Project Networks and Percept-Plan-Execute Loop to the rescue

Septimal Mind Ltd.

team@7mind.io

December 1, 2019

Overview

- What is this all about?
  - The Problem
  - What's wrong with everything?
- Some interesting stuff
  - A bit of boring definitions
- PPER loop and The Ultimate Machine
  - The Pattern
  - The Ultimate Machine
- What else?

What are we doing?

- We are building systems
  - Complex ones
- We need to deliver our work in time and avoid fuckups
  - But we also want to have a life
- So we are aiming for productivity
- What makes our life hard?
  - Complexity.

Where complexity lies?

Everywhere in the SDLC:

- Domain analysis: we need to understant The Problem first
- Initial formalization: make it fairly formal
- Design: it's hard to make Our Things maintainable and viable
- Development
  - State
  - Coupling
  - Tests
  - Protocols, especially asynchronous
- Integration
- Delivery
- Maintenance

. . . and there are thousands of opinions on How To Do The Things

So what?

We are going to talk about Domain Anaylisys and understand how to do things better.

- Recently we've discussed Event Storming as a way to get some grip on the domain
- We have Mindmaps for the same purpose
- And we have Flowcharts as a "simplified" formal definition
- And don't forget about UML (does anyone like it as a *modeling* tool?)
- and there are many other approaches

but...

The Hell

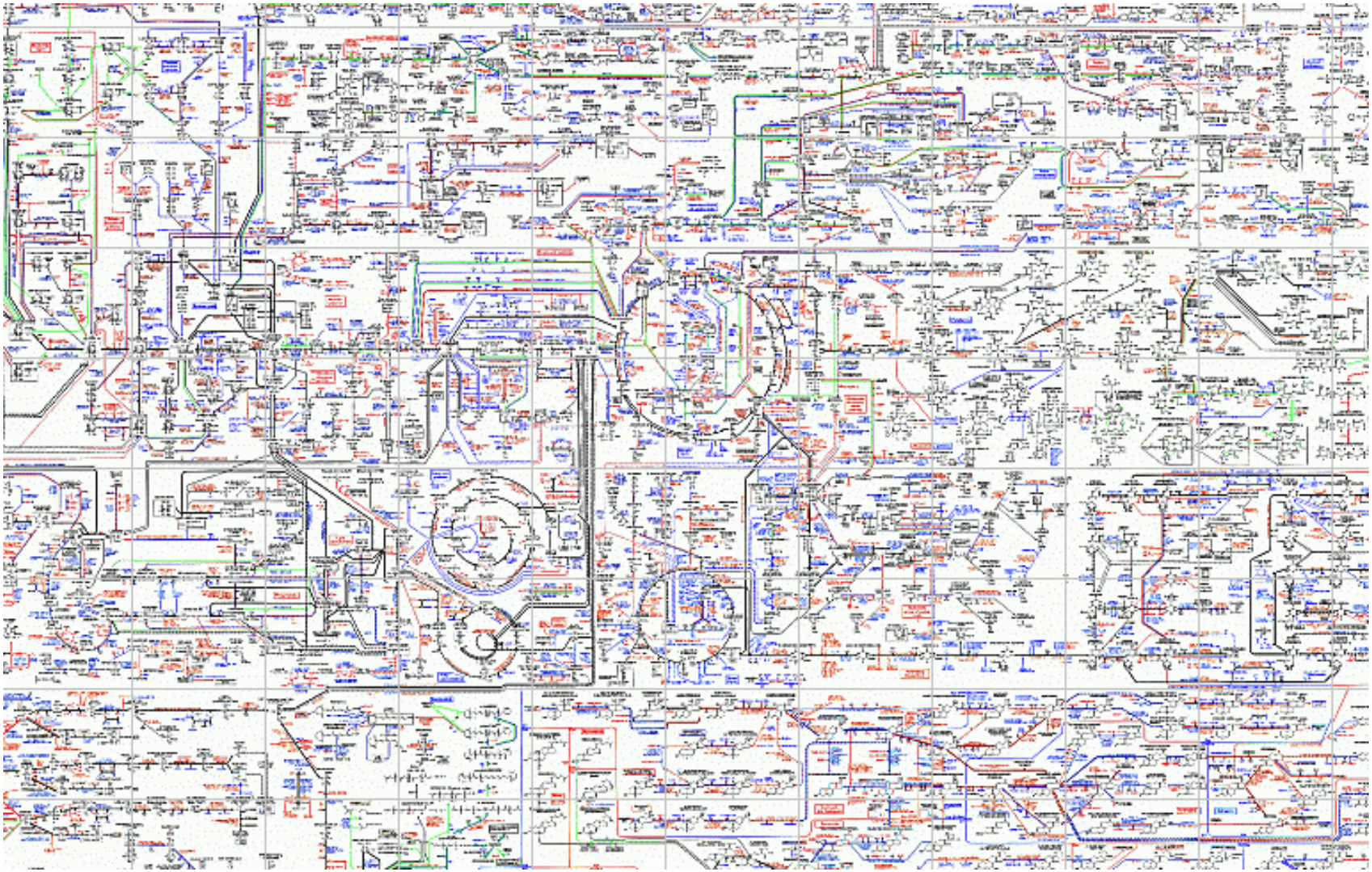
What to do in case our domain is *inconceivable*?
We may have:

