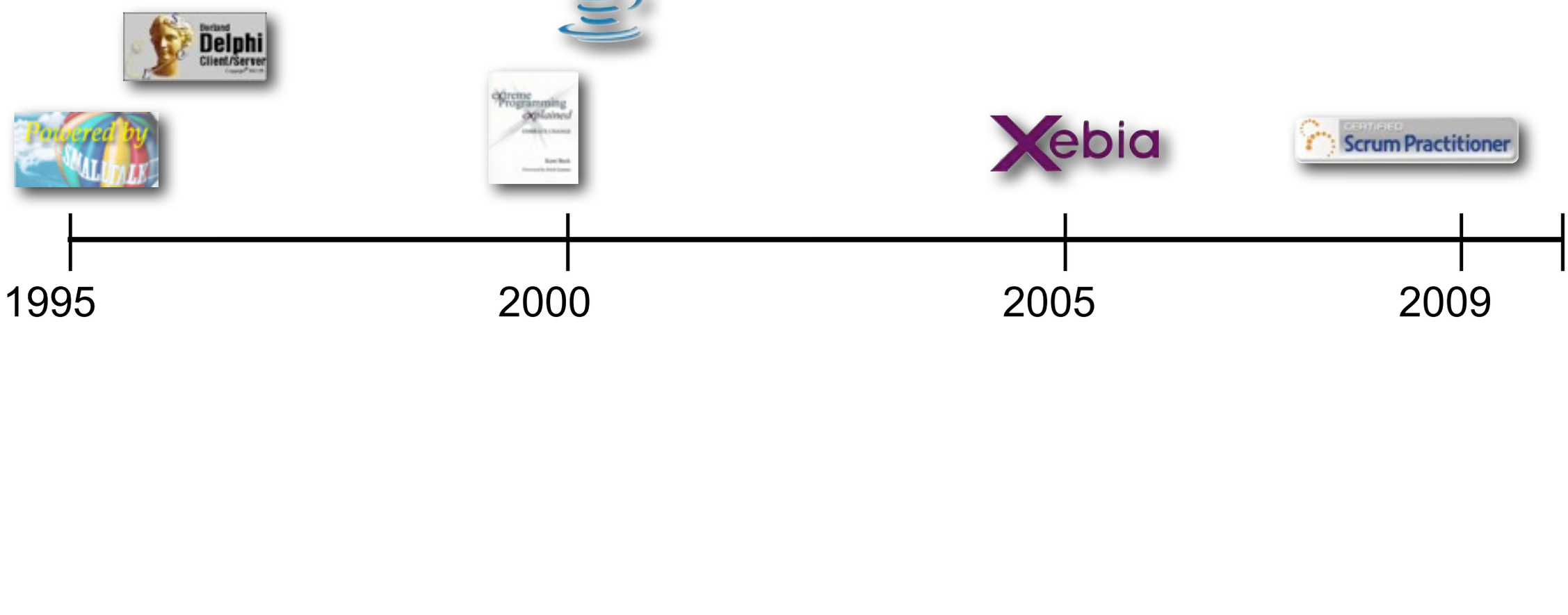
Practical tools for the Product Owner: Focus, Value, Flow

Serge Beaumont

About me


Delphi
Client/Server
Powered by
SMALLTALK
extreme Programming explained
Xebia
CERTIFIED
Scrum Practitioner
1995
2000
2005
2009

You are doing agile with Scrum

Scrum Master

Team

Product Owner

You are the Product Owner

- You are the *navigator*
- Represent **all** stakeholders
- Maintain the Product Backlog

Backlog
A
B
C
D
E
F
G
H
I
J
…

Statue: St. Brendan the Navigator. Is claimed to have found America even before the Vikings.

You have become the bottleneck

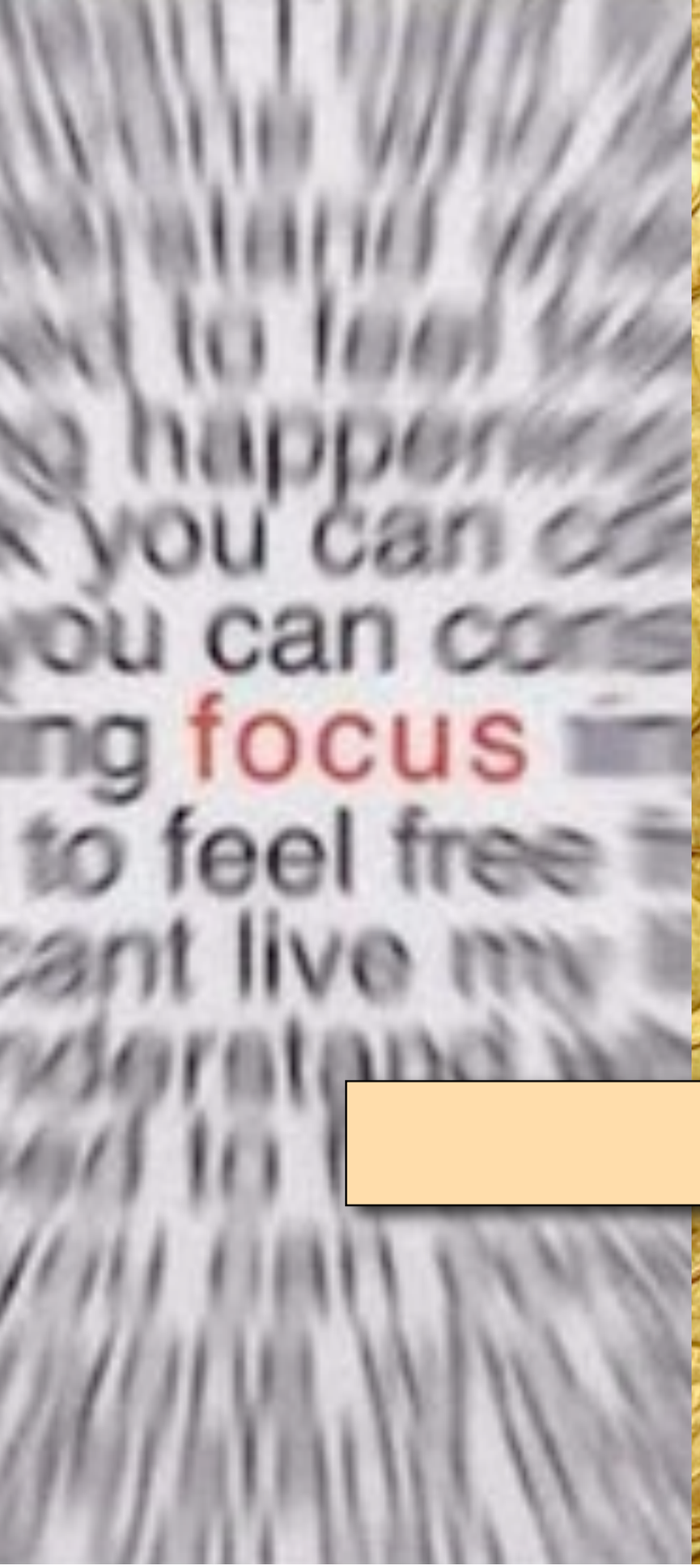
Backlog
I
J

Implementing Agile is like Pac-Man eating a power dot: the tables are turned, and they consume the backlog faster than a Product Owner can fill it up. I've seen on all my assignments: it is almost a law of nature for this to happen...

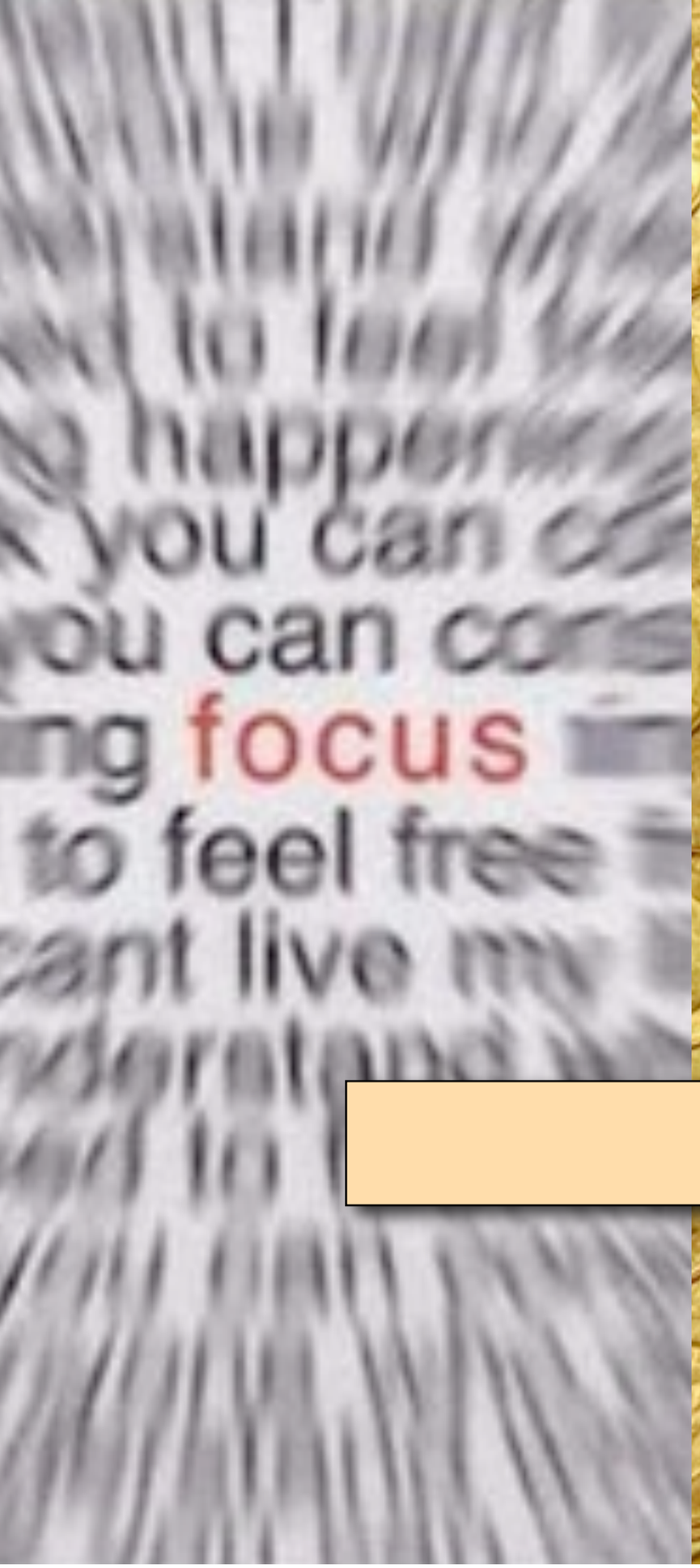
The reason of becoming a bottleneck is that there is a big difference between "I've got a thousand ideas!" and actually creating a clear vision with real value that is agreed upon by all stakeholders.

