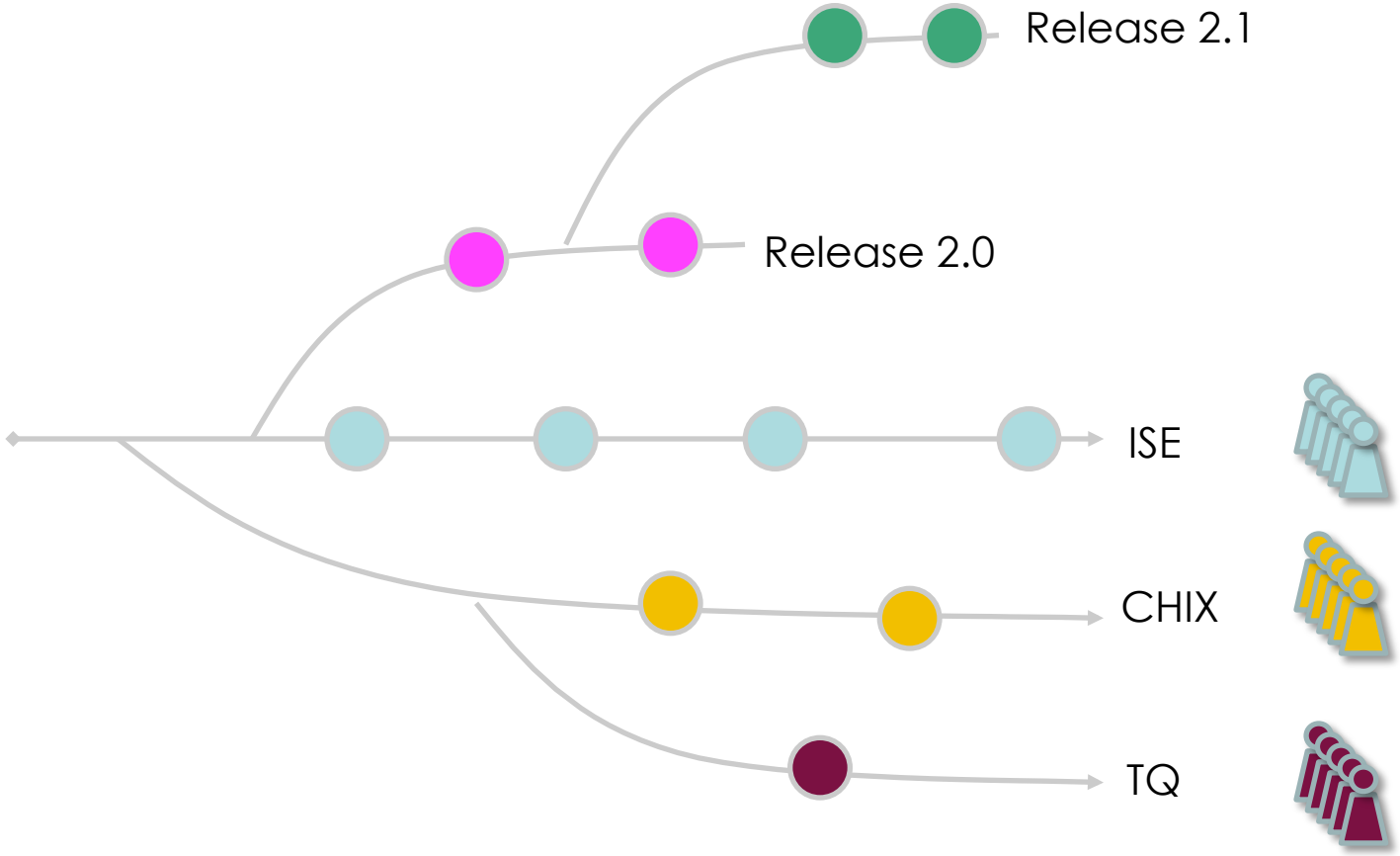
Release management

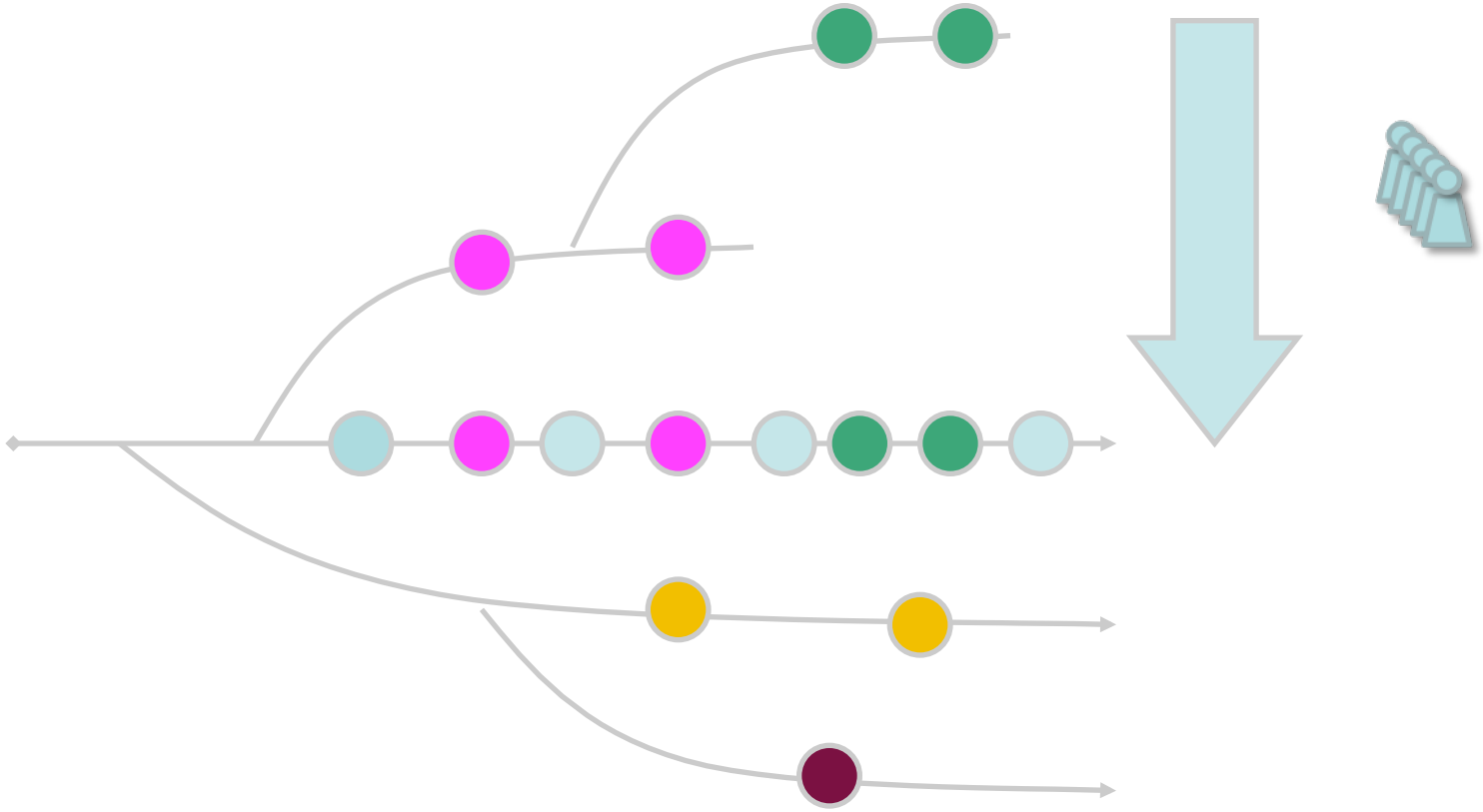