- thousands of entities we need to represent
- thousands of events
- complex flows
- sequential and parallel subflows
- undecidability

Examples:

- American healthcare: enormous amounts of entities, complex protocols involving a lot of human interaction
- Cluster orchestration: fairly low amount of entities but protocols are deadly complex, too much uncertainty and undecidability

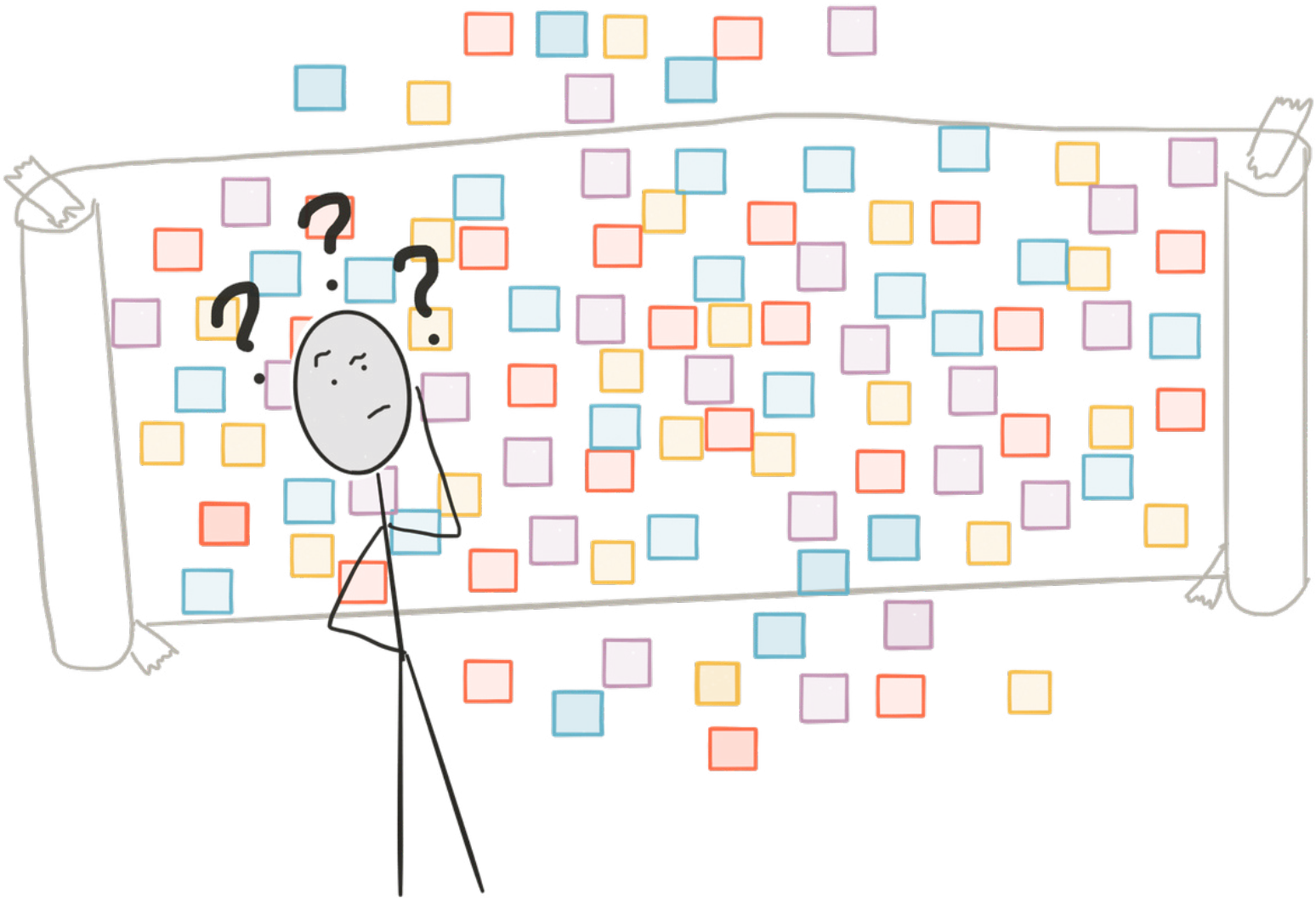
The Hell: flowcharts



The Hell: flowcharts

- They grow fast and it's easy to make them too complex to be useful
- Flowcharts are not hierarchical
- They allow us to create loops and conditions, so they are Turing-complete. Better write code.