You want to deliver ROI

- The *right* value
- As *soon* as possible
- At the right *cost*
- While *balancing* stakeholders
- *In service of* the larger organization



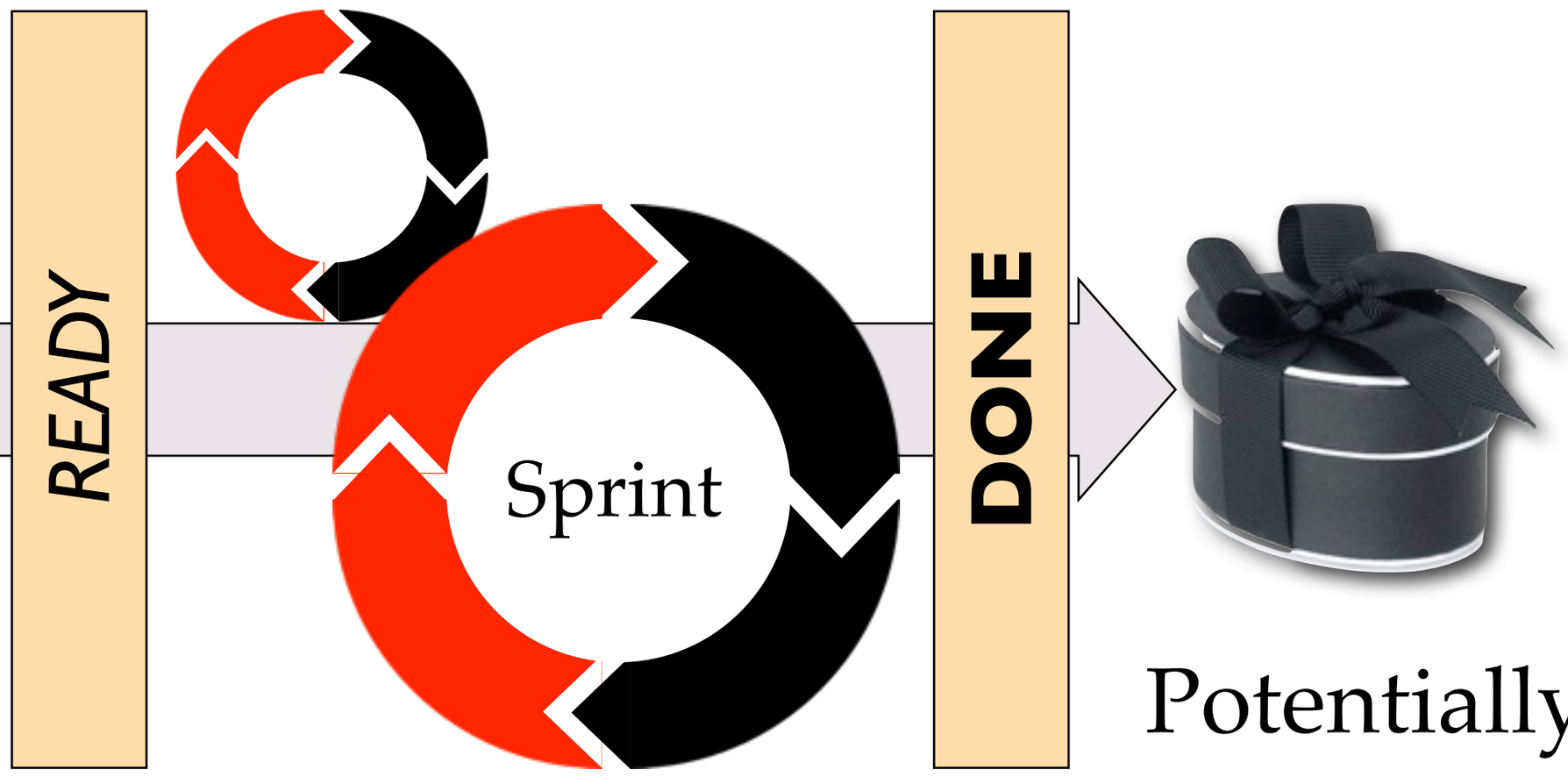
READY



READY

Scrum has *two* stable States

Backlog



Potentially Shippable Product

READY

READY

The Definition of Done ensures real, working product

Negotiate

Definition of Done

- ✓Acceptance Tested
- ✓Unit Tested
- ✓No increased technical debt
- ✓Documentation in order
- ✓Conform standard X

DONE

- **Product Owner** decides if OK
- Describes the **product** at end of Sprint
- Quality **checklist**
- **Static** constraints and requirements
- Can **vary** over time

READY

The Definition of Ready prevents Sprint thrashing

Negotiate

Definition of Ready

- ✓ Why? Business value
- ✓ What? Outcome vision
- ✓ How? Implementation strategy & cost
- ✓ Enough for 1,5-2 Sprints?
- ✓ Granularity OK?

READY

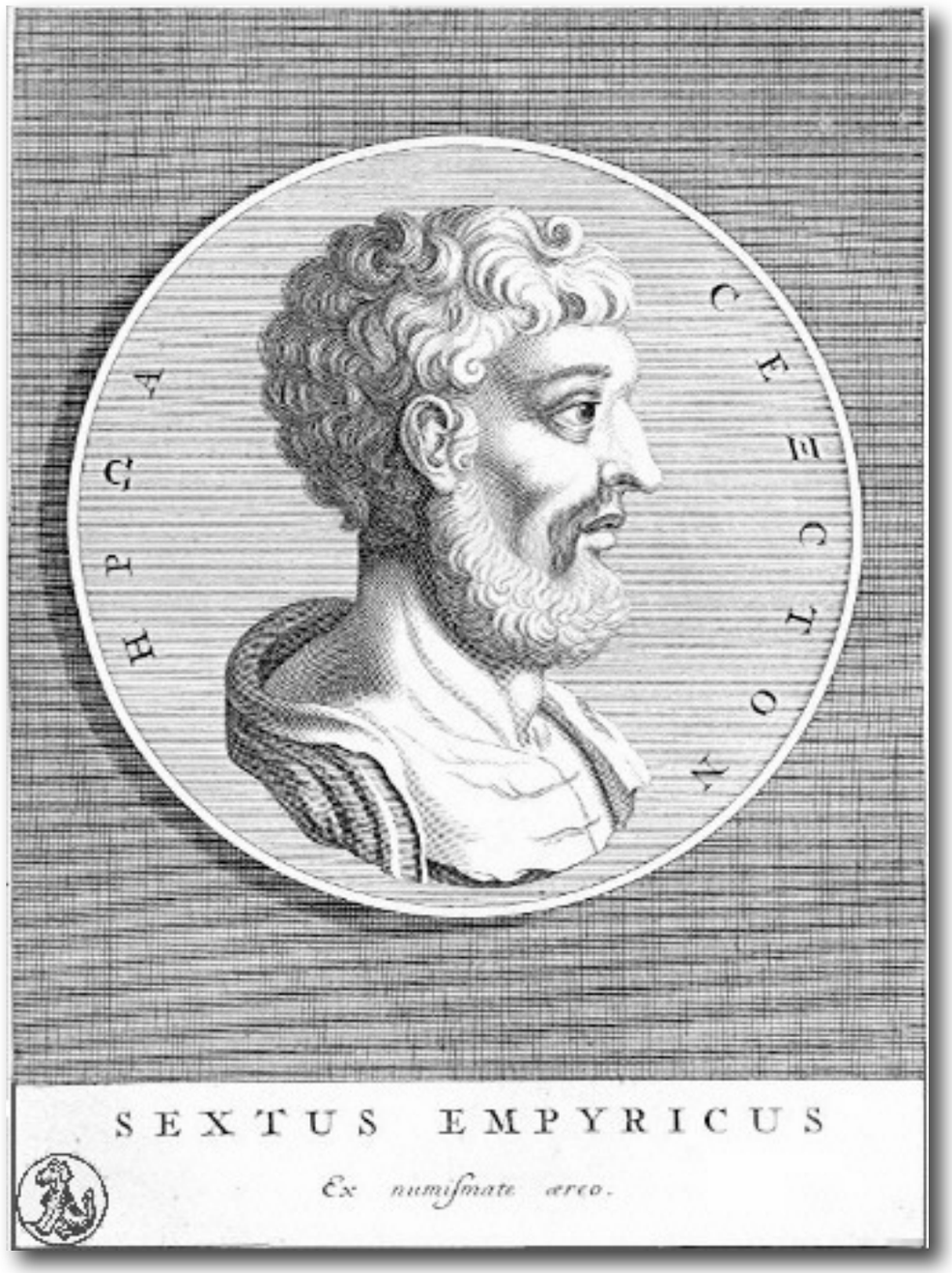
- **Team** decides if OK
- Describes the **Backlog** before Sprint
- Quality **checklist**

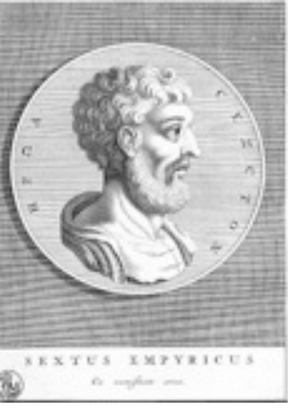
The Definition of Ready is - at least while I'm creating this course - not something that is known or used in the Scrum community at all. I needed to define some criteria for the quality of the backlog at a customer, and combined with what Jeff told me about the dynamic model of Scrum, the DoR clicked into place for me. I found it so useful that I think it should be part of standard Scrum.

Value

FLOW

READY


SEXTUS EMPYRICUS
Ex numiſmate æreo.



