PEOPLE LOVE TO DEBATE ABOUT THINGS.
PROBLEM IS: IN SOFTWARE ARCHITECTURE,
WE NEED DECISIONS AFTER THE DEBATE.

"Either Have Taste or Have a Software Architecture – Not Both!"

Matthias Bohlen <mbohlen@mbohlen.de>

LET'S COLLECT SOME TOPICS OF DEBATE

What opinions did you have debates about?

Put stickies here (use cmd-D or ctrl-D to duplicate):

- Green: your proposal
- Red: "their" proposal

Double-click on a word to edit it

Zoom in or out with your mouse wheel (or with two fingers on a touchpad)

Material UI vs other Design Framework vs DIY

standardize message structure of Kafka to include a "header"

no, wait till we know more

Functional JS

OO Javascript

Delivering well understood features > Software Crafting Skills

In a problematic code base, crafting might be more important

Kubernetes

Cloud Native

yours

yours

ThreeDiToolbox.threedi_plugin INFO The selected 3di model spatialite: None

microservice open for all languages

Migrate from premises to the cloud for all services

Start with the most problematic (interconnected) service and re-evaluate then

Microservice

Quality Goals

Serverless

CQRS/ES

Microservices everywhere !

We're agile, we will worry later

Do a proper modular monolith before going distributed

Monorepo

yours

Isolated DB per service

SCS vs Multi-Layered Microservices

One big RDBMS

all microservices typescript

React vs Angular

Always separate blocks/components that have different concerns or (technical) interfaces (and be able to scale them as needed) - despite the overhead of "wasted" resources or higher development time

Relevance

Data schema before model

Use of gRPC in microservices communications

do not require testers

Let's get started with polling/stateless.

We must use websockets!

get rid of ActiveX solution

postpone it until it's too late

GraphQL

AWS Lambda vs. ECS

Start with simple design and improve it aftewards if it doesn't meet performance requirments

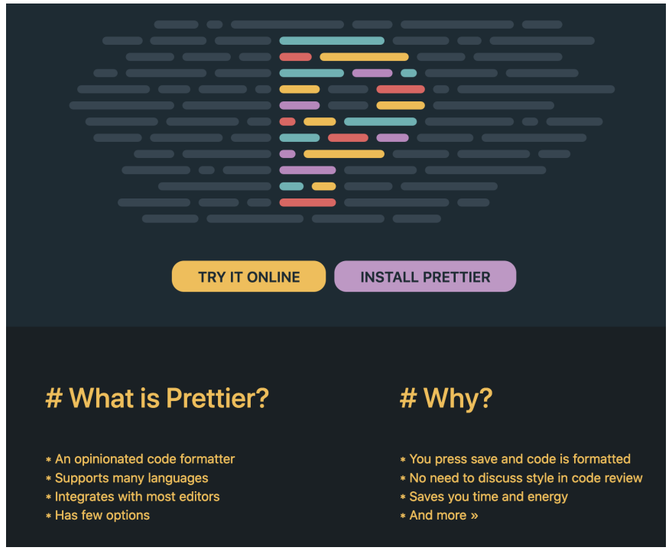
Solution may not meet the performance requirement s we should include more components in place alread

We should leverage postgis functionality as much as we can.

We must use the observer pattern due to the complexity of the domain.

Embrace full native cloud paas

WHY DO PEOPLE HAVE DEBATES?



https://prettier.io/ makes most of these choices **non-configurable**, ending the debates about this.

Example: Code formatting

How long did you discuss these questions?

Should a single lambda parameter have parentheses around it?

```
const putMarketingDocument = useMutation<MarketingDocument, unknown, MarketingDocument>(
    mutationFn: async (md : MarketingDocument ) ⇒
        axios
            .put<MarketingDocument>( url: `${apis.server}/marketing-docs/${edgeNodePair.link.id}`, md)
            .then((response : AxiosResponse<T> ) ⇒ response.data),
);
```

Should the keyword "else" better be on the next line?

```
if (result === undefined && this._undefinedResult) {
    this._undefinedResult();
} else if (result !== undefined && this._normalBehaviour) {
    this._normalBehaviour(result);
}
```

PEOPLE DEBATE B/C THEY OPTIMIZE FOR DIFFERENT GOALS

Architecture is not as easy to decide as code formatting.