The Hell: events



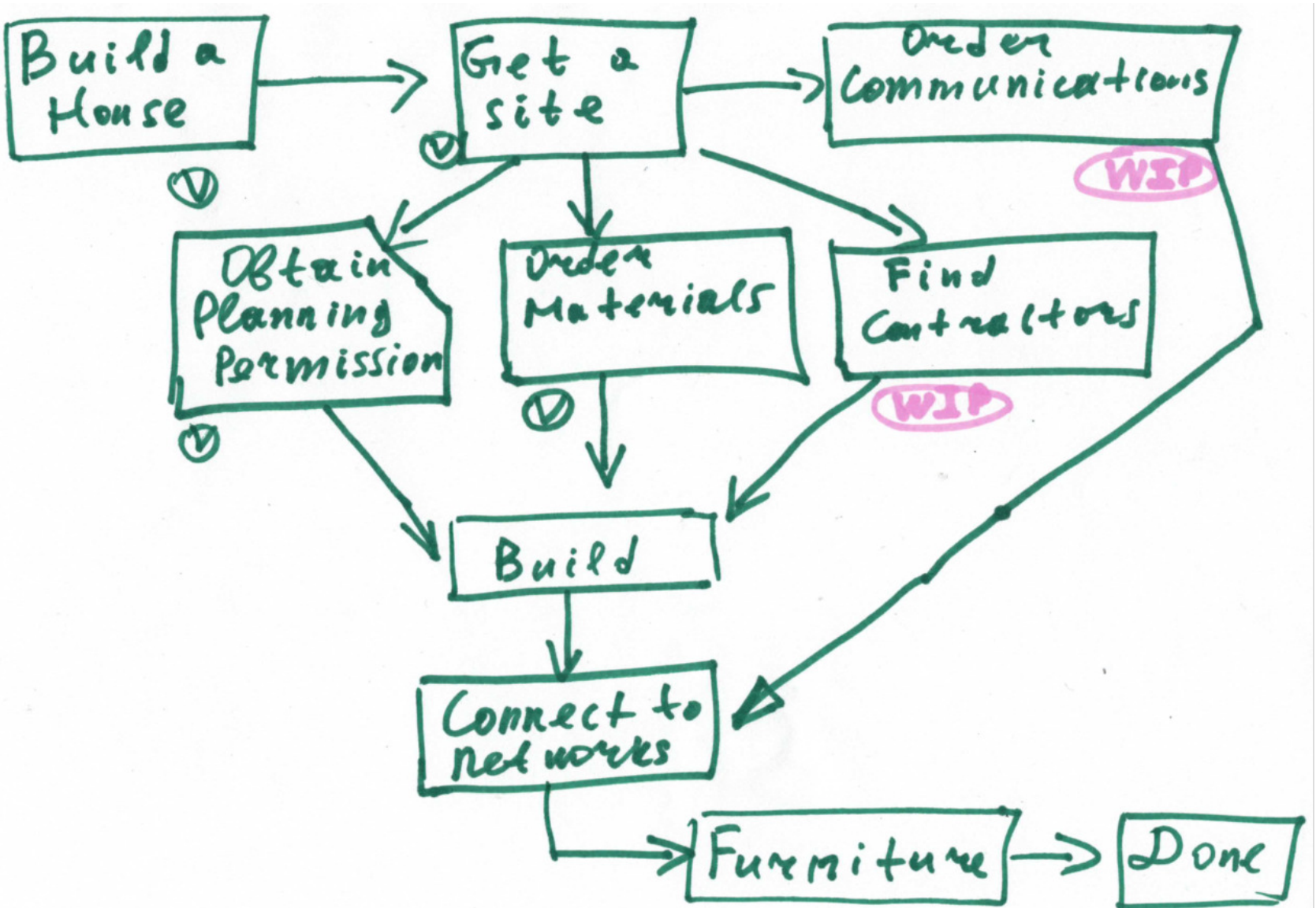
The Hell: events

- There is a duality of Entities and Events
- Event Storming board is flat
- These hundreds of sticky notes are hard to handle
- We still need Turing-complete formalisms