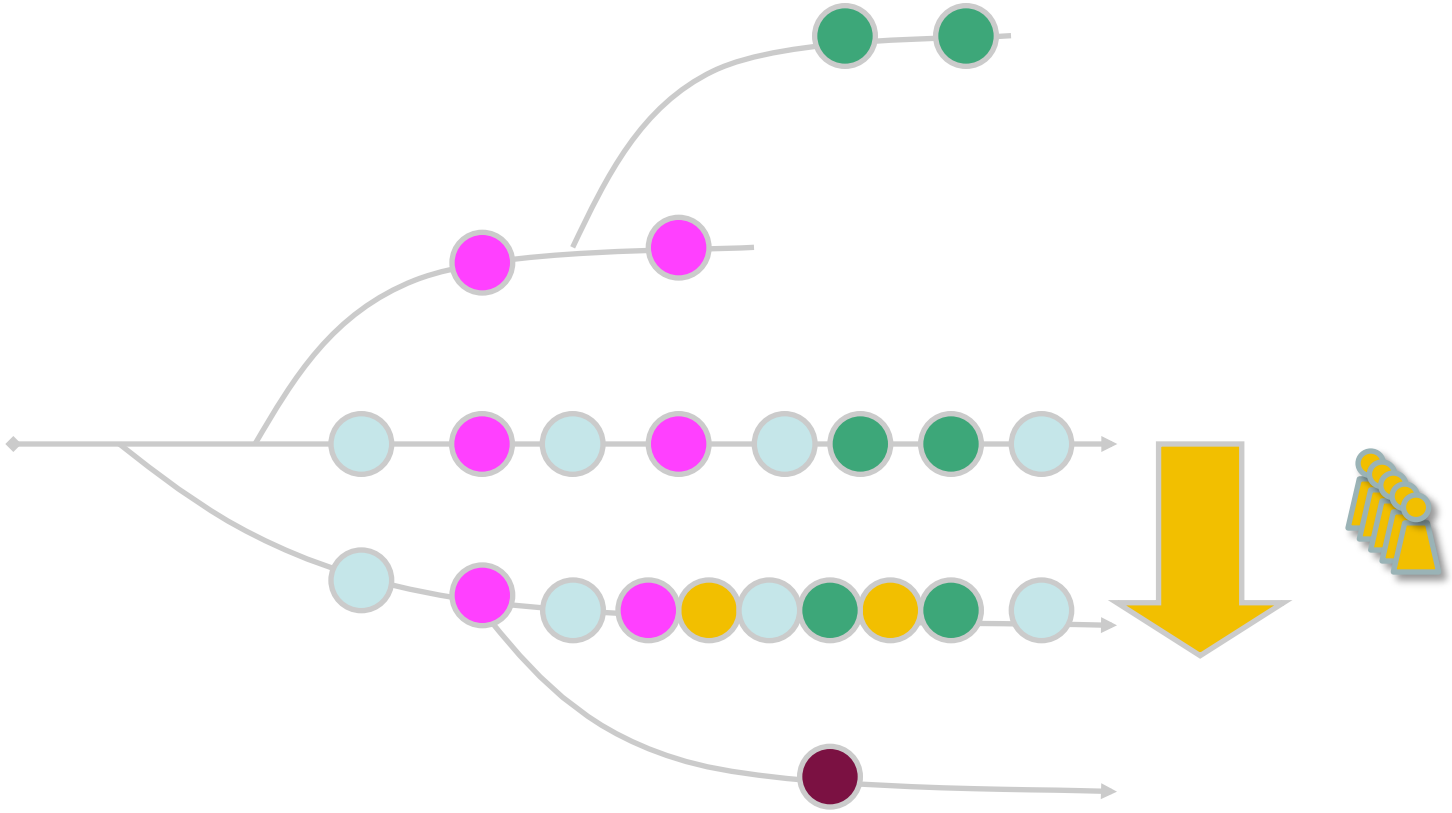
But how do you scale?