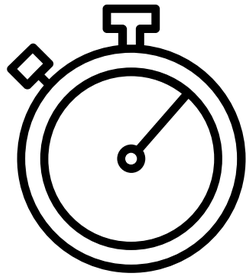
Architectural debates might take 10x longer!

I want performance.

"Let's add a cache."

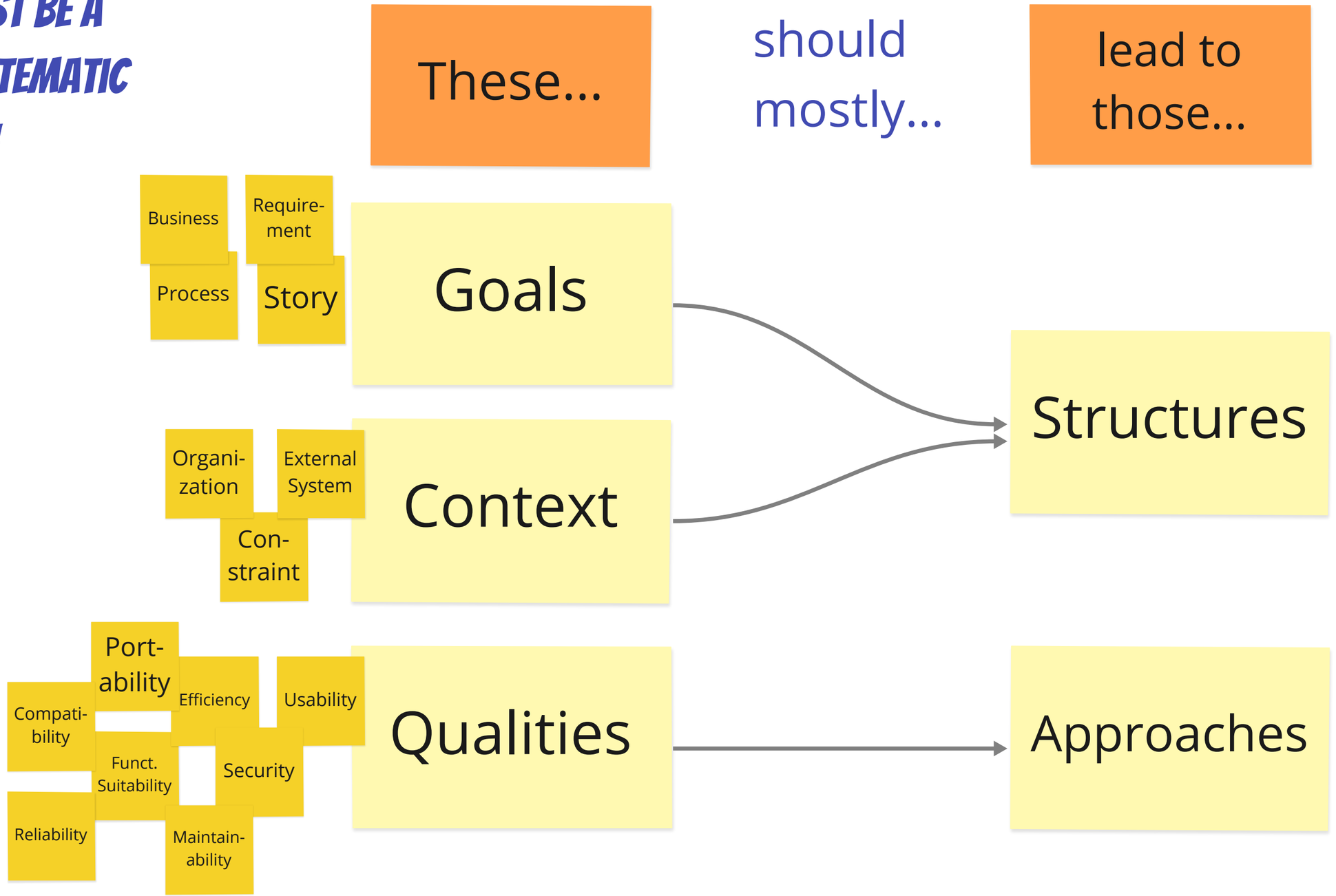
I want a system that's easy to maintain.