Empiricism embraces *a posteriori* knowledge

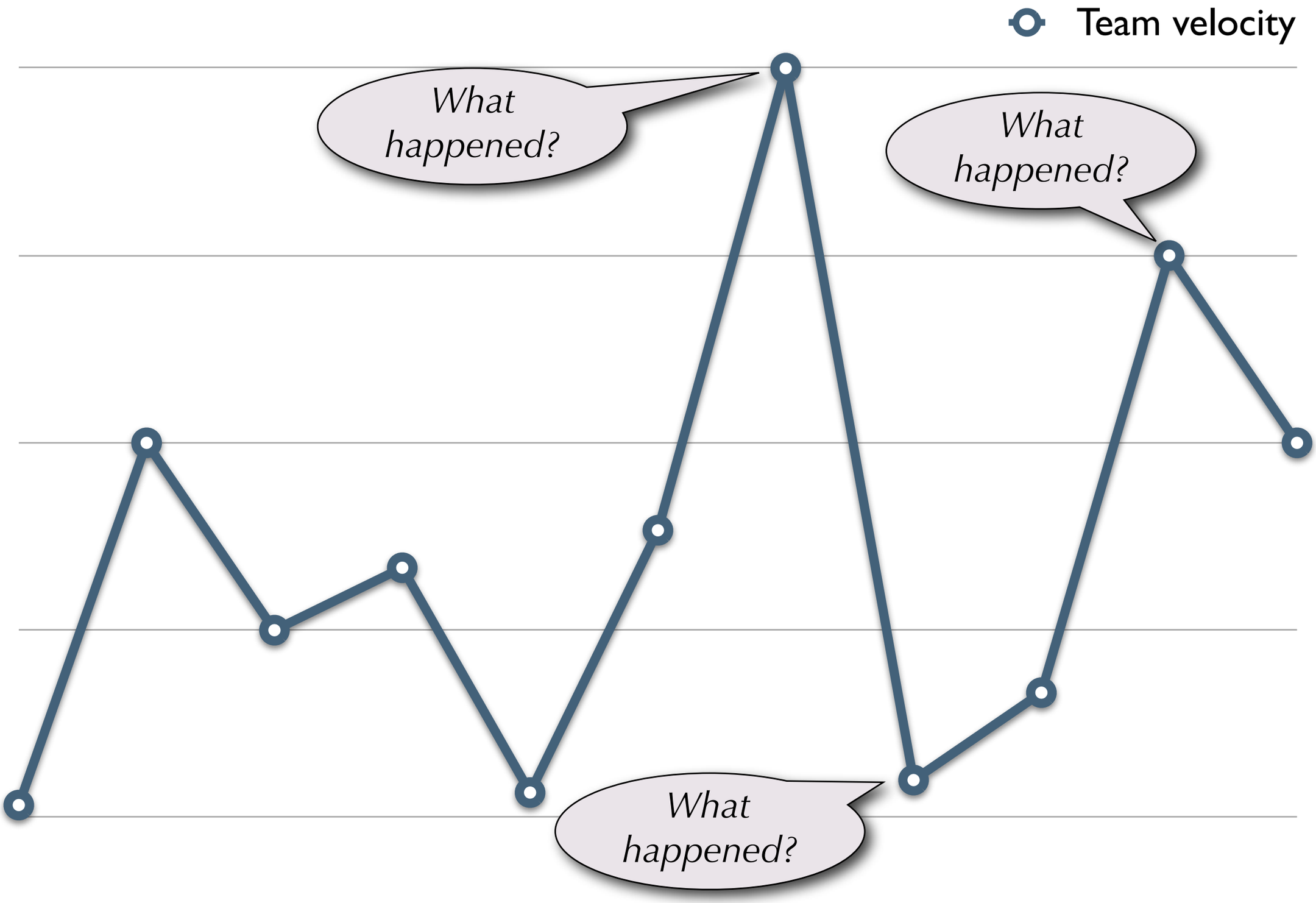
a priori:
argumentation

a posteriori:
justification

Experience

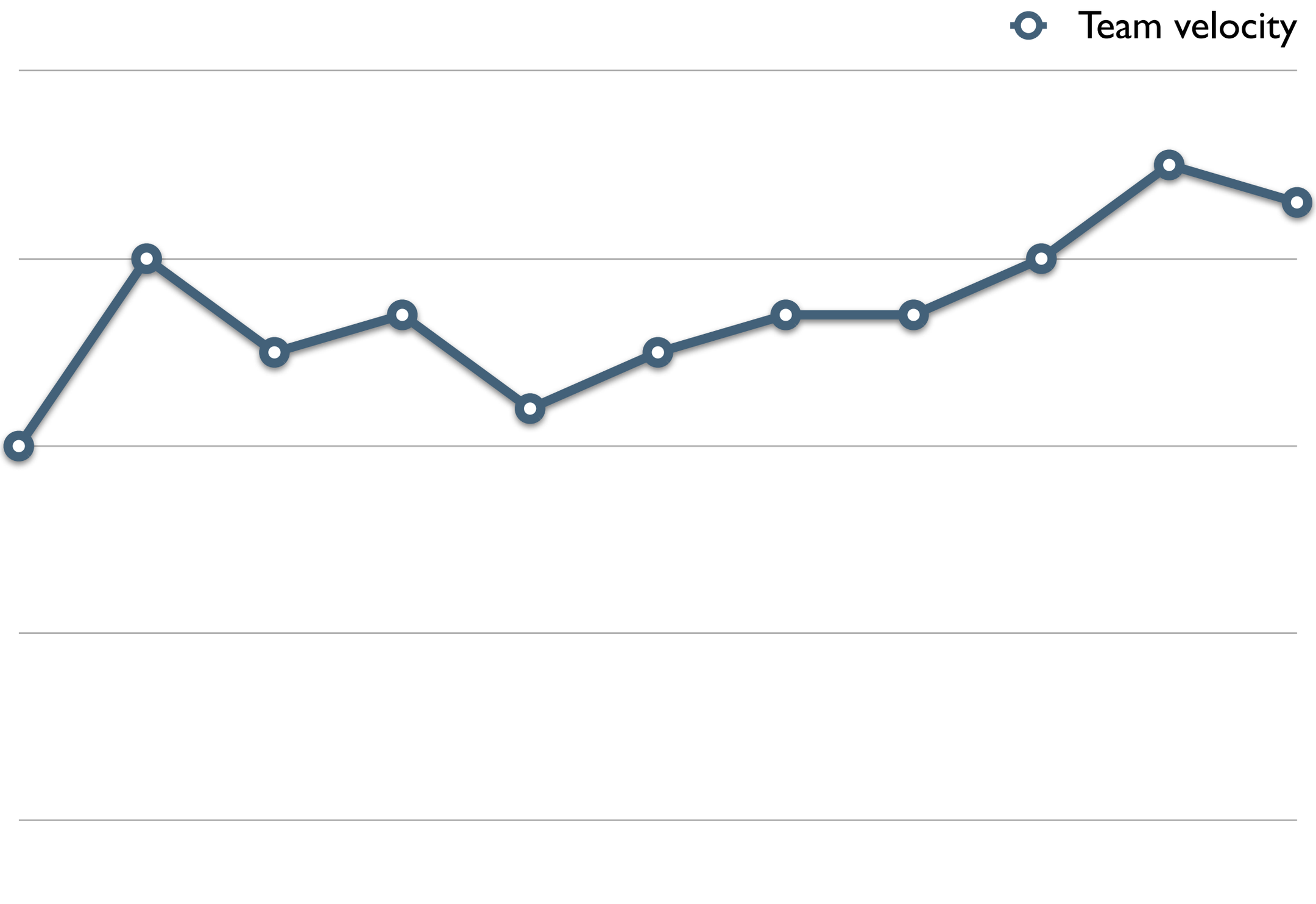
Time

Instability hurts



Since the empirical model is based on learning from experience, that experience should actually be a good basis for learning. An unstable situation does not allow this: what happened to make the current situation what it is? What would be the effect of a single change in the way we do things? In an unstable situation we just don't know.

Try to achieve rhythm and stability



Rhythm example

Rhythm allows **Speed**

Stability allows **Improvement**

Flow to READY,
Iterate to DONE

The nature of a Product Owner's work is closer to a flow than to an iterative process. Business requests come in constantly, and working on the backlog is a continuous process. The READY flow allows the iterations behind it to be stable. For this presentation I focus on the "Flow to READY" side.

A Backlog has "flow regions"

In Sprint

READY

Preparing

New

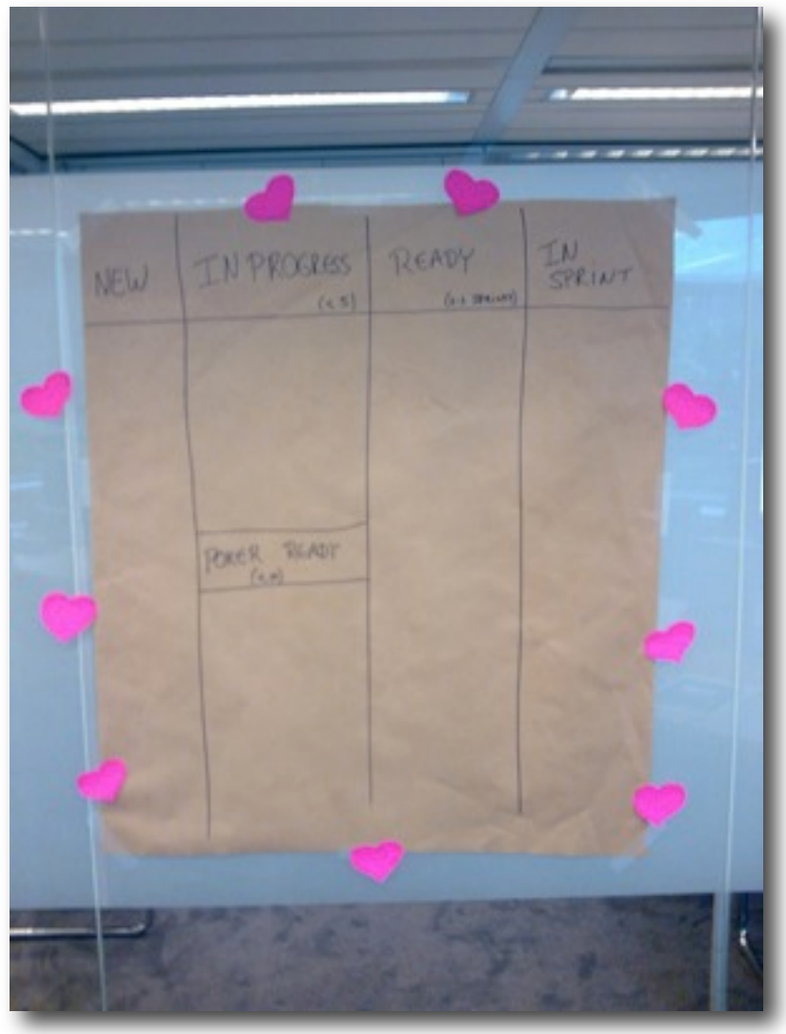
Note that this is an idealized view of things. In practice the lines are blurred somewhat because the mapping of priority on the workflow steps is not always as clean as you'd like. New items might show up really high in priority, putting it in between the READY items. On the other hand this way of viewing the backlog could be used to enforce the prioritization: something that's READY could by definition be prioritized higher than something that's not.

READY Kanban

New	Preparing	READY	In Sprint
• *Prioritized Buffer* • Unlimited size	• *Work In Progress* • Limited to PO capacity • Cycle time • Promotion model • Flow & Pull	• *Prioritized Buffer* • 1,5-2*Team Velocity	• *Work In Progress* • Limited to Team Velocity • Iterations

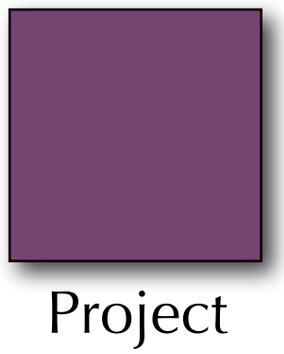
The preparing workflow step should be limited in Lean fashion to prevent thrashing. The main metric is the cycle time, or the average time it takes to take a New item and get it Ready.