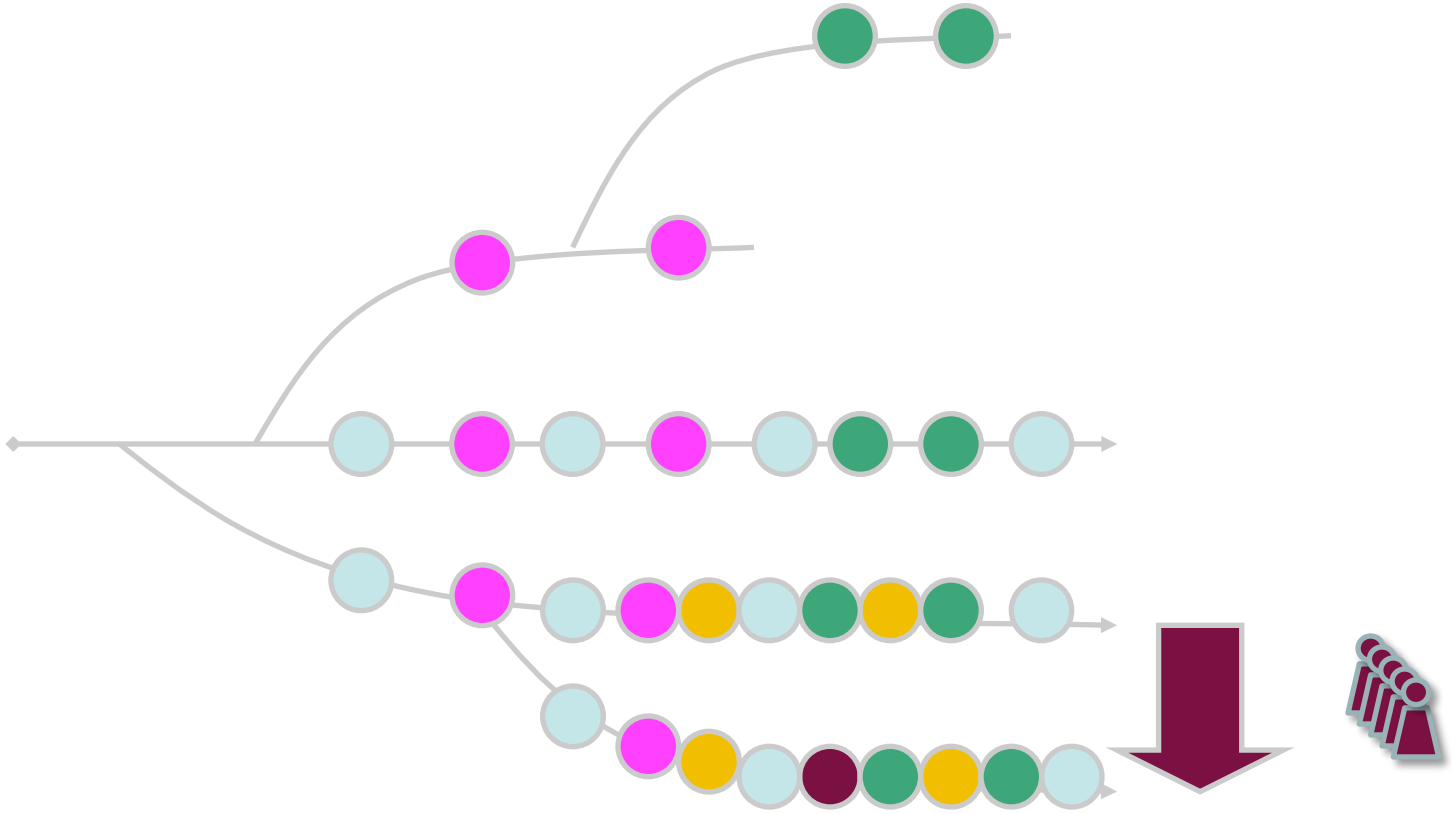


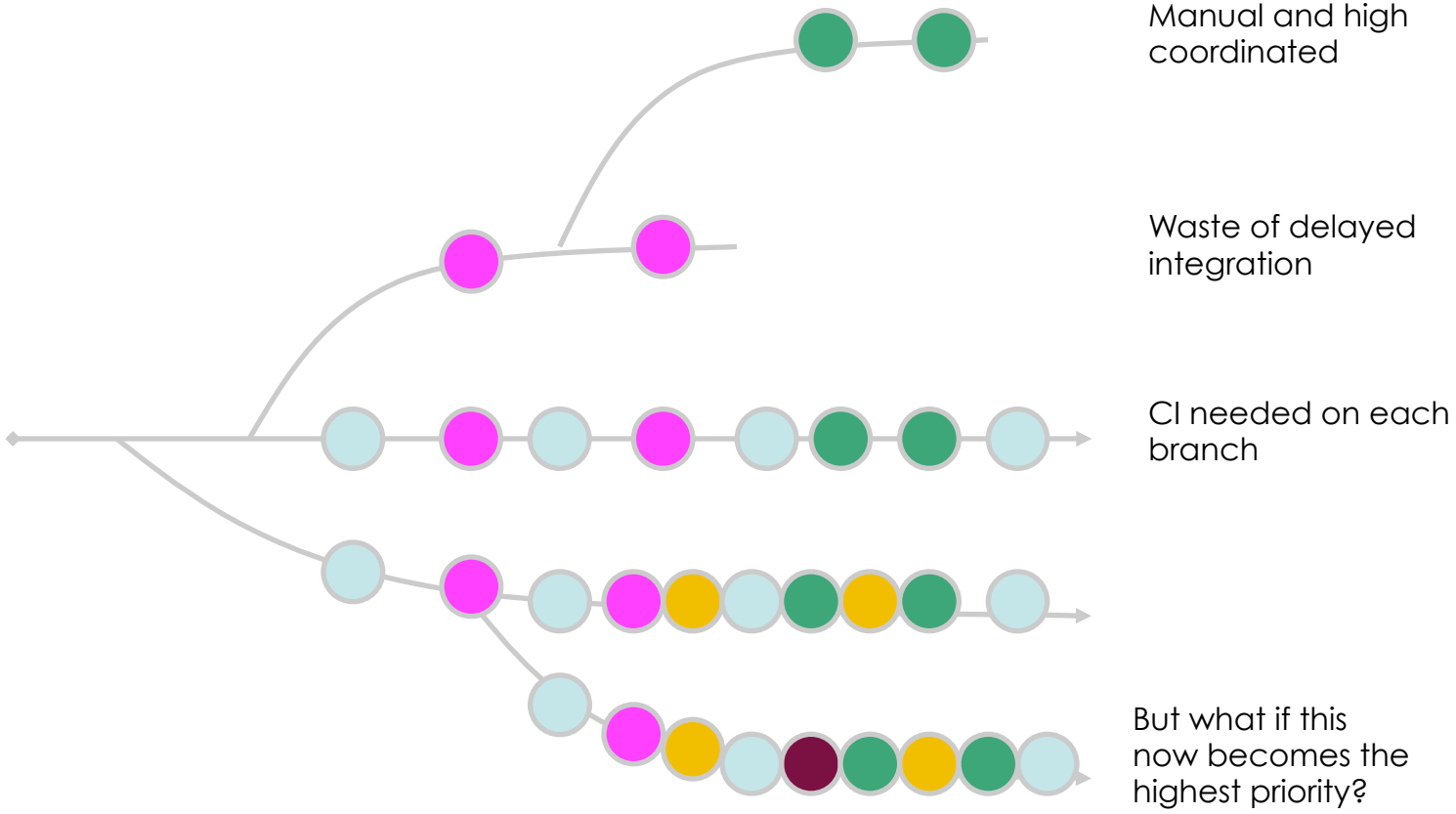
Let's introduce a process

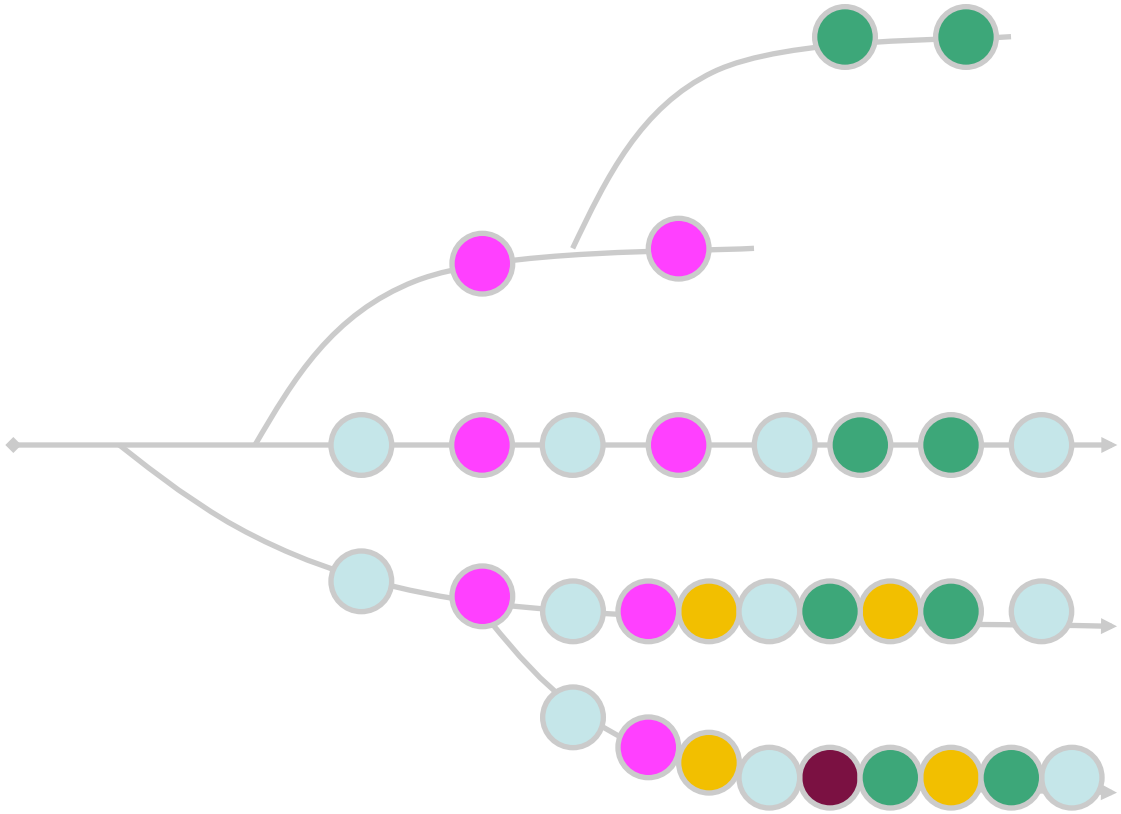









Manual and high coordinated
Waste of delayed integration
CI needed on each branch
But what if this now becomes the highest priority?