"Let's avoid a cache."



Do we have time for this?

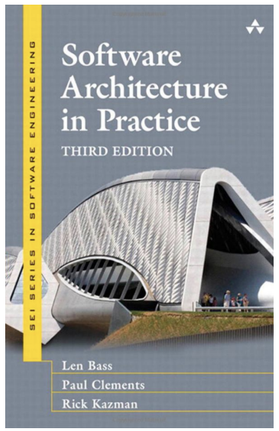
THERE MUST BE A MORE SYSTEMATIC APPROACH



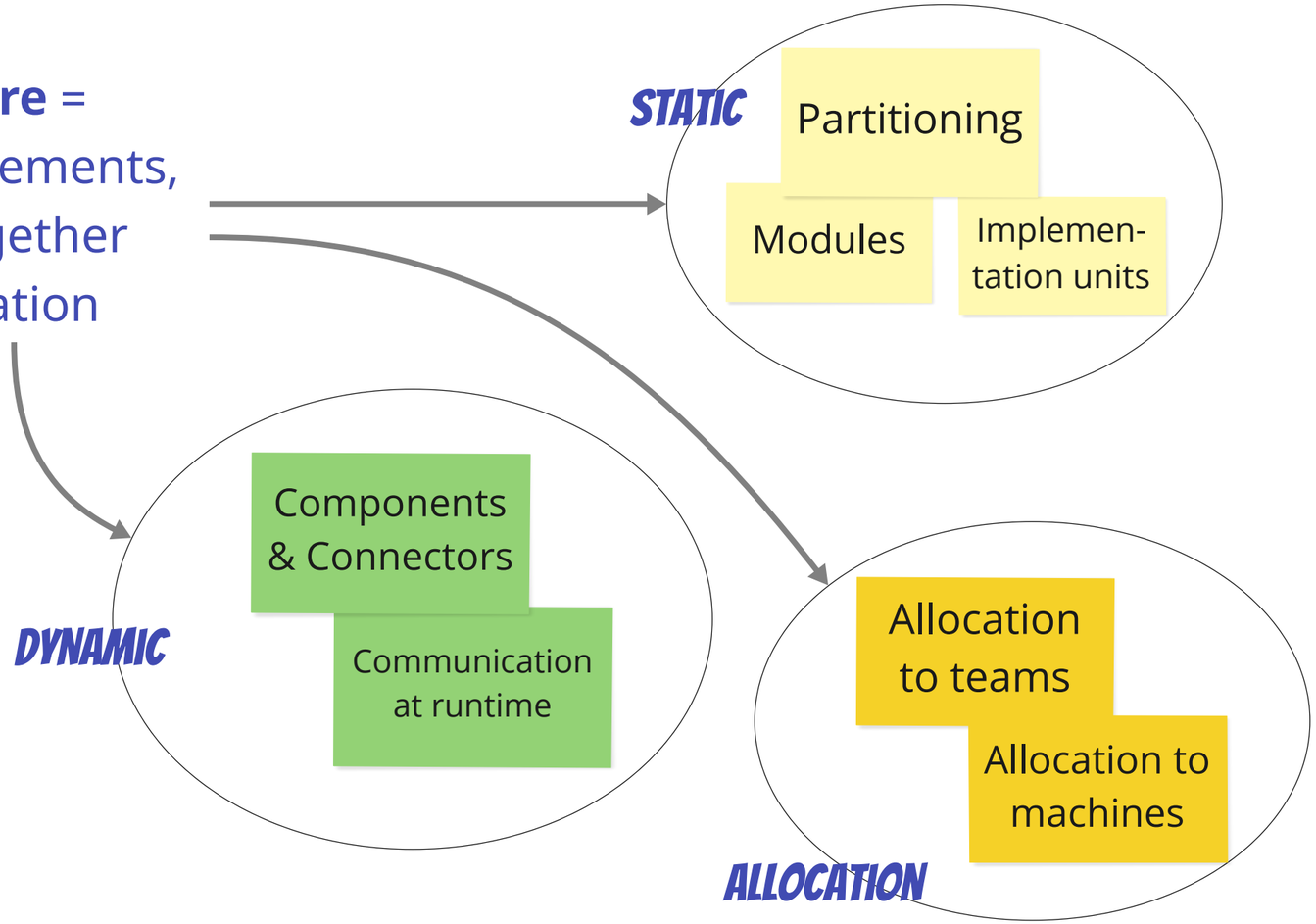
SOFTWARE ARCHITECTURE IS...