READY Kanbans

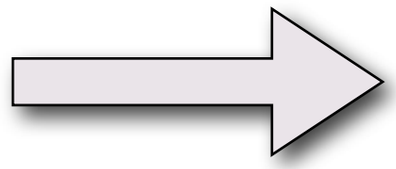


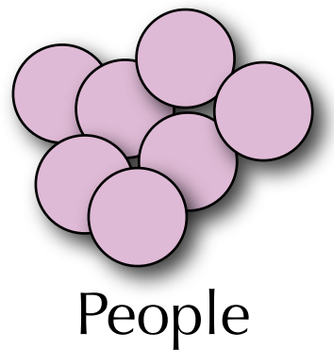
Stable teams *rule*

Portfolios *over* Resourcing

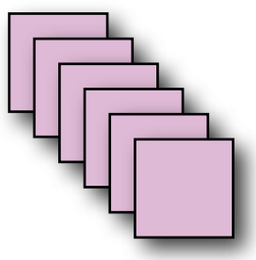


Project

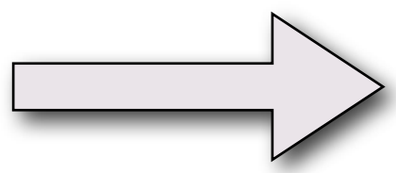


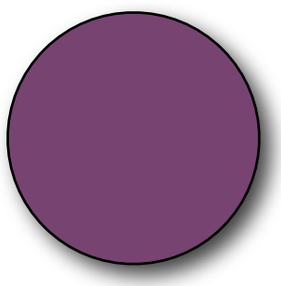


People



Portfolio





Team

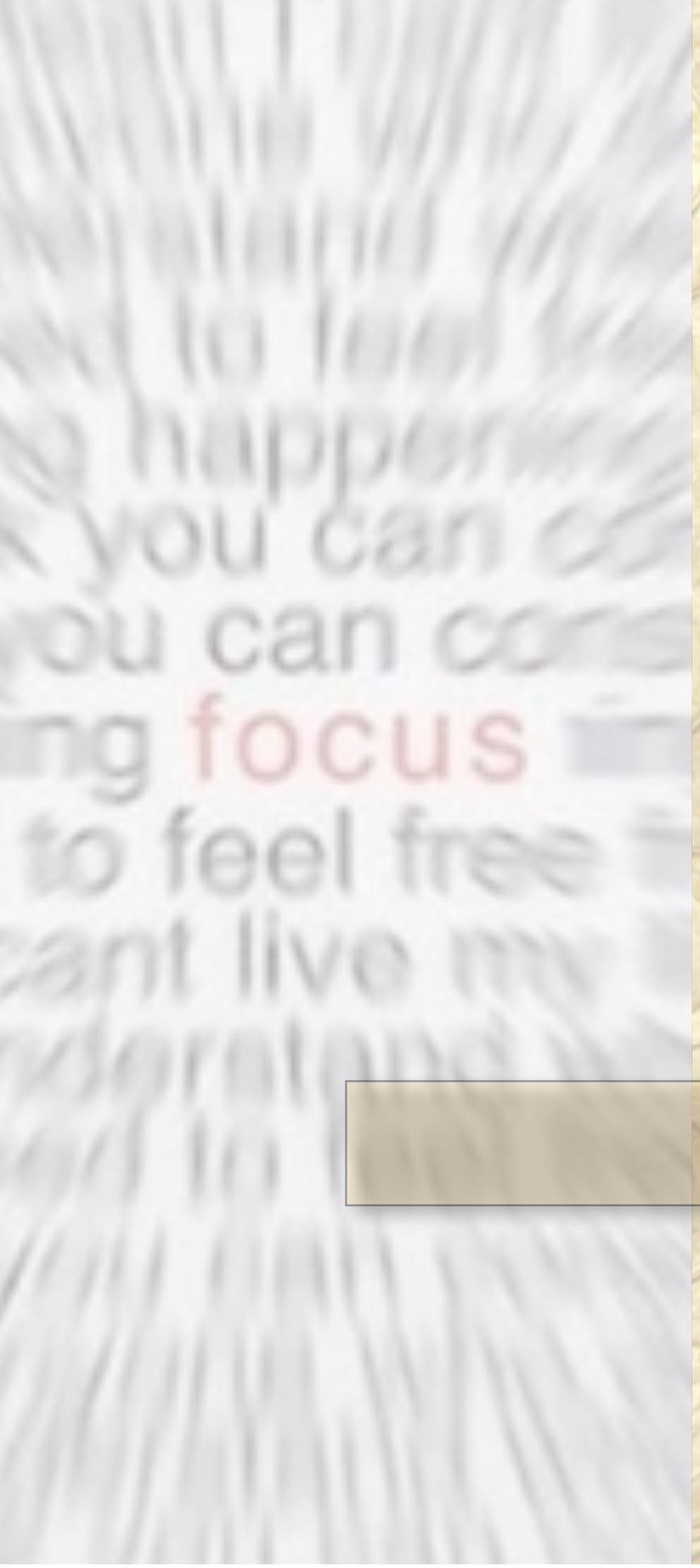
Portfolio Alignment Wall

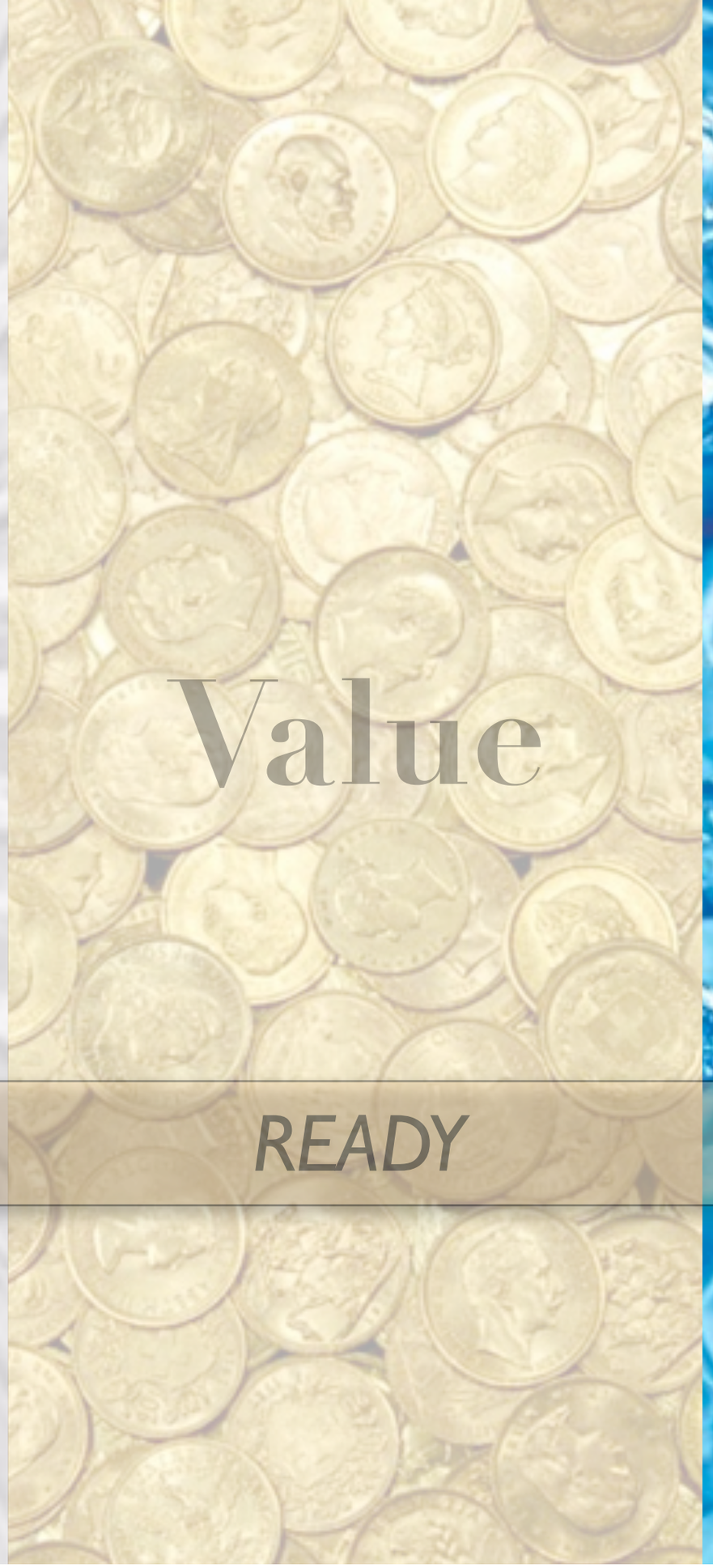
Team 1 | Team 2 | Team 3 | Team 4

Theme	Year Goals	Q1	Q2	Q3	Q4
Customer Experience	Reduce Customer Service contacts	CE1, CE2	CE3, CE4		
Operational Excellence	Improve website performance	OpEx1	OpEx2	OpEx3, OpEx4	
World Domination	Enter market X with existing product line	WD1			
Business Development	Introduce Product A	BD A1			
Business Development	Introduce Product B	BD B1	BD B2		