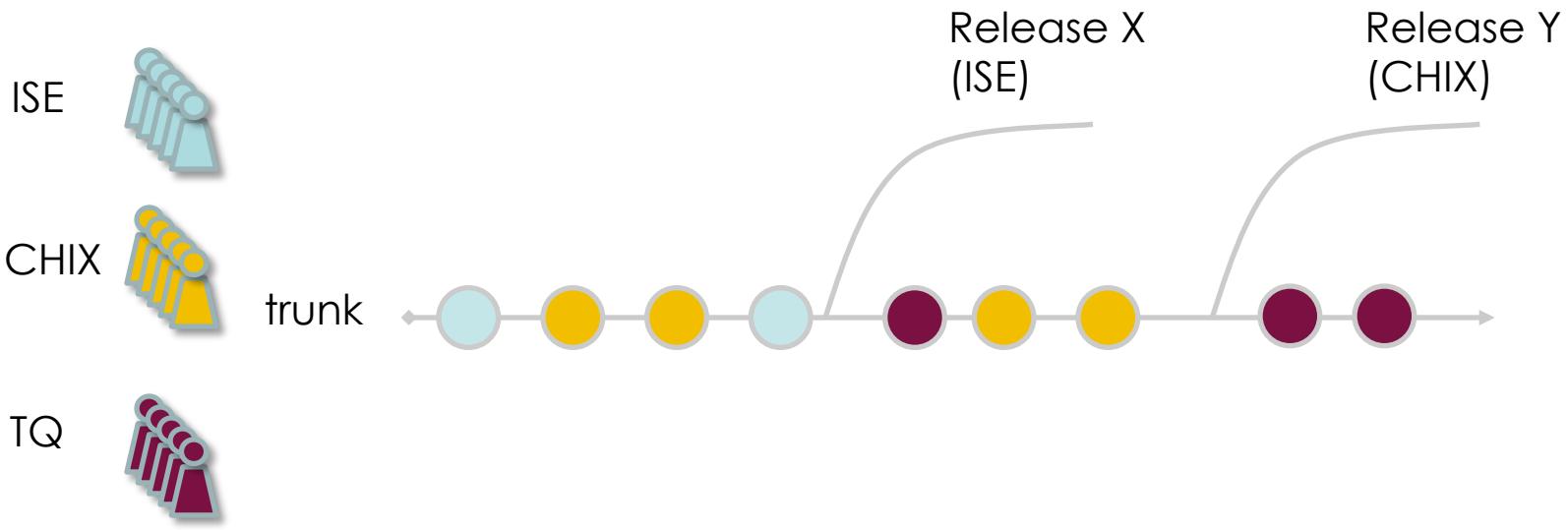


How do you address changing feature priority?

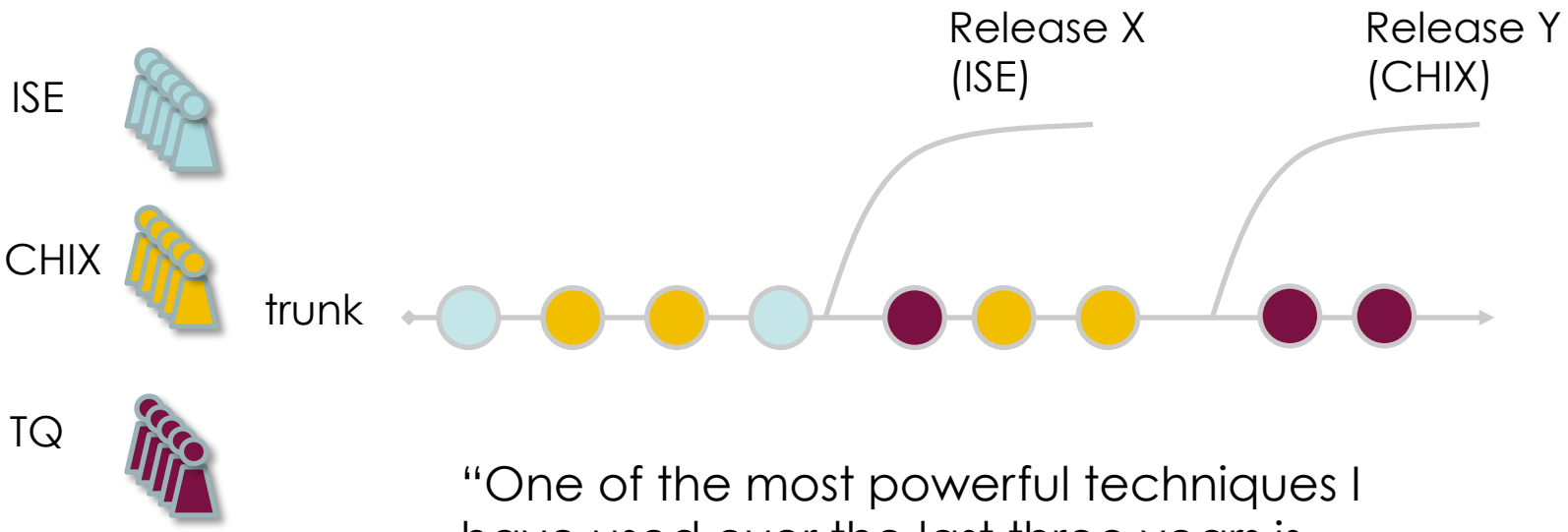
How do you allow incremental feature development?

How can you have an agile release function?