the set of **structures** needed
to **reason** about the system
which comprise software elements,
relations among them,
and **properties** of both.

Bass, Clements, Kazman:
Software architecture in practice,
3rd edition, page 4



Structure =
set of elements,
held together
by a relation



Reasoning
about an attribute of the
system that is important
to some stakeholder

Approach =
a way in which patterns
and tactics affect particular
quality attributes

WHAT PATTERNS AND TACTICS DO WE USE TO ACHIEVE QUALITIES?

portability, maintain-ability

Layers

at expense of performance

scalability, maintain-ability

Publish/sub-scribe

usability, modularity

MVC

availability

Active redundancy

SOME EXAMPLES:

Pipes and filters

reuse, loose coupling

Peer-to-Peer

availability, scalability

performance

Caching

at expense of maintain-ability

Client / Server

modifiability, ease of deployment

WHAT APPROACHES DO YOU USE IN YOUR ARCHITECTURE?

Put stickies here (use cmd-D or ctrl-D to duplicate)

Double-click on a word to edit it

Write it as "A for Q", i.e. "approach A to reach quality Q"

Zoom in or out with your mouse wheel (or with two fingers on a touchpad)

Improving Throughput

API authentication for Security

OAuth2 for Security

Horizontal scalability for performance

Code readability for maintainability

Reactive Programming for Performance

configuration Data for flexibility

CQRS for maintainability

Kafka for reliability

Modularity for Integration into platform

DDD for Maintainability

multithreading for performance

Monitoring & Alerting for availability

Layers for maintainibility

Dependency Injection for Testability

Logging and Monitoring for Mainainability

FaaS for scalability

API for decoupling

Layering for loose coupling/changeability

Autoscaler for Availability

Service interaction for reusability.

DB Partitioning for scalability and performance

Distributed Tracing for analysis/transparency

Automatic vulnerability scanning for security

Event streams for multiple views of same data

Cache for Performance

Spatial calculations within db for performance.

Duplicate these and change the name

Kubernetes for maintainability / extensibility (same structure)

Test coverage for stability

Versioning for changeability.