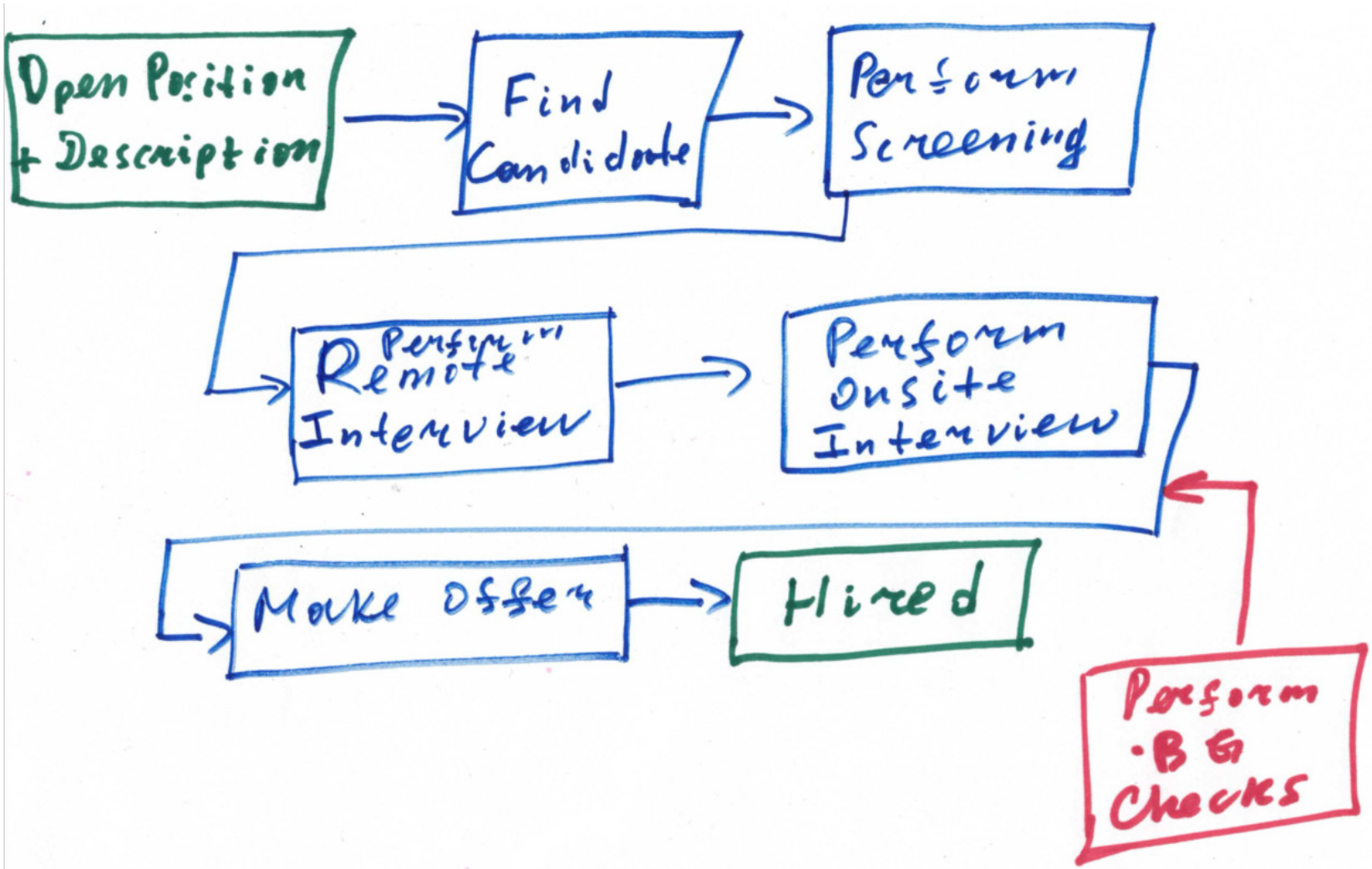
Project networks

- Usually it's not so hard to express the Happy Path.
- It's hard to handle cornercases, branches, etc
- *Project network* is a DAG where nodes are actions to do and edges are dependencies
  - Natural way to express parallelism