Portfolio Alignment Wall

Source: Arlen Bankston



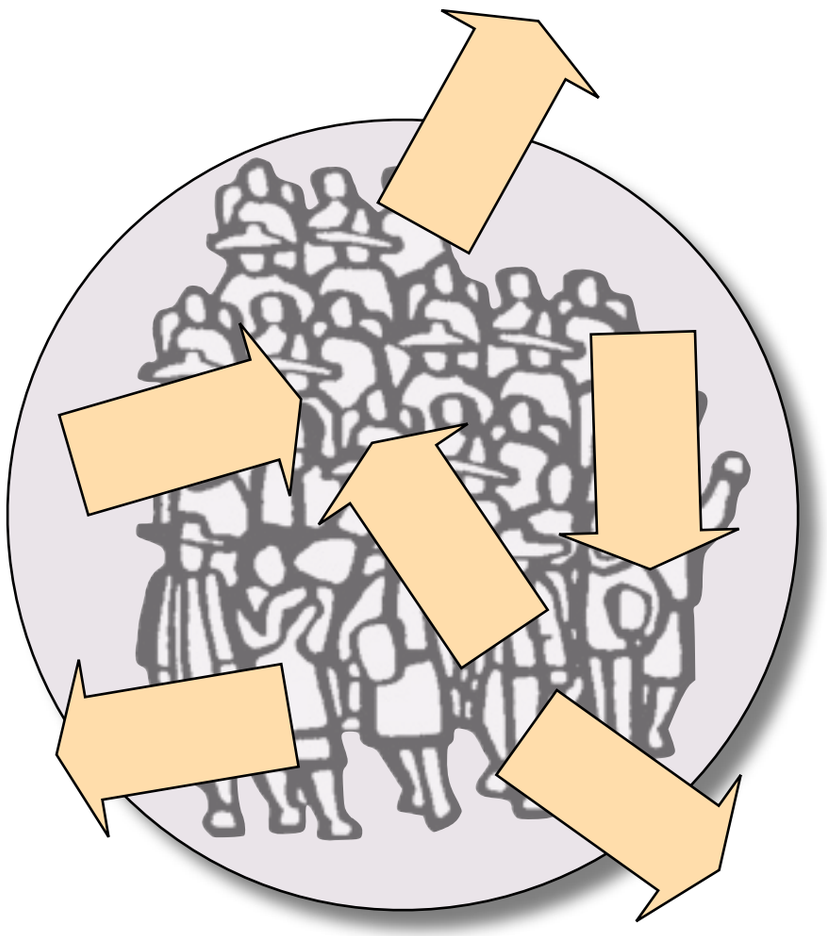


Value

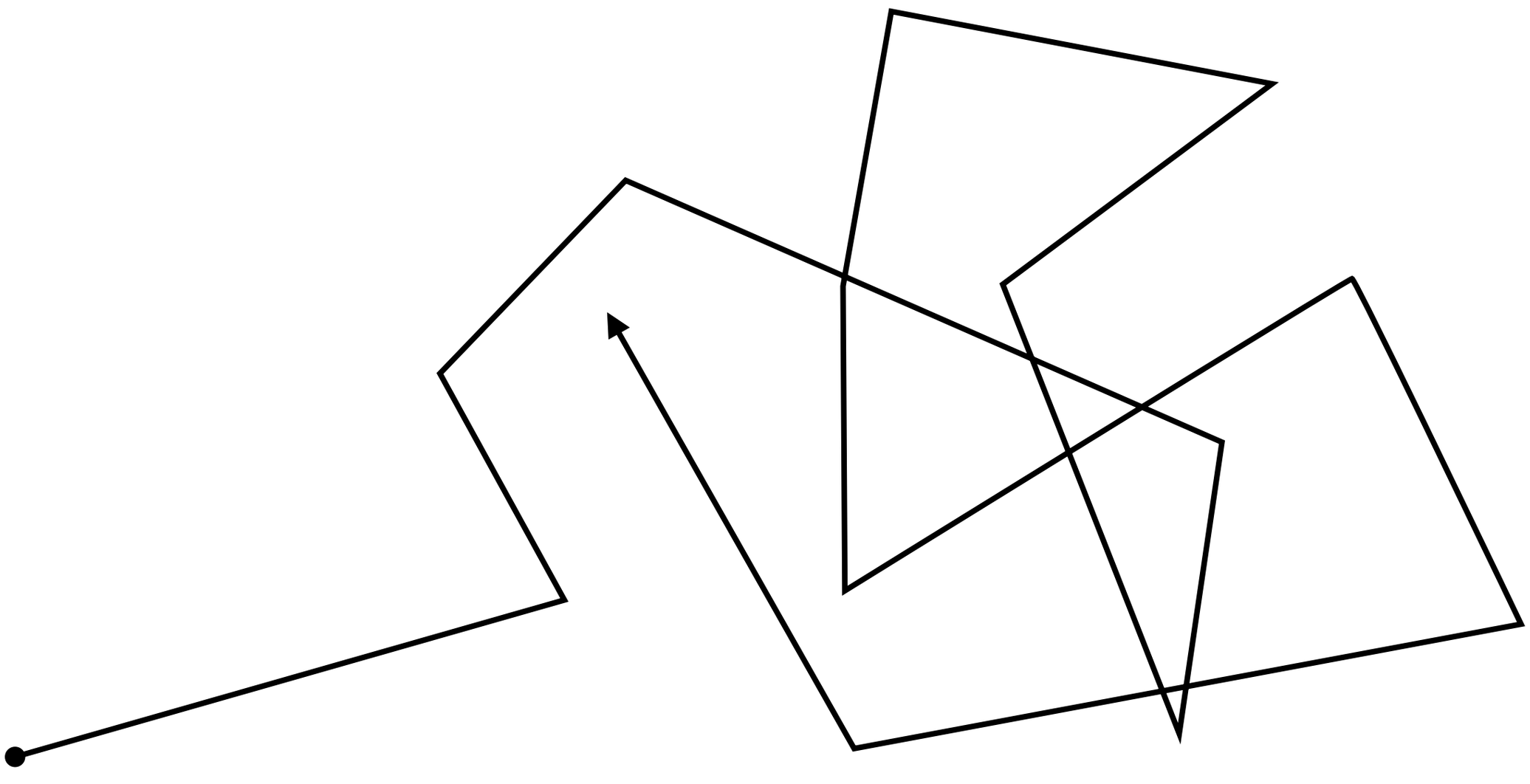
Flow

READY

Self-organization does not happen spontaneously

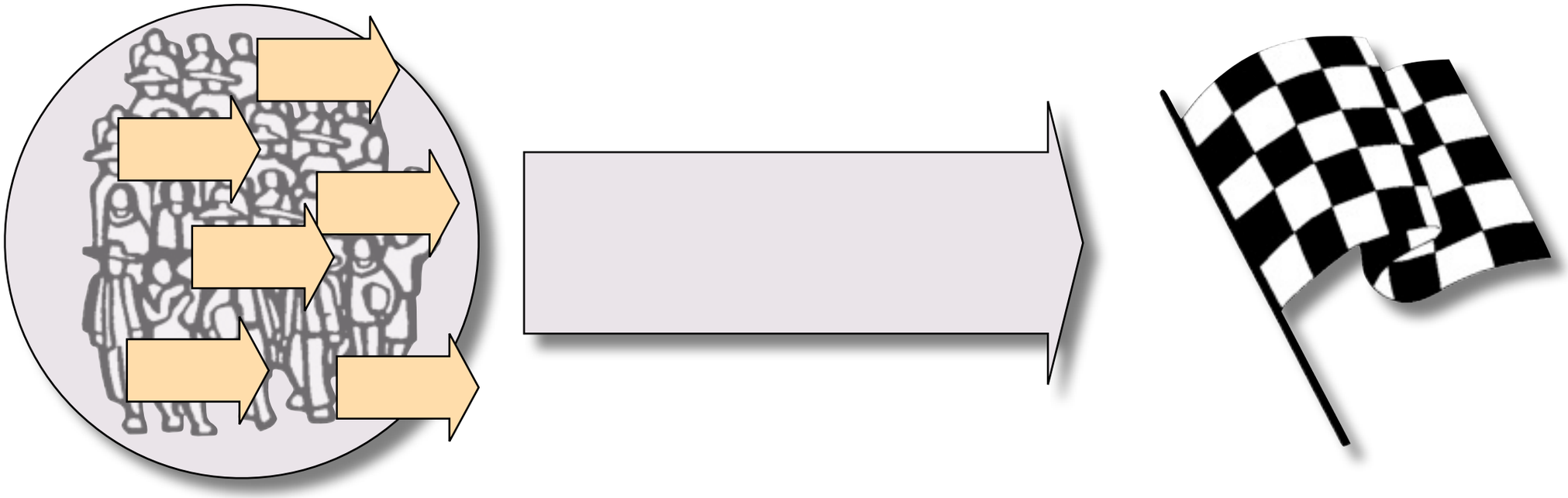


If you have no destination, any road will take you there

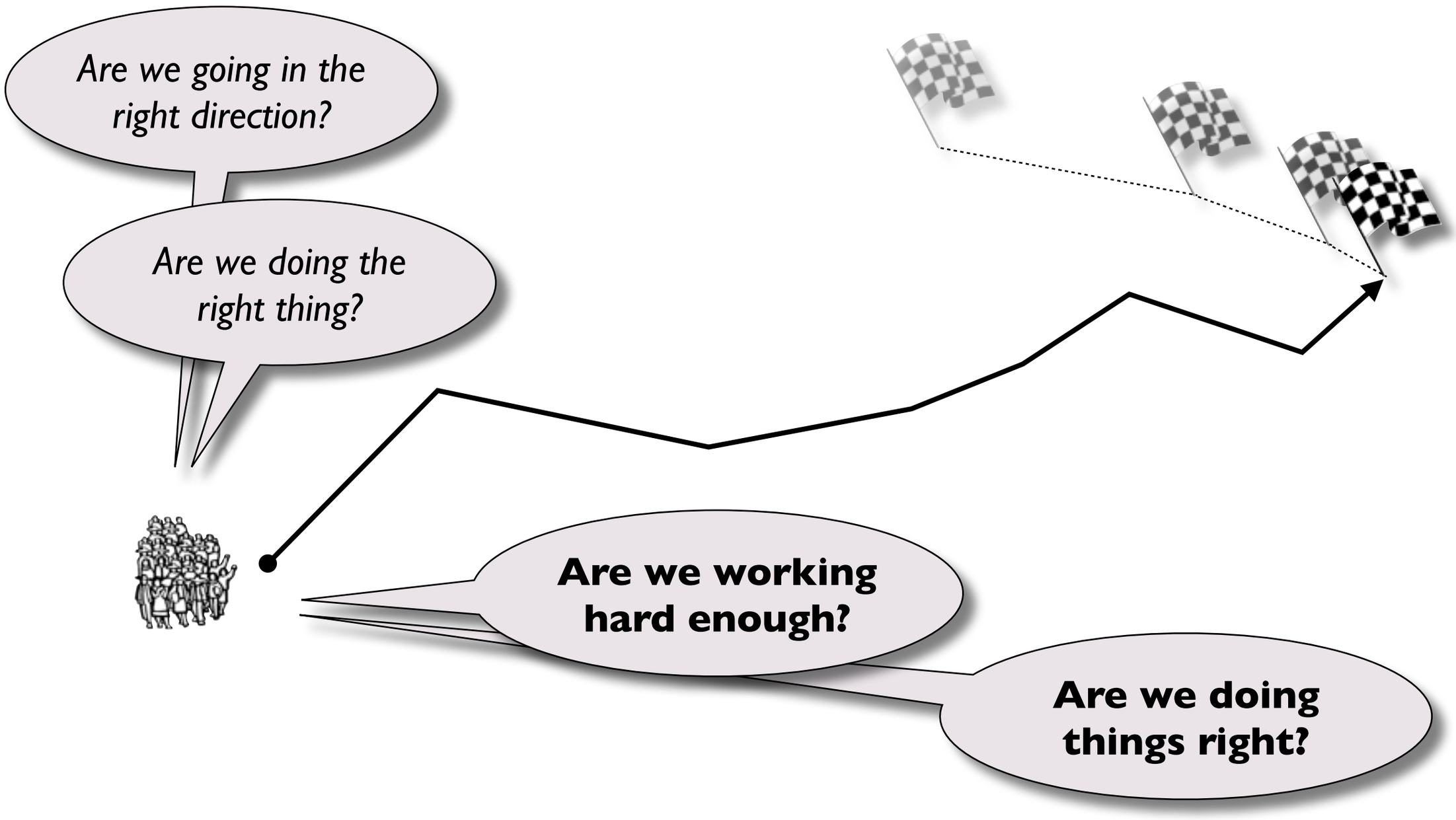


A trap of working in an incremental way is that you start thrashing. This happens when there is no longer term goal that is used as an aiming point. It is also typical when the business has a very reactive mindset, for instance changing directions with each new client's wishes.

Self-organization aligns towards a shared goal



Outcomes over **Activities**



...and to make this point even clearer: this applies to Scrum itself! If you have a better way to achieve the same outcomes (empirical learning, motivation, productivity, value...) then go for it!

Not just any outcomes, but **organizational** outcomes

Organizational **Goals**

Does the product *really* support the organization with its goals?

Product **Qualities**

You have tools to apply focus

Vision

Backlog
A
B
C
D
E
F
G
H
I
J
…

DoR

DoD

The Vision

Vision

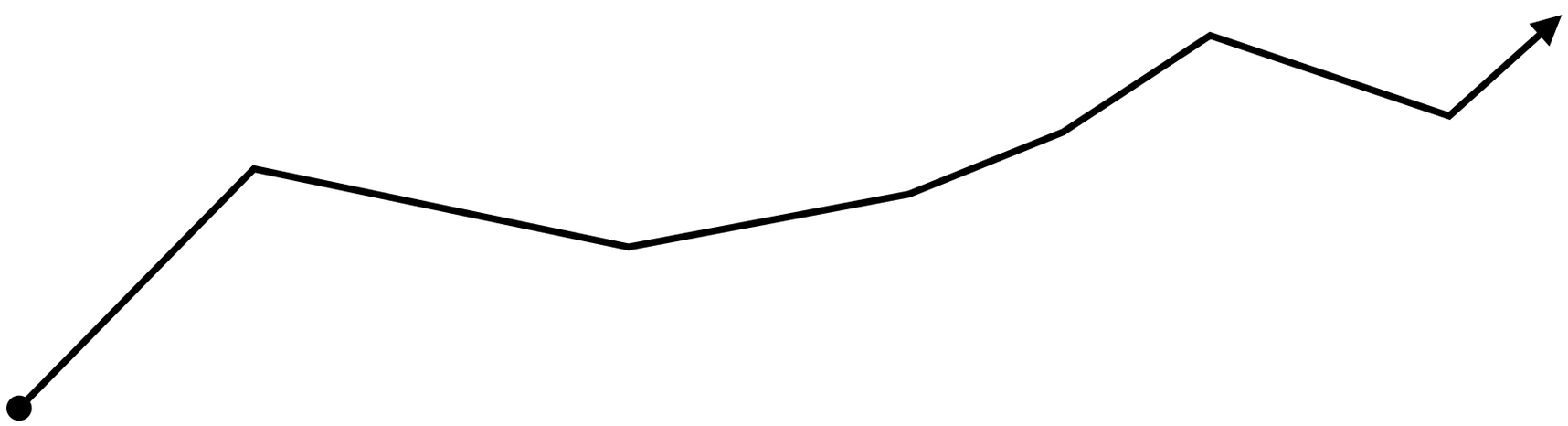
Backlog
A
B
C
D
E
F
G
H
I
J
...

DoR

DoD

Self-organization needs a *target* to organize to

Vision



An outcome vision is nothing radical to implement: you could use a sheet of A4 paper with drawings, a slide deck, or any other means to communicate a vision. The hard part in many cases is the need to make choices, or to have a vision in the first place. Politics, lack of decisiveness and fear of closing options are common barriers to a useful vision.

A Product Vision *orders* the backlog and provides *context*

Vision

orders

Backlog
A
B
C
D
E
F
G
H
I
J
...

provides context

If you have a clear vision of where you want to go with the product, you will find you will have a much easier time prioritizing the backlog than without it. You could say that the vision provides the ordering rules: indeed a good vision is simply a coarse-grained top-ten list of goals that is provided by upper management.

The team is enabled in its self-organization by knowledge of the larger goal. By providing context you will enable a team to come up with much better solutions. This is especially important to achieve the best value vs. cost ratio, which is definitely not going to happen if a team implements the exact lower-level goals (user stories) that a Product Owner provides.

The Definition of DONE

Vision

Backlog
A
B
C
D
E
F
G
H
I
J
...

DoR

DoD

The Definition of Done clarifies the finish line

Outcomes over activities

- Describe the *observed outcome*
- Be *wary* of activities
  - Focuses on ceremony, not results
- *Quantify*
- Add legal and standards *compliance*

✓Acceptance Tested
✓No Blockers or Highs
✓No increased technical debt
✓Documentation in order
✓Conform standard X

The DoD should - like the vision - describe what you can observe when you “snapshot” the outcome of a Sprint. A common pitfall is to describe activities, but that crosses into the “how” of things, which is within the mandate of the team itself. There are border cases like the common “acceptance tested”, but those generally exist because a better definition can’t be found. Like so many cases, the question “Why?” leads you to the right answer. Why do you want acceptance tests? If you want to ensure low bug counts, put a quality criterium for bug count in the DoD. If you want end users to sign off on the functionality, then put a “end users should be happy” criterium in.

Above all, be wary of putting “activity X done” items in the DoD. They put the wrong focus on the team, valuing going through the motions more than achieving results.

The Backlog

Backlog
A
B
C
D
E
F
G
H
I
J
...

Vision

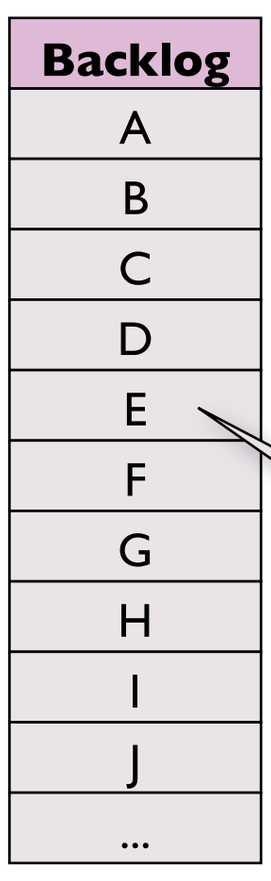
DoR

DoD

A Backlog item is *always* a delta

Where you are	**Where you want to be**
No feature	Feature present
Problem	Problem gone
Needs change	Change implemented

Deltas on the backlog, static requirements on the DoD



***Maximum* page refresh time allowed: 3 seconds**

DoD

***Improve* page refresh time: from 3 to 2 seconds**

Non functional performance criteria should go in the DoD, but initiatives to improve them should go on the backlog (deltas!). When a new performance level has been reached, you can update the DoD to ensure that the improvement is sustained.