Message queue for reliability

Y-Principle for separation of concerns

AWS Lambda for Availability

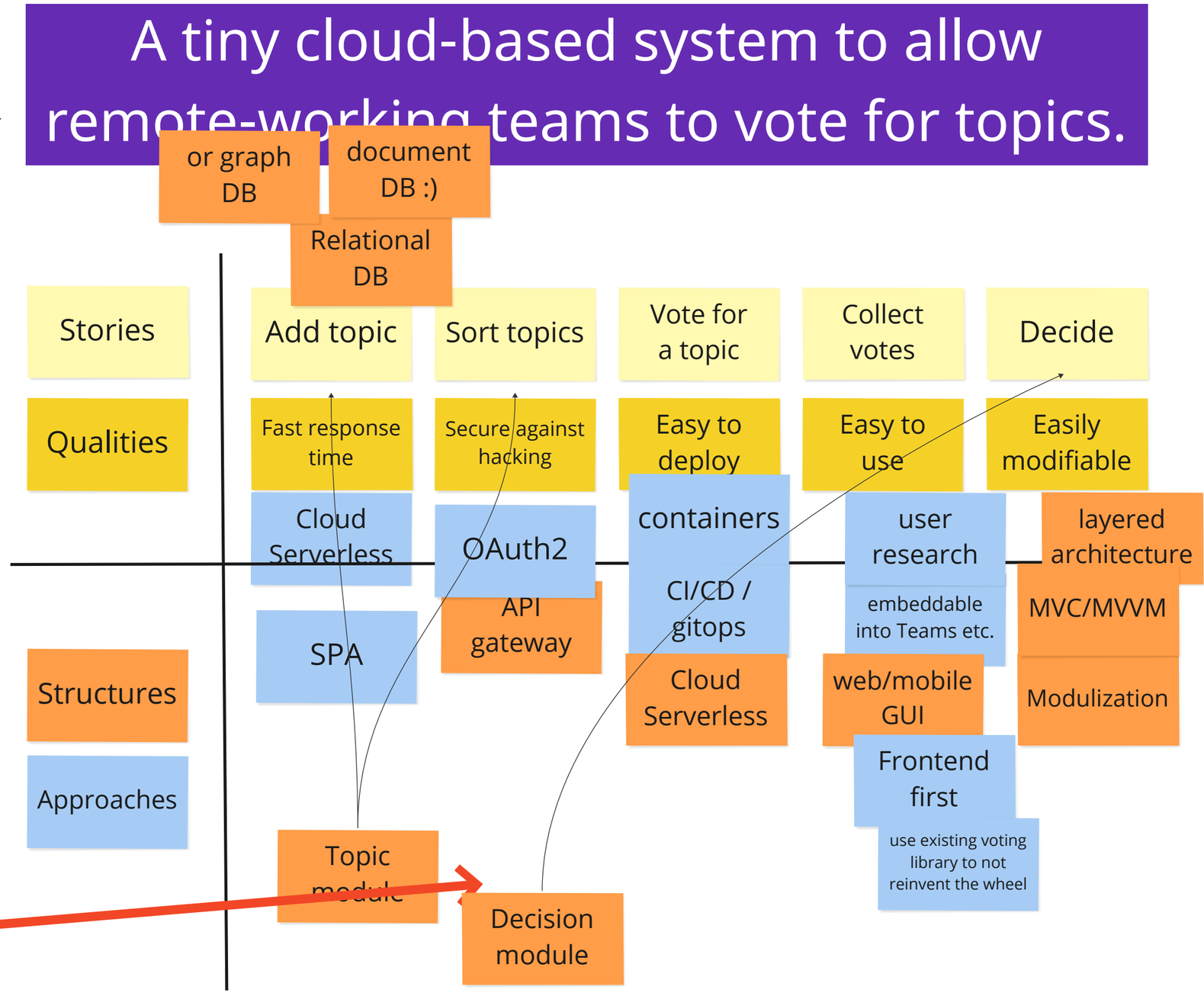
THE TINY VOTING SYSTEM EXAMPLE

Context

Goals and Qualities

Structures and approaches

Name the blue **approaches** and orange **structures**, and link them to the yellow **stories** and **qualities**:



Standardize your architecture vocabulary

Whenever these come in...