This is what we want

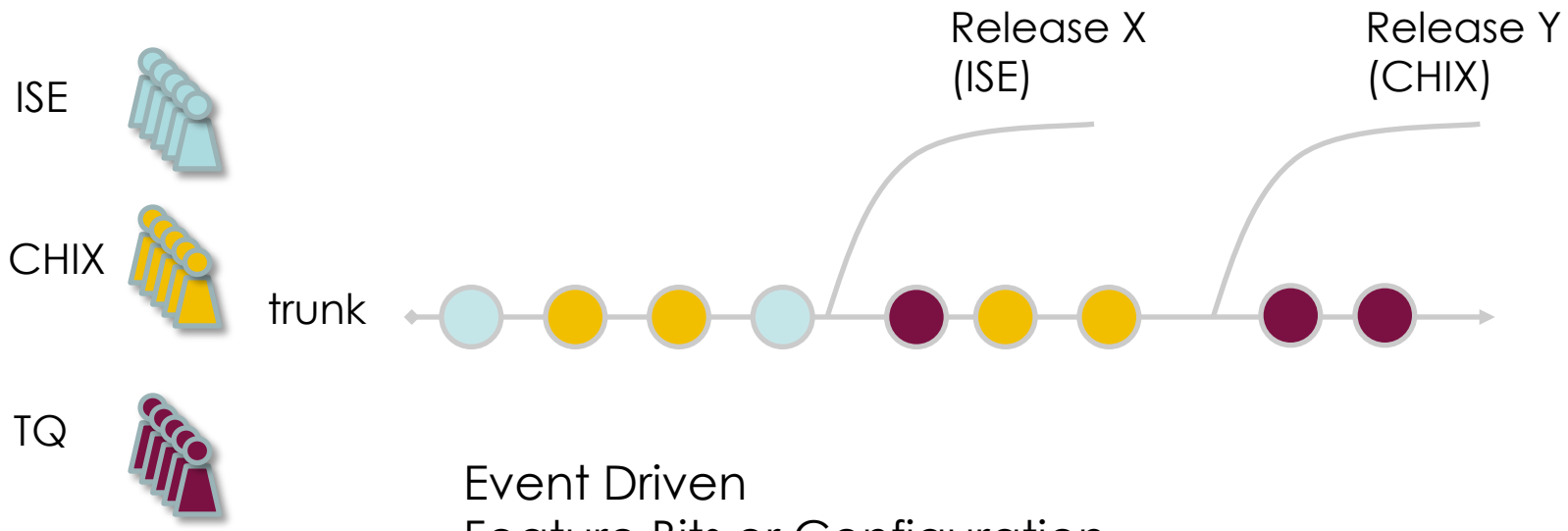


Latent Code Patterns



"One of the most powerful techniques I have used over the last three years is latent code patterns."

Latent Code Patterns