The Definition of READY

Backlog
A
B
C
D
E
F
G
H
I
J
...

Vision

DoR

DoD

Answer three questions to get to READY

Why?

- **Value** quantified
- Business context, goals and problems clear
- Aligned with product goals

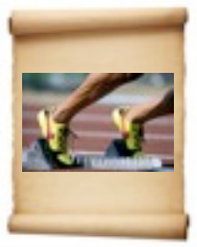
What?

- **Outcome** envisioned
- Acceptance criteria known

How?

- Implementation **Strategy** known
- **Cost** estimated (Story Points)
- **Small** (relative to Velocity)

So far I have found this to be a powerful Definition of Ready. It has a rough promotion model embedded in it: it's not useful to talk about the "what" if you don't even know "why" you're doing it, and it's not useful to think about the "how" if you have no idea "what" the required outcome is. In practice it's hardly this clear cut, you will need a bit of "what" and "how" even when answering "why", and a bit of "how" when answering "what". But thinking in terms of this rough promotion model does help in case of glaring deficiencies.

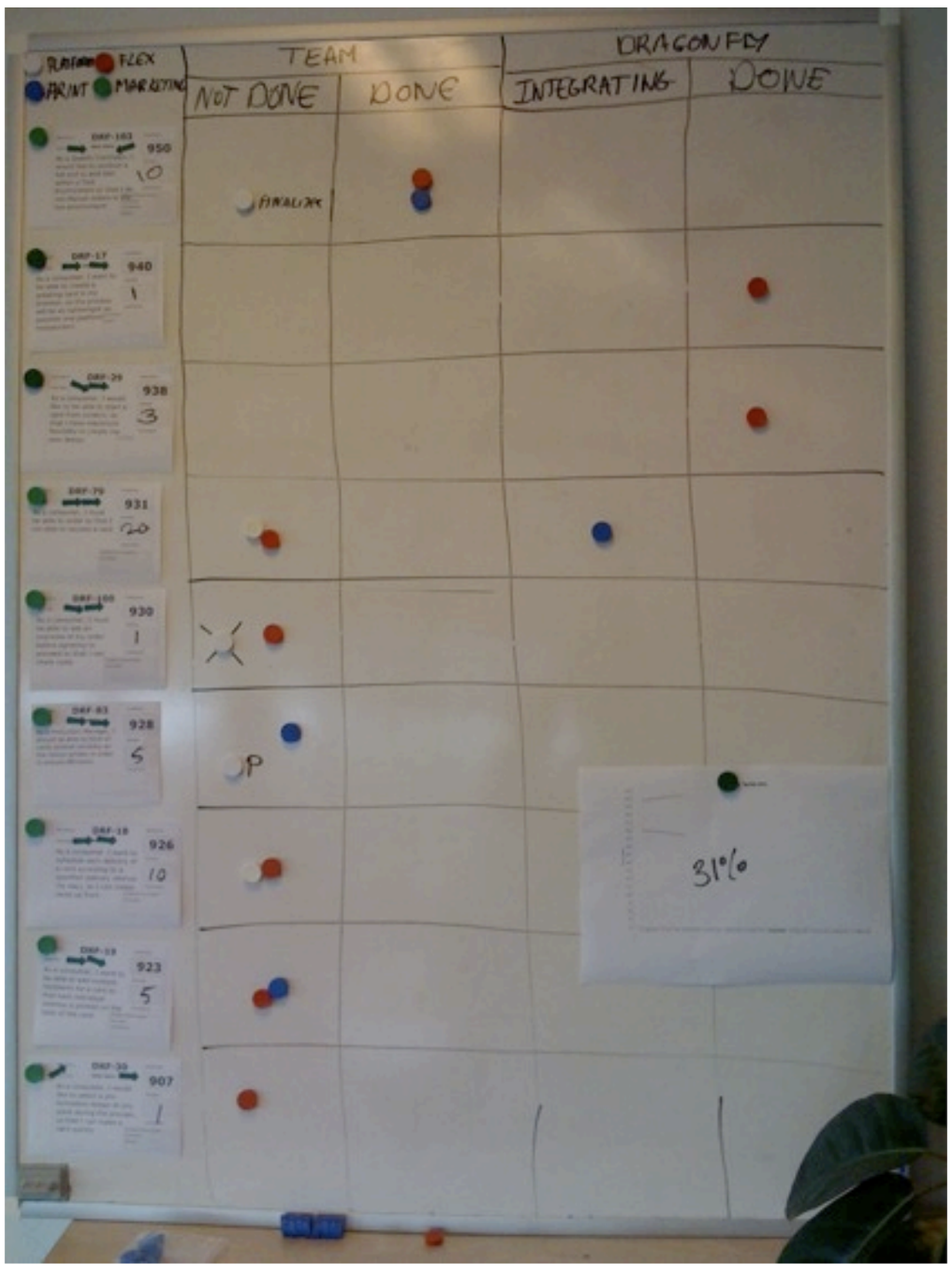


Your team will choke on Elephant Burgers

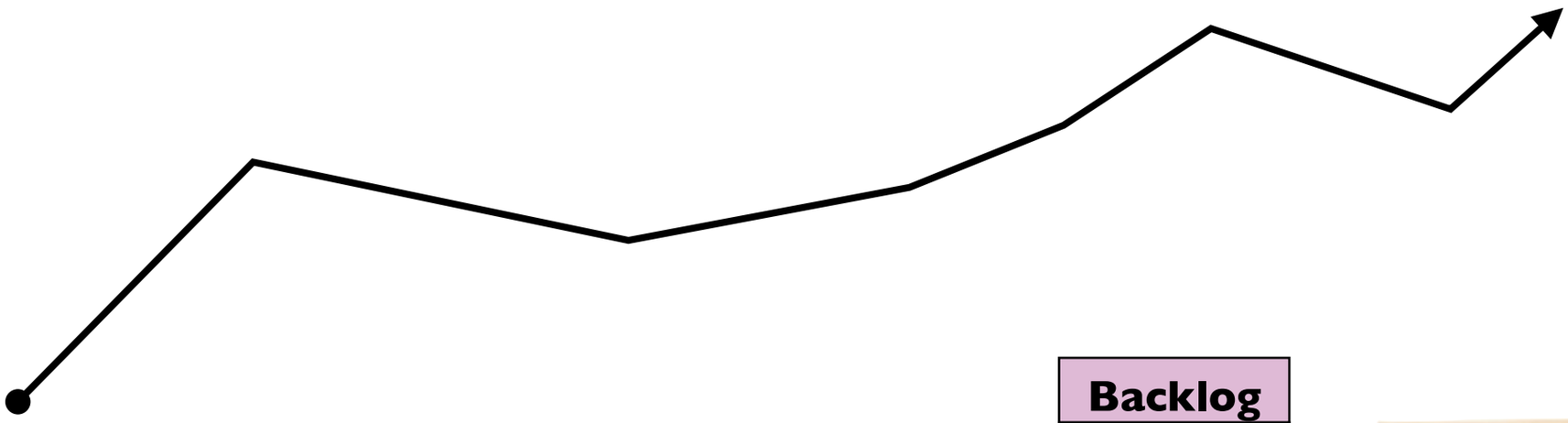
Tackle Icebergs outside-in

Roll-up board

- Team To Do, Team Done, Integrating, Total DONE
- Each (sub)team has its own color
- 1 Magnet per participating (sub)team