- Context
- Goal
- Quality

Those need to be created / modified:

Patterns, tactics, approaches

- Layers
- Publish/sub-scribe
- MVC
- Peer-to-Peer
- Caching
- Client / Server
- Pipes and filters

Static structures

- Partitioning
- Modules
- Implemen-tation units

Dynamic structures

- Components & Connectors
- Communication at runtime

Allocation structures

- Allocation to teams
- Allocation to machines

REPEATABLE DECISIONS (ARCHITECTURAL POLICIES)

"When we need another quality, we look for a known approach for that."

quality

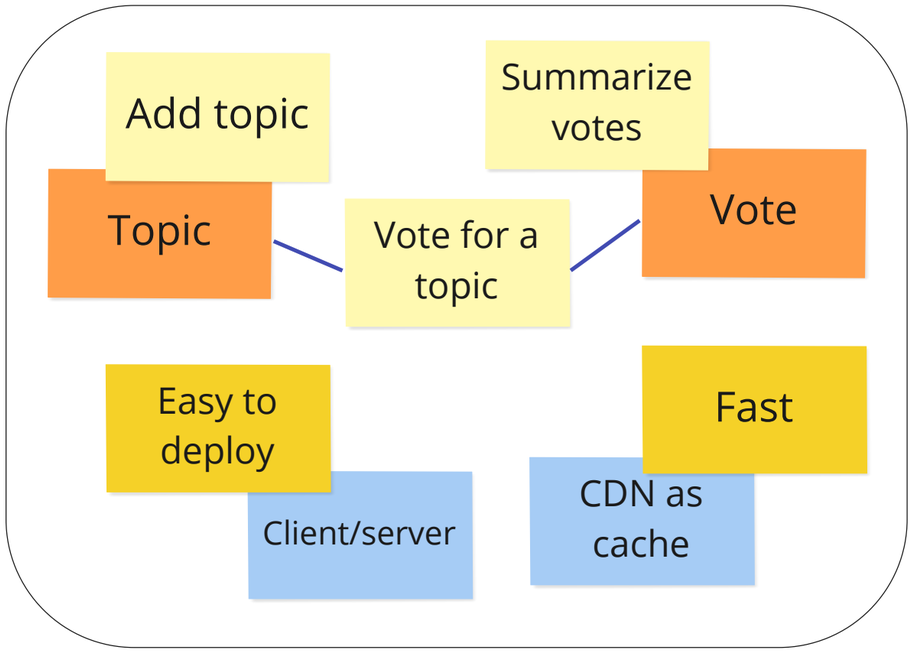
Attributes like

- performance
- security
- usability...
- (ISO 25010 has 5 more!)

Some stakeholder wants...

- a function
- a report
- another kind of "story"

goal



approach

pattern

tactic

To make stories run, we need to make

- a module
- a component
- a communication mechanism

structure

Ways to affect a quality attribute

- tactic = to-do
- pattern = known solution for a problem
- approach = larger concept that uses tactics and patterns

"When we get a goal (e.g. a story), we look for structure(s) that can support it."

PUTTING IT ALL TOGETHER

Incoming architectural forces

Context

Quality

Goal

in

Architectural reasoning

out

out

Don't forget to get feedback and iterate!

Approaches

portability, maintain-ability

Layers

at expense of performance

scalability, maintain-ability

Publish/sub-scribe

usability, modularity

MVC

availability

Active redundancy

Pipes and filters

reuse, loose coupling

Peer-to-Peer

availability, scalability

performance

Caching

at expense of maintain-ability

Client / Server

modifiability, ease of deployment

Structures

Partitioning

Modules

Implemen-tation units

Components & Connectors

Communication at runtime

Allocation to teams

Allocation to machines

TO-DOS FOR MONDAY

Run an "architectural vocabulary" workshop with your team

1

Show the TVS example

Brainstorm existing vocabulary

2

Collect context & goals

Collect qualities

3

Collect existing structures

Identify element types

Identify relation types

Structure = set of **elements**, held together by a **relation**

Collect, write, use, improve!

5

Name all the stuff you found

Put vocabulary and policies into your team's Wiki

Brainstorm existing tactics

Brainstorm existing patterns

4

Bundle them as approaches

IF YOU NEED HELP WITH THIS OR WITH DOMAIN-DRIVEN DESIGN...

Hire me: mbohlen@mbohlen.de