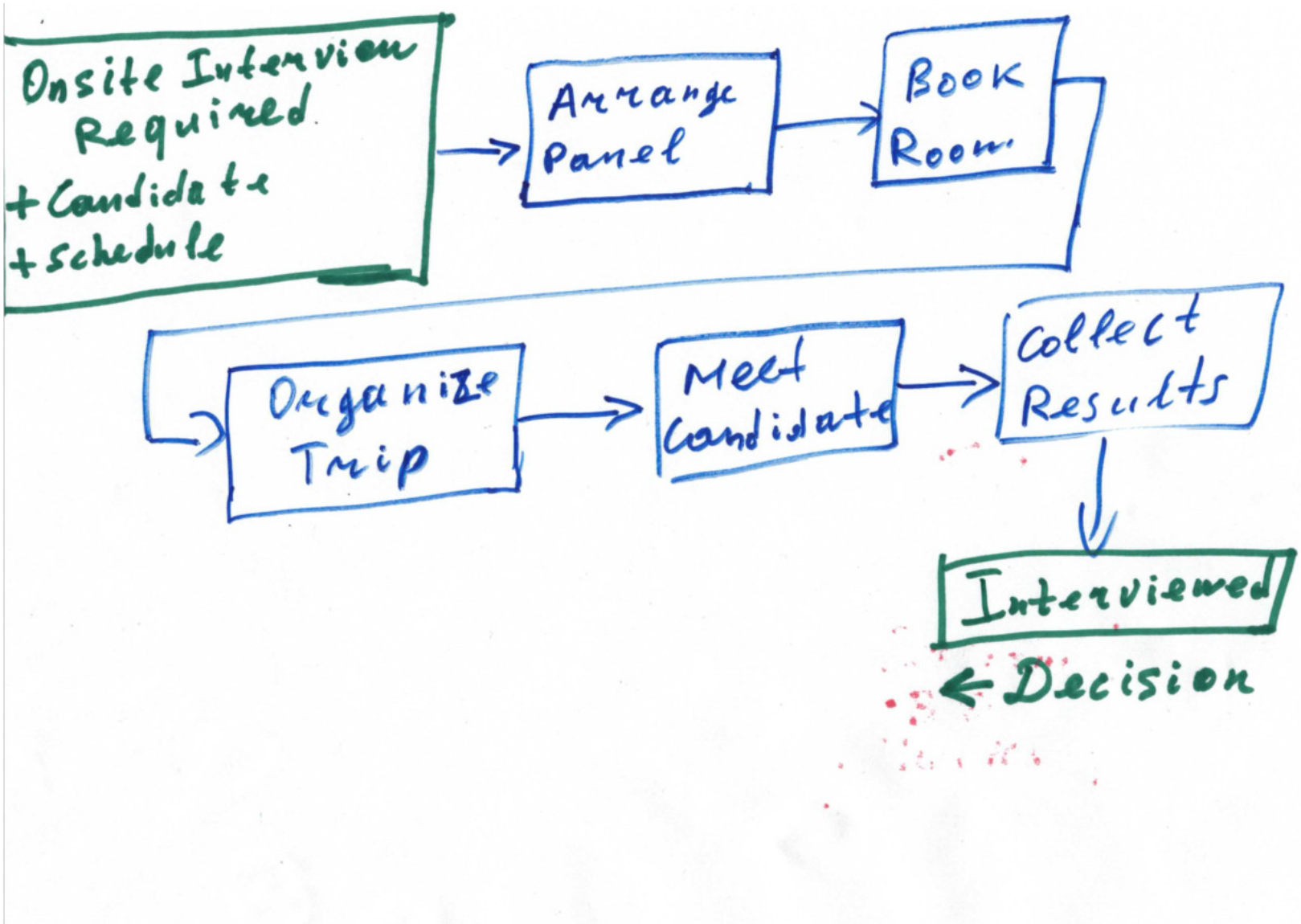
Example: build a house



Example: hiring process



Example: hiring process step



Project networks: how to use

- Let's represent our flows as Project Networks
- Nodes would stand for actions to do, *verbs*
- Let's call a Node *trivial* in case it represents an atomic action
- Let's call a Node *complex* in case it represents a complex action

Important to remember:

- A Complex Node can be represented as another, *nested* Project Network
- We can build a hierarchy of abstractions
- Each abstraction level is described by Verbs and Flows composable out of them
- Verbs available at a level would form a DSL
- Each subordered level should have higher detailization and locality

Case Management

A boring definition:

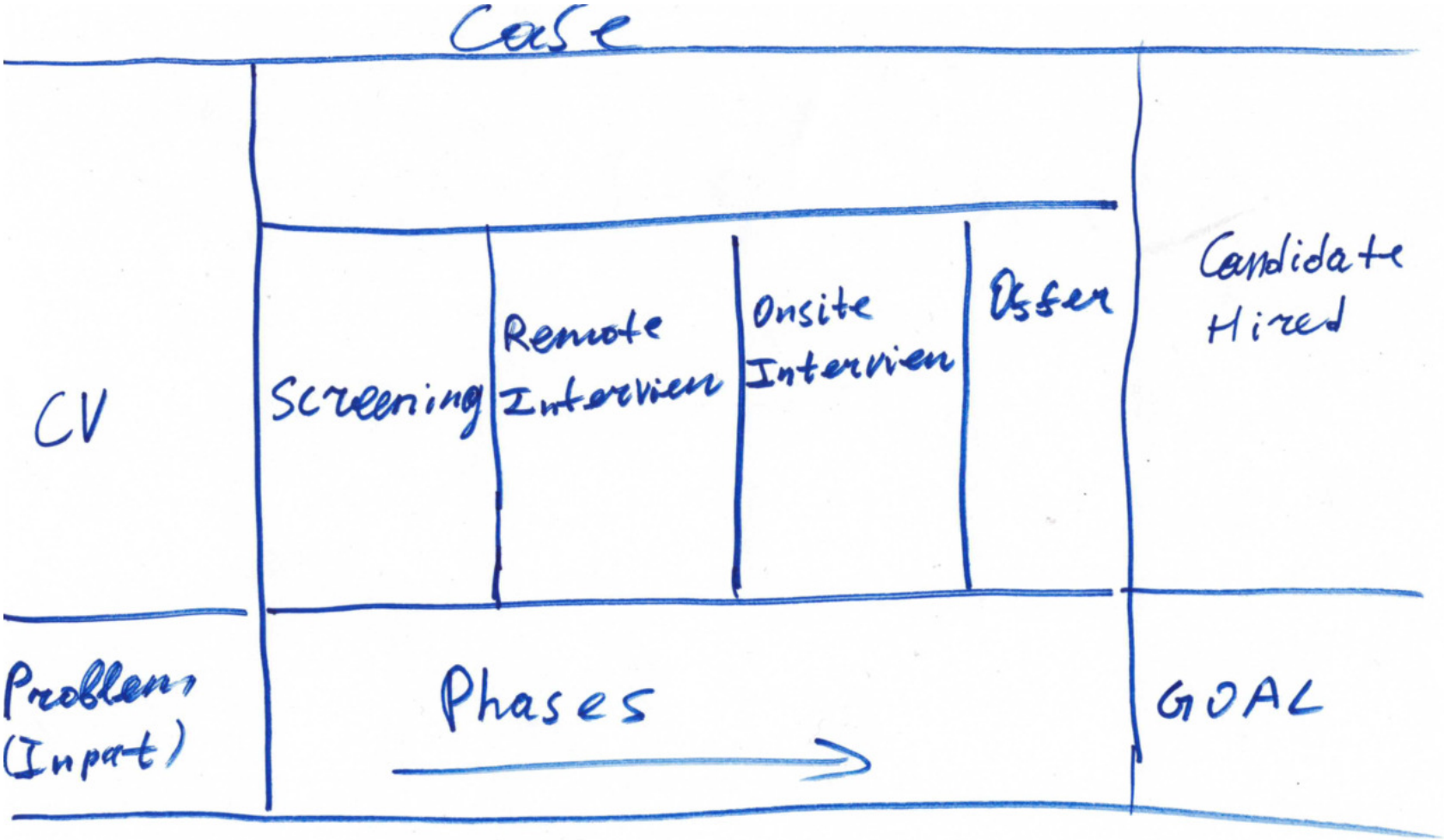
- "*A* Case is a collection of information and coordinated activities by knowledge workers or case workers"
- "Represents an entity that the organization must process and it is sometimes identified by having a subject"