Event Driven
Feature Bits or Configuration
Modularity or Dependency Injection

Keeping it green

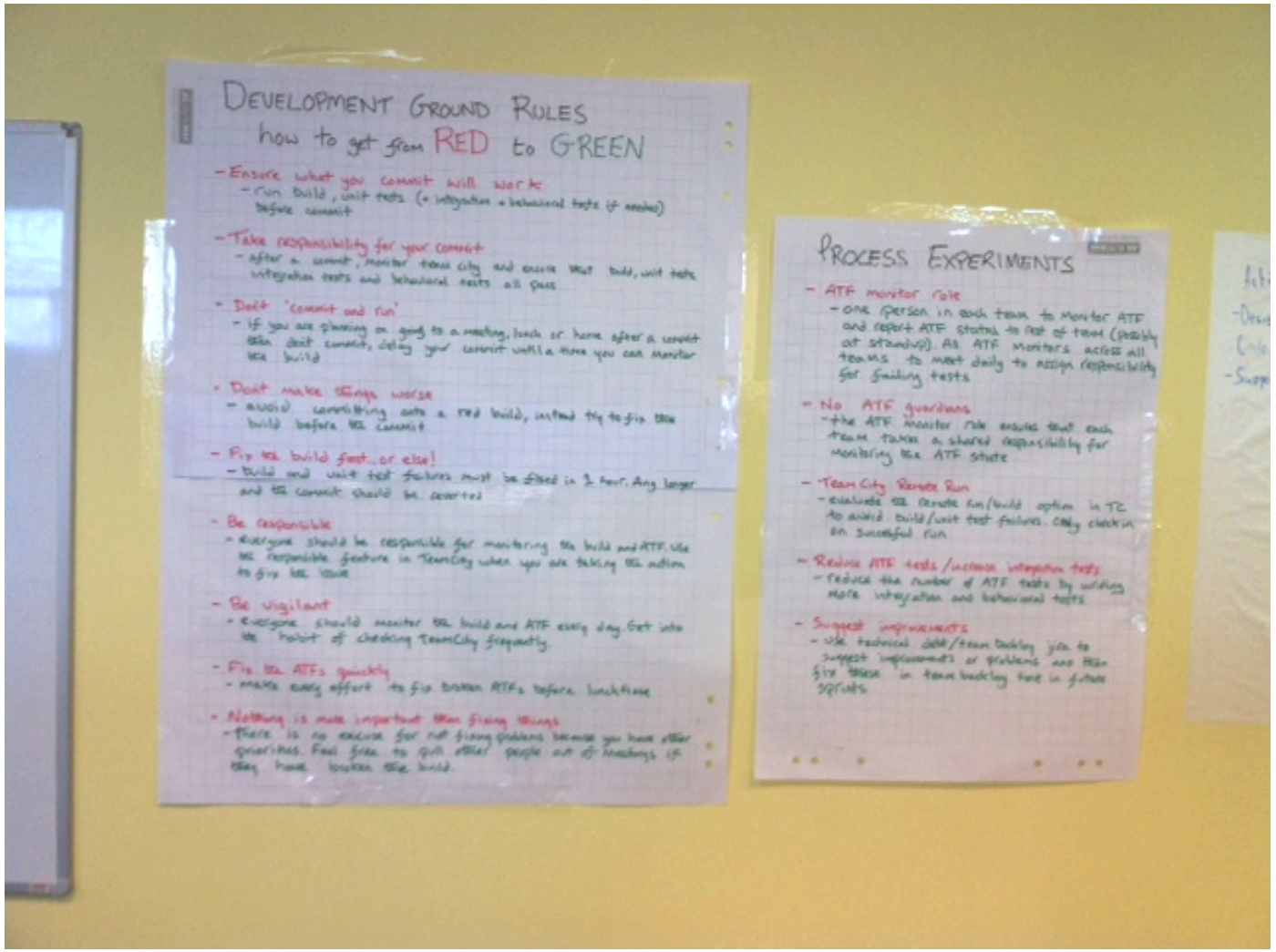
200 commits per day
1000 artefacts updated per day
1 commit every 5 minutes peak