Apply focus to enable self-organization

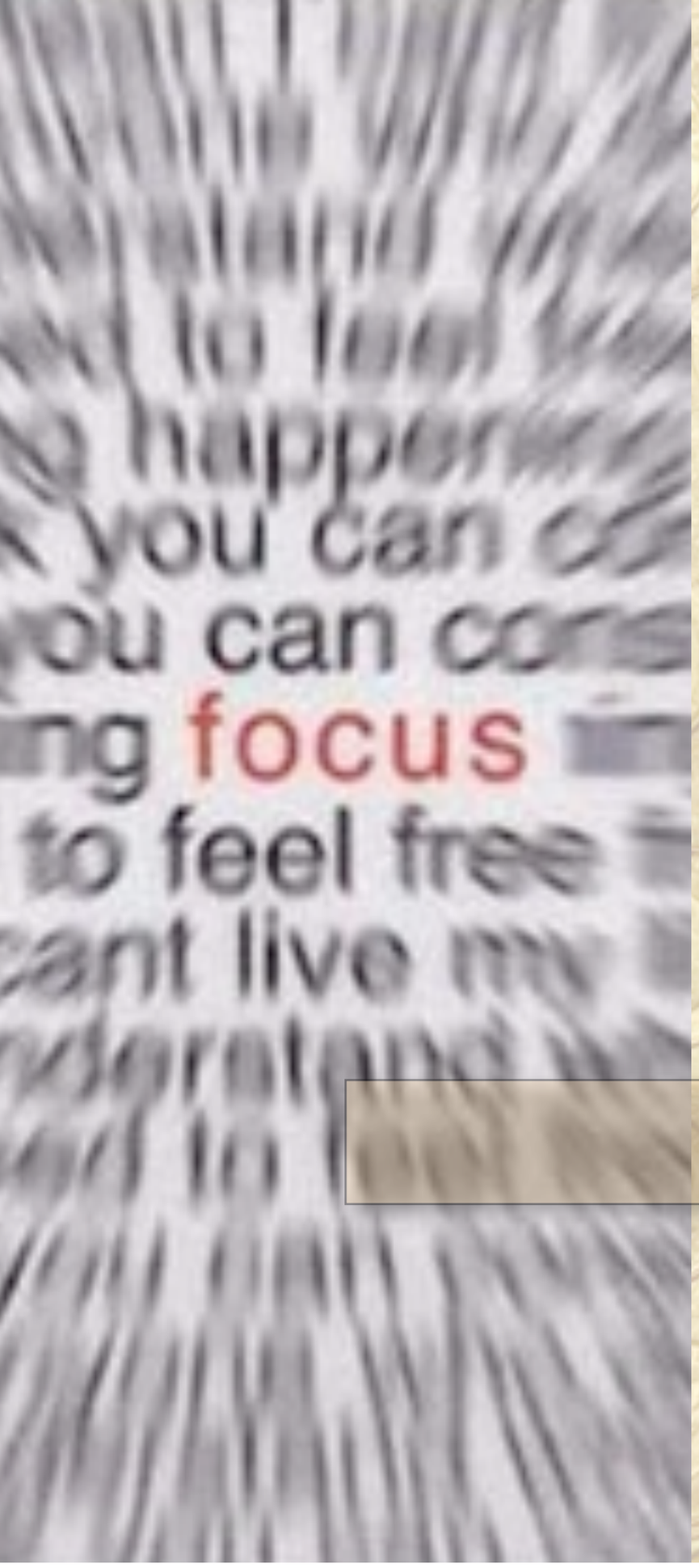


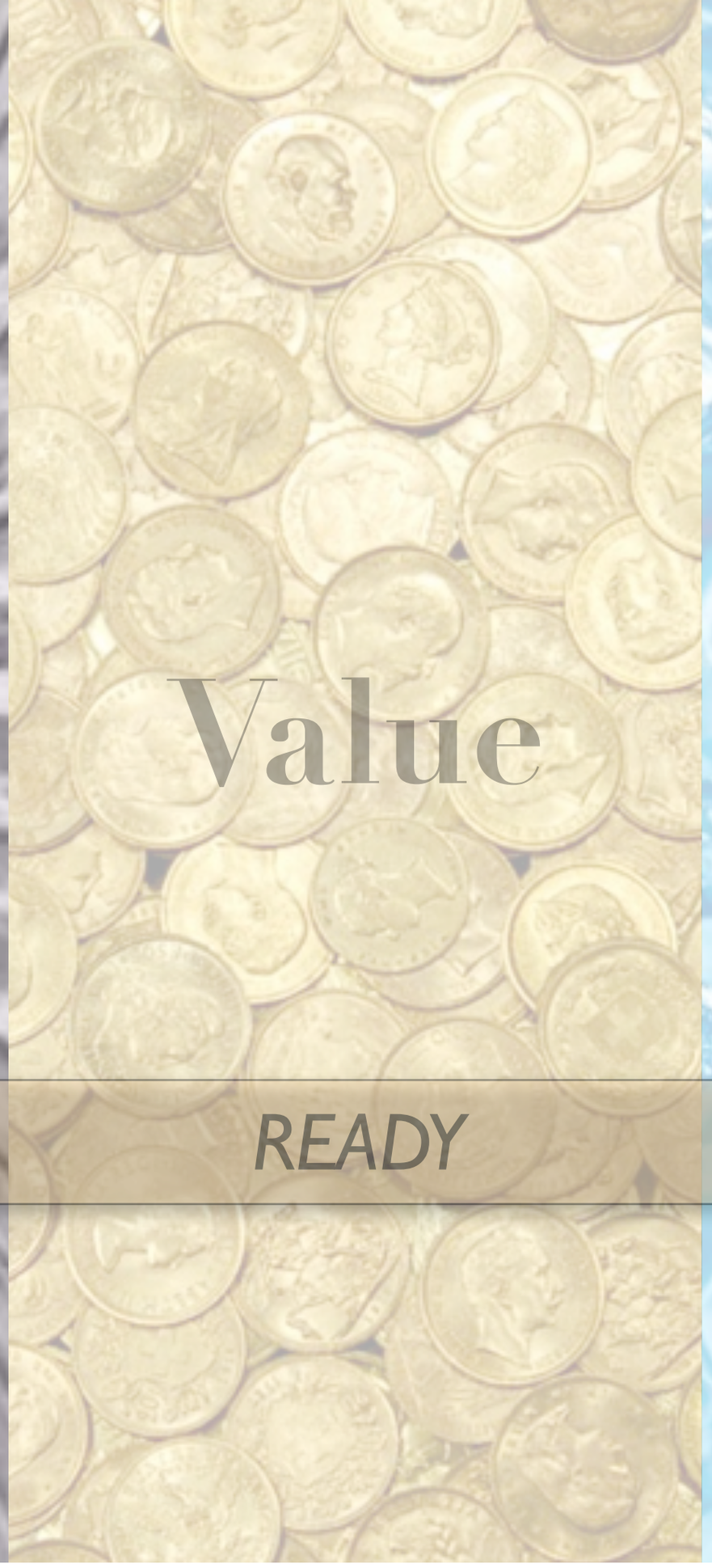
Vision

Backlog
A
B
C
D
E
F
G
H
I
J
...

DoR

DoD





READY

What is your main prioritization criterion?

In absence of anything rational, you'll invariably get "Prioritization by decibel meter"... It might even work for you, it's definitely the easiest method... ;-)

There is a large gap between user stories and business value

The Cause-Effect Trap

As a user
I want **a button**
So That **I can go to the next screen**

As a ... **I want a button** So That ...

Ask "Why?" *a lot*

...I can achieve my
FTE savings goals

Why?

...I can save time
on the usual cases

...I can **short-cut**
the order entry

Why?

...I can go to
the next screen

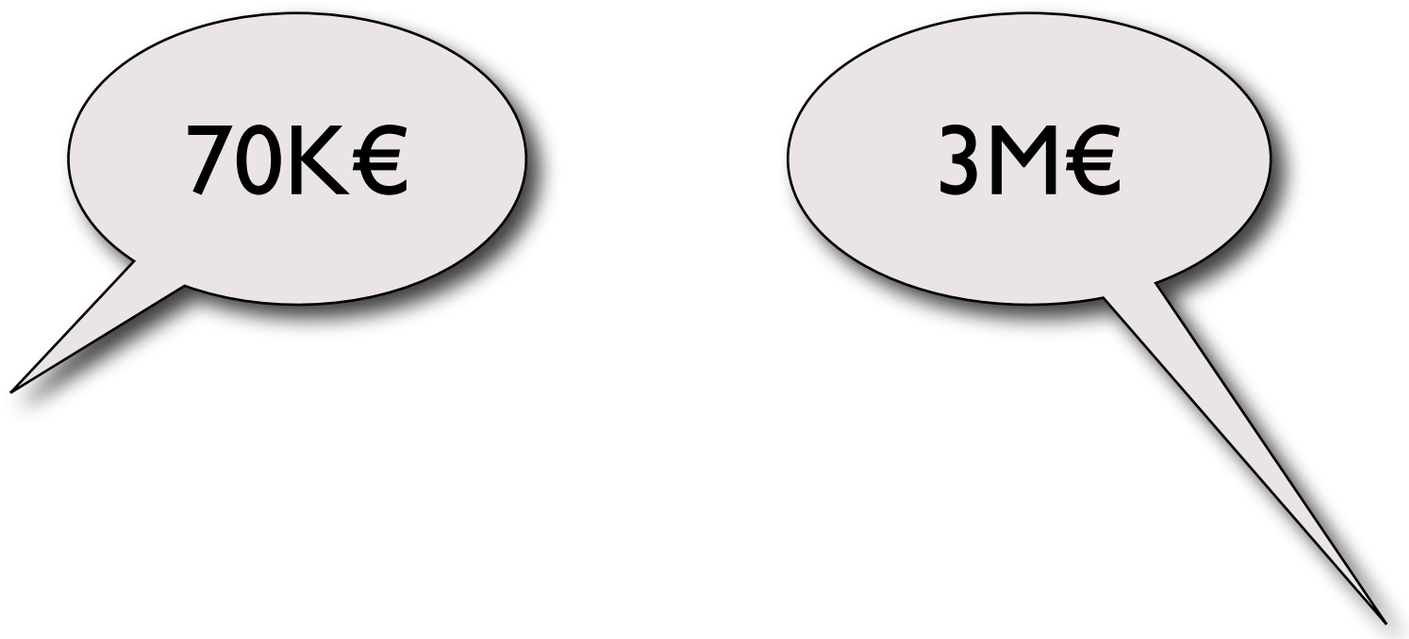
Why?

The Cause-Effect Trap ***Fixed***

As a department manager
I want **short-cuts in the order entry**
So That **I can achieve FTE savings**

As a ... **I want a button** So That ...

Objectify the discussion through quantification



To me quantification was illustrated best when the CFO forced his CEO and Customer Service Manager to put their respective favorite feature - that they had been arguing over as to which was to be done first - through a (simple!) spreadsheet with defined corporate business case metrics. It turned out that the CEO's feature would net 70K, and the Customer Service Manager's would net 3 million. Even with a large fudge factor the difference was clear. The simple act of quantification turned it from an emotional, decibel-controlled discussion to a rational and reasonable one.

Quantification is not Metrics

Past

Measured Performance

Future

Quantified Goals

Use Planguage for quantified requirements

Scale

X per Y

Meter

How to measure?

Now, Goal, Wish,
Stretch, Survival

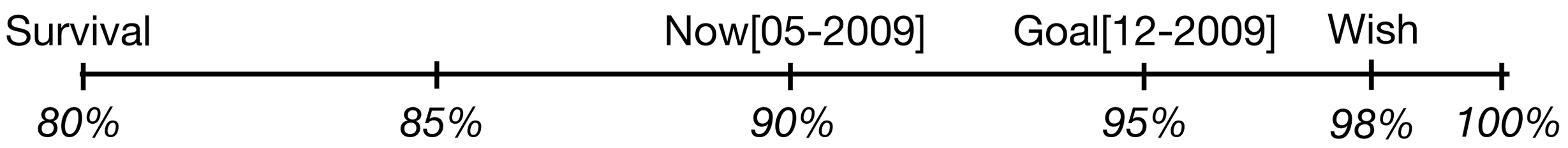
Points on the scale

Goal[Sprint 4, Subsystem A]

Qualifier

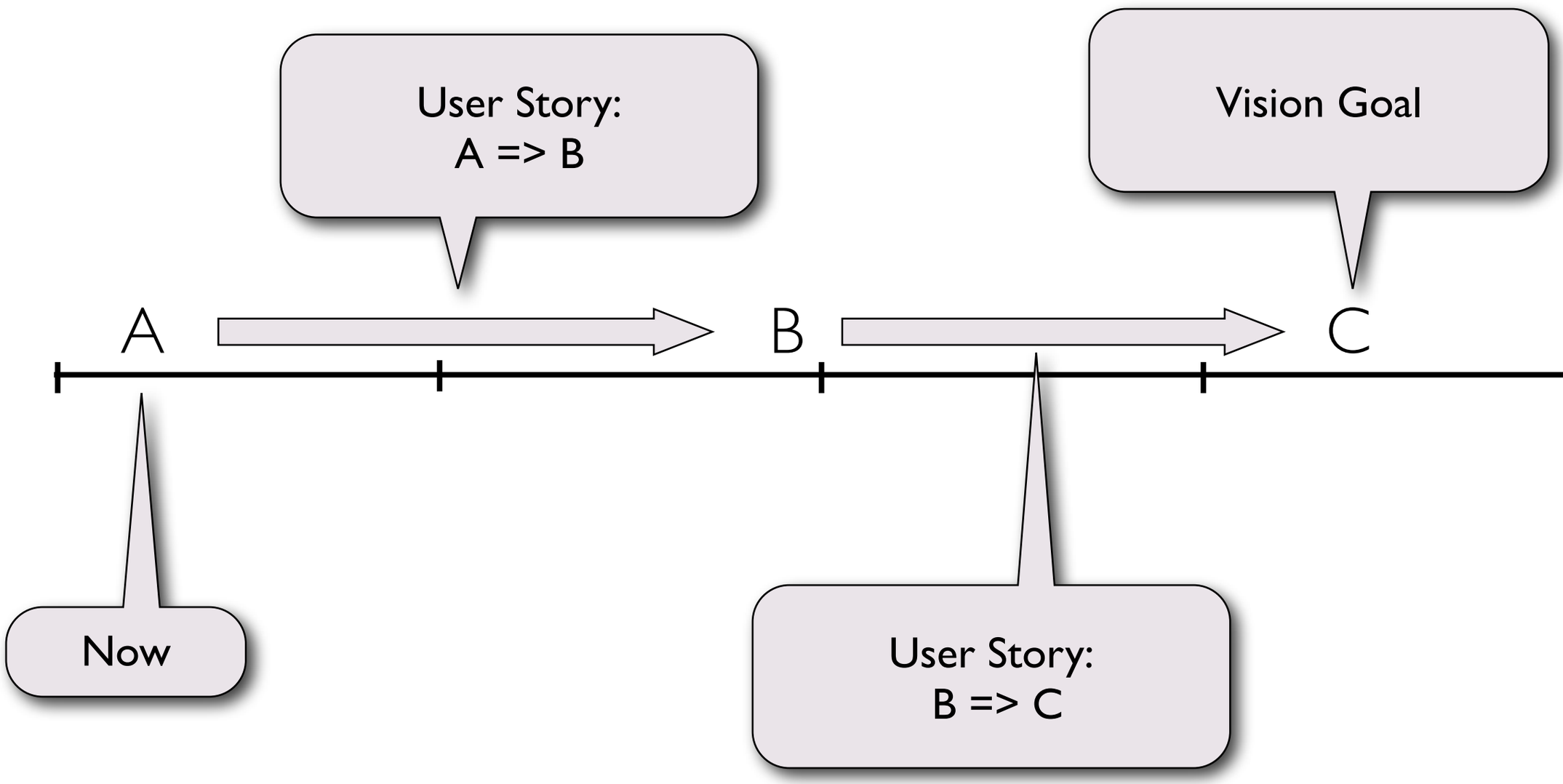
Planguage was created by Tom Gilb, http://www.gilb.com

Scales of Measure



Tag: CustomerCase.Resolution.Speed
Supports: Cost.CustomerService
Scale: *The percentage of costumer cases that is handled within 24 hours*
Meter: Monthly call center report
Survival: 80% <- Customer Support Manager
Now[05-2009]: 90% <- SWAG Serge
Goal[12-2009]: 95% <- Director of Customer Experience
Wish: 98% <- Director of Customer Experience

From Now to Vision in Steps



Portfolio

1. Goal 1
2. Goal 2
3. Goal 3

Business Goals

Why?