Case Management

Common organizational pattern:

- We have an abstract context named *Case*
- A case consisting of *Facts* — abstract datums
- Initially a case consists of an *Problem* (input) and a *Goal*
- We have a nondeterministic sequence of *Phases* which should allow us to achieve the Goal
- Each Phase may add some new Facts into the case
- Each Phase can be executed by an idependent actor
- Case Lifecycle is governed by coordinating actor named *Case Manager*
- The context is considered alive by the Case Manager unless the Goal is reached

Case Management



Viable System Model

- An organizational model of an animal neural system
- Introduced by Stafford Beer
- Proposes a way to implement *homeostasis* in an abstract system
- Intended to deal with complexity and uncertainty

Key points:

- Algedonic regulation
- Hierarchy of interacting subsystems
- Levels are local autonomic
- Metalanguage stack
- Variety increases top-down and decreases bottom-up: we are operating a limited set of high-level abstractions on top level and many low-level abstractions at the bottom level

How not to get crazy on domain analysis

- Let's represent our domain as a set of Cases
  - Works well for many, many real processes
  - Not for every though. Not a silver bullet.
- Let's draw Project Networks for each case
- Don't try to cover all the cornercases, concentrate on primary flows
- Try to keep vocabulary tiny
- In case you need a branch try to create a sub-network

How to make it formal enough to use?

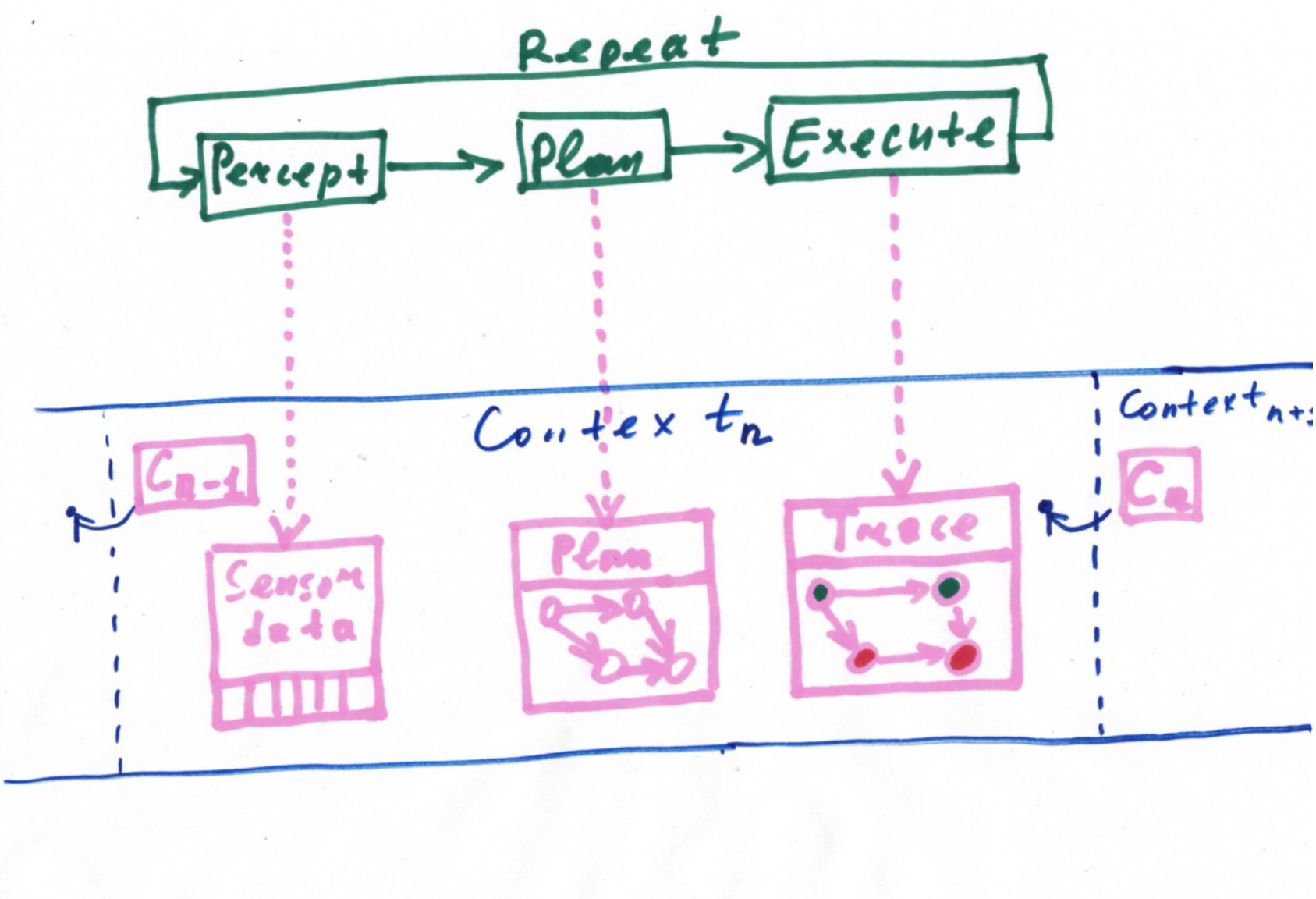
- Okay, this may be useful for getting grip on our domain,
- But now to turn it into the code?