A single bad commit ...

A wasted day

13 hours elapsed time wasted
500 hours of effort wasted

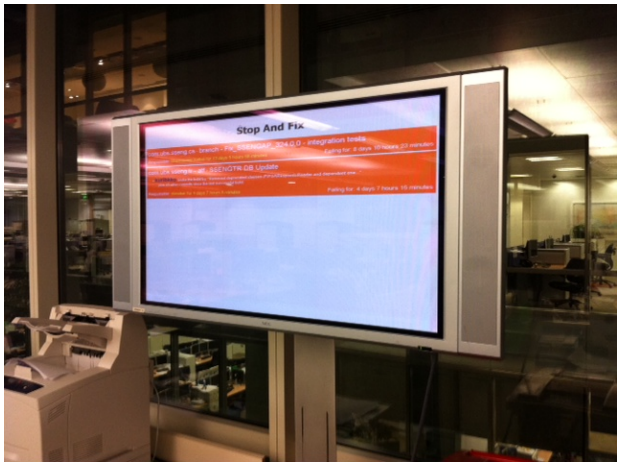
Coaching


DEVELOPMENT GROUND RULES
how to get from RED to GREEN
- Ensure what you commit will work
- run build, unit tests (+ integration + behavioral tests if needed) before commit
- Take responsibility for your commit
- after a commit, monitor team city and ensure that build, unit tests integration tests and behavioral tests all pass
- Don't 'commit and run'
- if you are planning on going to a meeting, lunch or home after a commit then don't commit, delay your commit until a time you can monitor the build
- Don't make things worse
- avoid committing onto a red build, instead try to fix the build before the commit
- Fix the build fast...or else!
- build and unit test failures must be fixed in 1 hour. Any longer and the commit should be reverted
- Be responsible
- everyone should be responsible for monitoring the build and ATF. Use the responsible feature in TeamCity when you are taking the action to fix the issue
- Be vigilant
- everyone should monitor the build and ATF every day. Get into the habit of checking TeamCity frequently
- Fix the ATFs quickly
- make every effort to fix broken ATFs before lunchtime
- Nothing is more important than fixing things
- there is no excuse for not fixing problems because you have other priorities. Feel free to pull other people out of meetings if they have broken the build.
PROCESS EXPERIMENTS
- ATF monitor role
- one person in each team to monitor ATF and report ATF status to rest of team (possibly at standup). All ATF monitors across all teams to meet daily to assign responsibility for failing tests
- No ATF guardians
- the ATF monitor role ensures that each team takes a shared responsibility for monitoring the ATF state
- TeamCity Remote Run
- evaluate the remote run/build option in TC to avoid build/unit test failures. Only check in on successful run
- Reduce ATF tests / increase integration tests
- reduce the number of ATF tests by writing more integration and behavioral tests
- Suggest improvements
- use technical debt/team backlog jira to suggest improvements or problems and then fix these in team backlog time in future sprints

Fast visible feedback

- 24 Build Targets

- 37 Test Targets



So how did we get here?

- Protect the team and empower them
- Go and see
- Embrace “muddling through”
- Don’t accept the status quo – have courage
- Follow through with change – be tenacious
- Respect and trust people
- Invest in the engineering
- Stop worrying about big – make it small