Product

Product Vision

1. Goal 1
2. Goal 2
3. Goal 3

Product Goals

Why?

Release Backlog

Marketable Feature Set

Why?

Product Backlog

User Stories

Incrementing

Source: Arlen Bankston

Incrementing calls for a fully formed idea.

On time delivery requires dead accurate estimation.

Slide copied from Arlen Bankston's CSPO material.

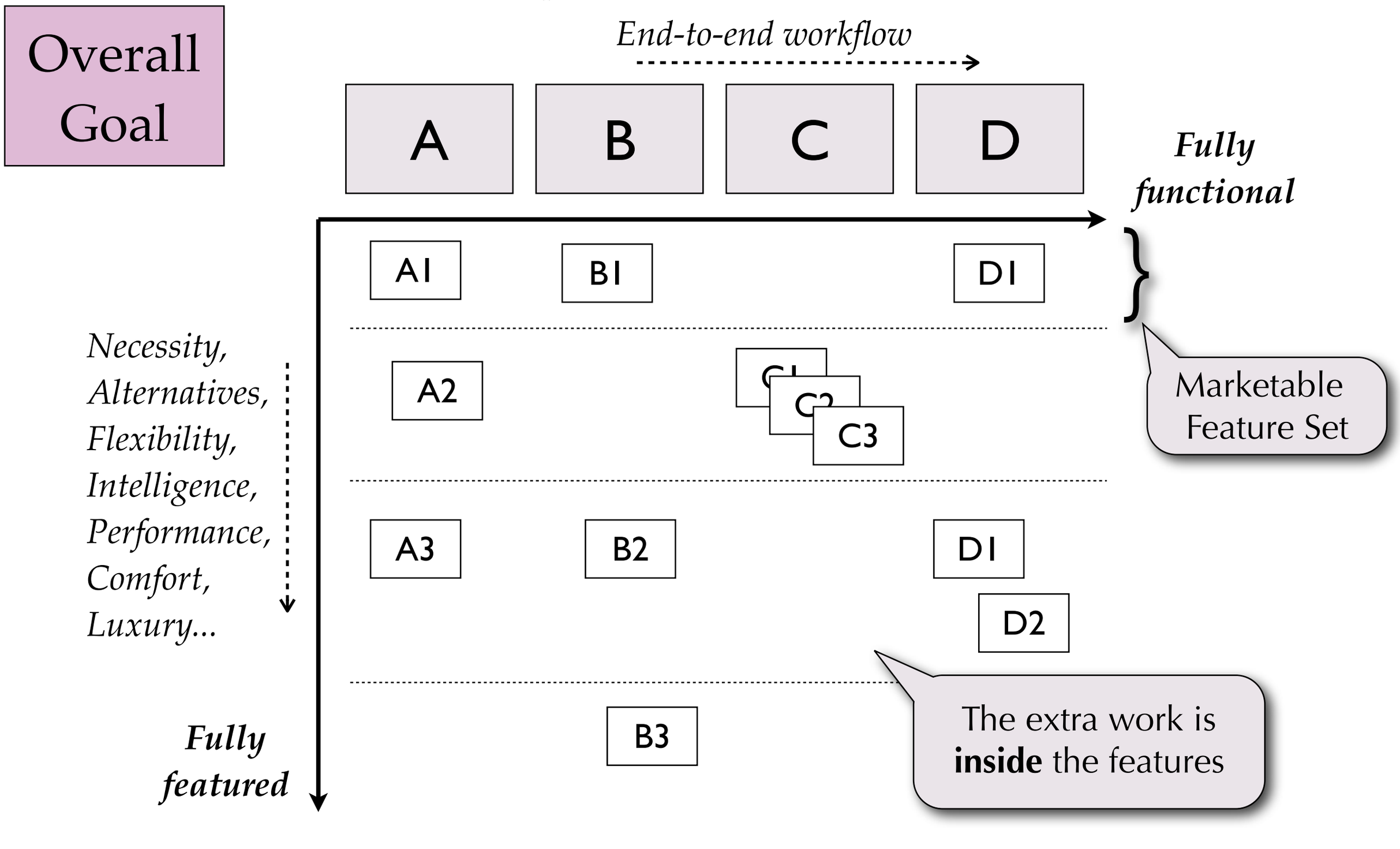
Iterating

Source: Arlen Bankston

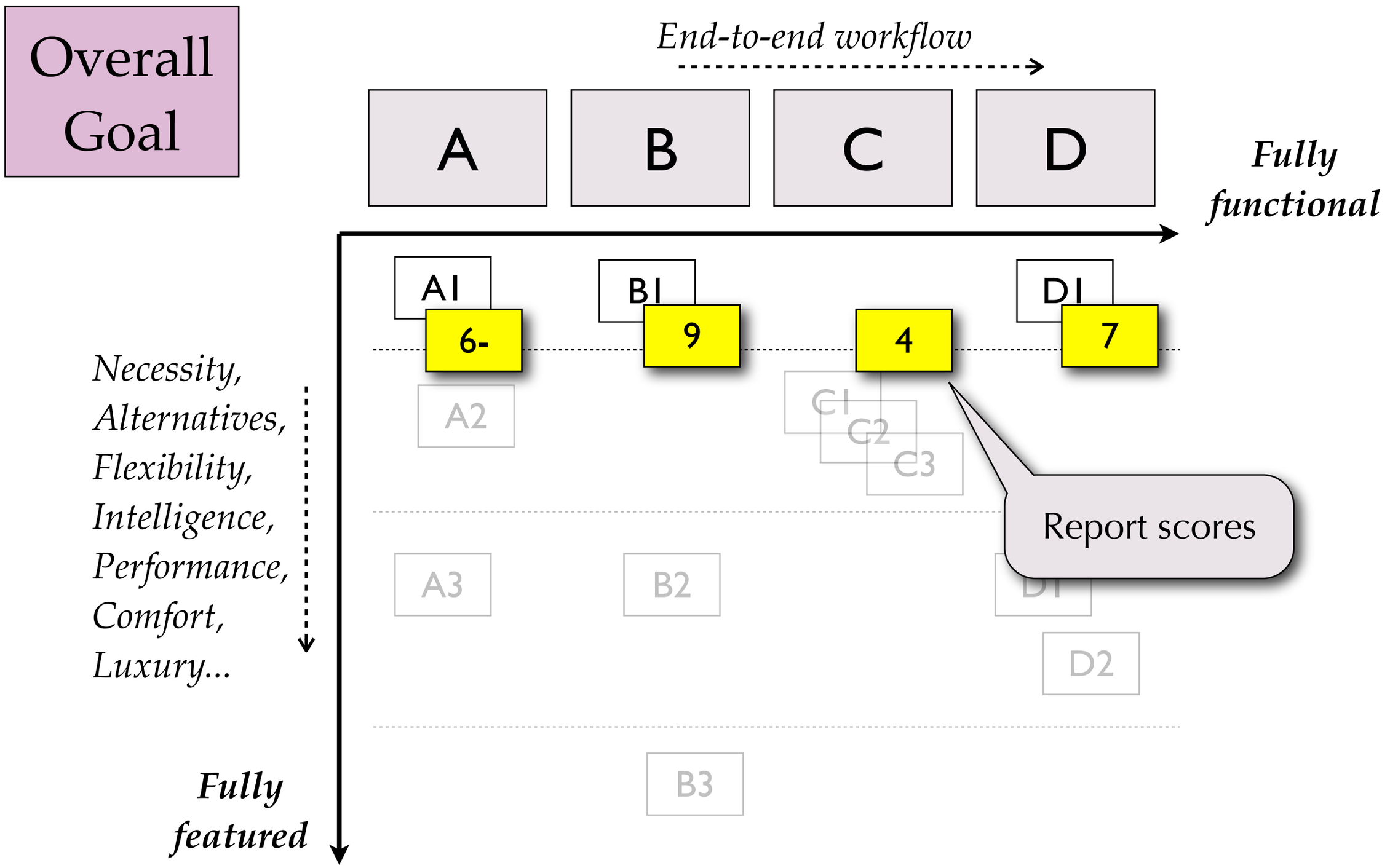
Iterating allows you to move from vague idea to realization.

Slide copied from Arlen Bankston's CSPO material.

Fully **functional** versus fully **featured**



Control per iteration



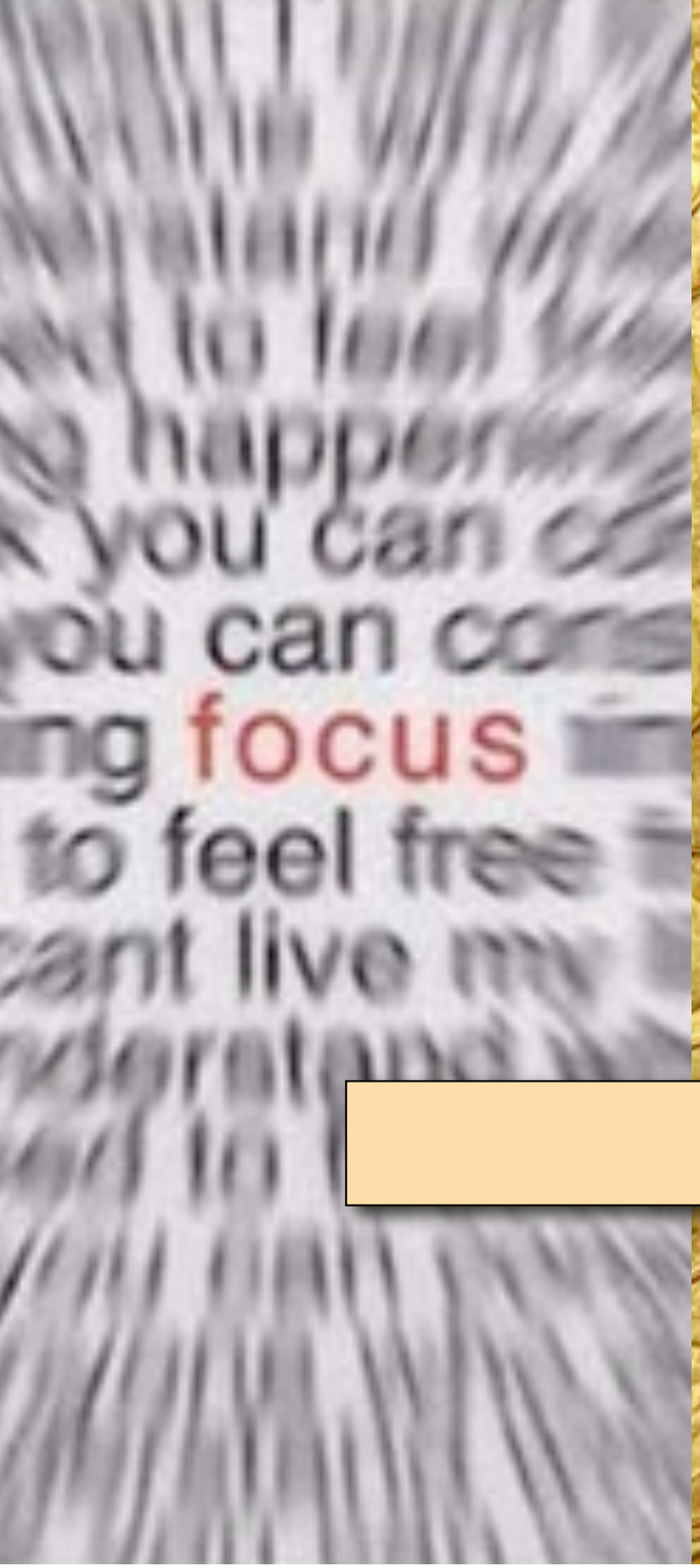
Value

READY

You can take off the T-shirt

Backlog
A
B
C
D
E
F
G
H
I
J
…

You can deliver ROI



READY

Thank you

Serge Beaumont

sbeaumont@xebia.com