PPER Loop

A very generic and a very important pattern:

1. Acquire data from the outer world (*Percept*)
2. Produce a Project Network, *Plan*. It may be incomplete, it should allow us to progress (*Plan*)
   - Plan is a DAG, remember?
3. Execute the Plan (*Execute*).
   - Perform the steps of the Plan
   - Mark your Plan nodes according to the results of their execution
   - Let's call marked plan as *Trace*
4. Go to step 1 unless termination criteria reached (*Repeat*)

PPER Loop


Repeat
Percept
Plan
Execute
Context t_n
Context $_{n+1}$
C_{n-1}
C_n
Sensor data
Plan
Trace

Parts for The Ultimate Machine

1. *Sensor* or *Afferent Channel* is a data source. Allows us to get some state from the outer world.
2. *Planner* an abstract entity taking Sensor Data, Previous Plan and Previous Trace and producing new Plan
3. *Executor* or *Efferent Channel* is an entity able to change outer world and get a result

Let's put things together

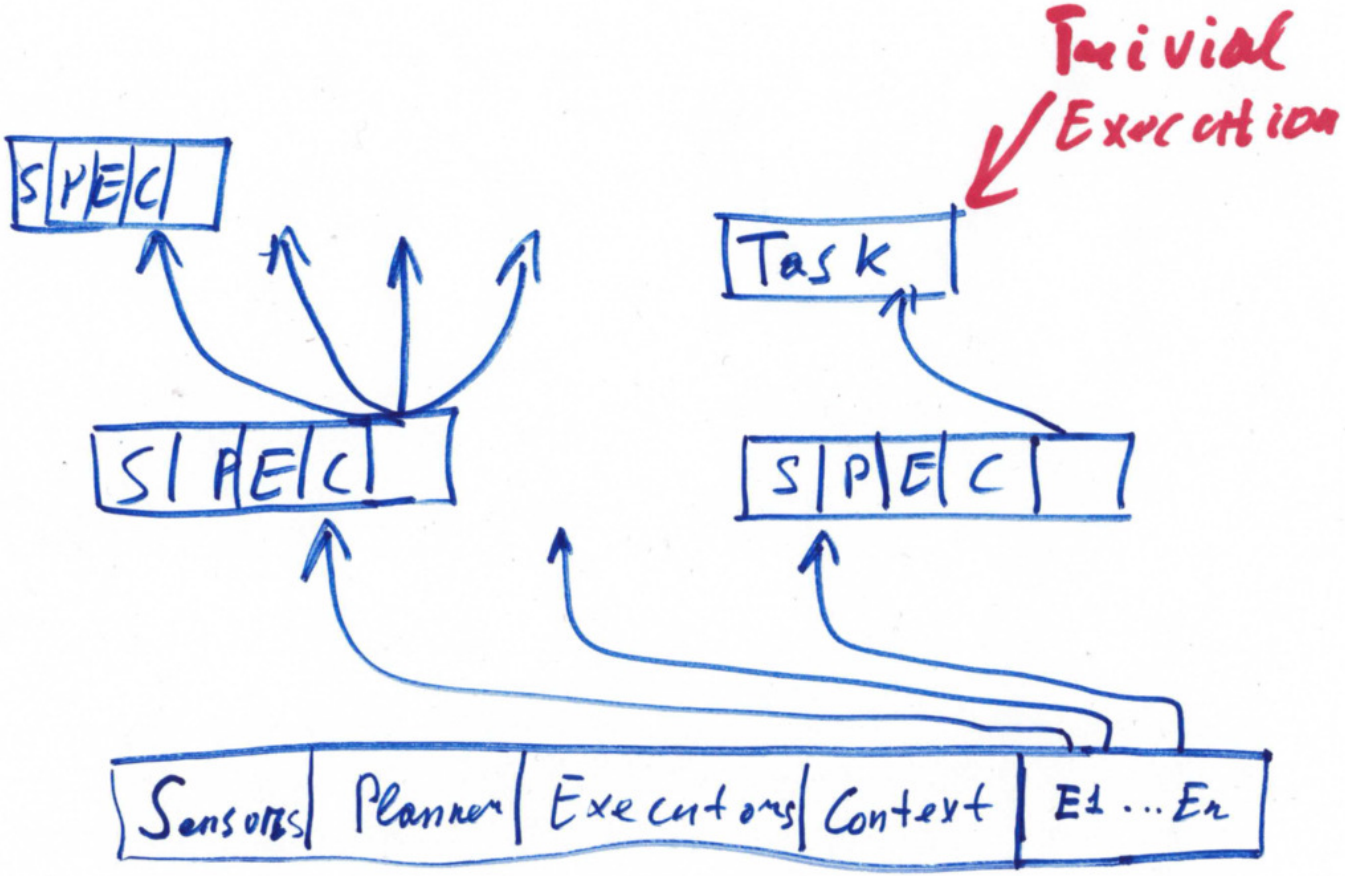
We may imagine an abstract machine supporting PPER loop:

1. A Specification is a set of $(name, type)$ pairs.
   $S = [(name, type)]$
2. A Verb is a name plus input specification plus output specification. $v = \{name, In_v, Out_v\}$
3. A language is: set of verbs, bindings for executors for each verb, planner, sensors, goal specification, problem specification. $L = \{planner_L, E_L, S_L, V_L, g_L, p_L\}$
4. So we have a hierarchy of abstract incomplete languages, *Metalanguage* $ML = [L_1, \ldots L_n]$

Let's put things together

1. Let's isolate each PPER loop in a separate stackframe
2. Each stackframe contains Context, Sensors, Planner, Executors, current Plan, current Trace, previous Trace, previous Plan and array of subordered frames
3. In fact it's not a stack, it's a tree
4. When an Executor starts working on a non-trivial Step we should create a new stackframe

The Ultimate Machine



What do we have now?

- All the entities are stateless, the state is held in the machine's stack.
- Turing Complete but *pure* Planners.
- Turing Incomplete Plans are easy to analyse.
- **Effect Separation** as a part of our design.
- Arbitrary planning strategies
- Testability: Planning is repeatable so easy to test.
- Perfect simulations.
- Introspectability: it's easy to visualize even very complex processes.
- Traceability: we may record all the history stack states.
- Portable State: machine state is easy to serialize, restore, explore offline.
- Parallelism

Let's go further

- Planning may be **delegated to a human**!
- So may Execution
- We may add a UI with a task queue (like Amazon Mechanical Turk)
  - Perfect Business Process Management solution
  - Perfect HRMS
  - Gamification
  - Onboarding
  - Plans, produced by a human, may be reused
  - ML, Data Mining...

Applications

Many problems related to Operations Research and requiring dynamic planning

- Homeostasis problems: orchestration, robotics, adaptive/viable systems...
- Project Planning and dynamic business flows: HRM, CRM, Project Management
- Build tools, DI frameworks...
  - I have implemented a DI framework, exploiting the PPER principle :)

How to plan?

- An endless subject.
- Easier than usual because of the separation
- Any approach you wish: ML, Constraint solving, Genetic Algorithms...
- An interesting idea: to make planners typesafe and prove safety in compile time
- Prolog and other constraint solvers would work great

More interesting stuff

The author would be happy to make more presentations regarding some projects he is working on right now:

- Staged Generative DI framework. (Best one, you wouldn't need Guice anymore)
- Data modeling and interface definition language. (Forget about GRPC/Protobuf)
- Structural logging (and how to make it free)

Thank you for your attention

We're looking for clients, contributors, adopters and colleagues ;)

About the author:

- coding for 18 years, 10 years of hands-on commercial engineering experience,
- has been leading a cluster orchestration team in Yandex, "the Russian Google",
- implemented "*Interstellar Spaceship*" – an orchestration solution to manage 50K+ physical machines across 6 datacenters,
- Owns an Irish R&D company, https://7mind.io,
- Contacts: team@7mind.io,
- Github: https://github.com/pshirshov