The EPI Framework

Tim Müller (CCI Group, UvA)

t.muller@uva.nl

UMC Utrecht SIG - 23 October 2023

Before we begin

- I’m a computer scientist
- But I’ll try to keep it nice and concrete :)
- Except when you see ⚙
  - Then it *does* get technical

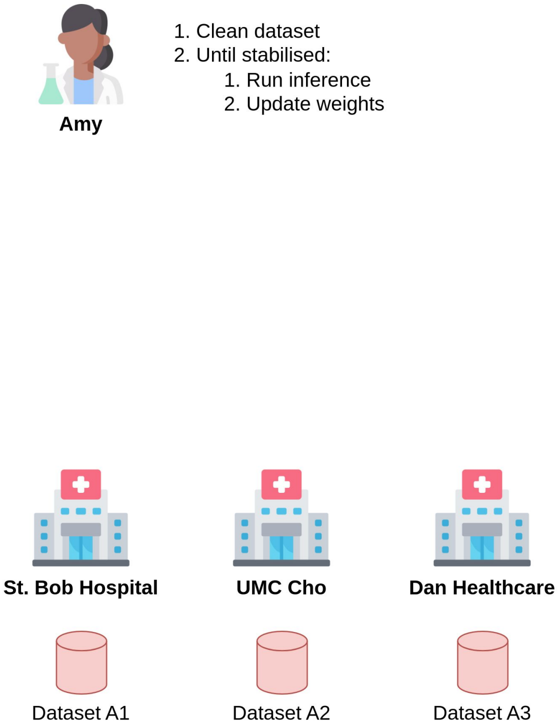
- Also note, the **EPI Framework** used to be called **Brane**, so I’ll use them interchangeably!

I. Introduction

Background

- There is a **distributed dataset**
  - Spread out over multiple hospitals (domains)
- Amy is a **data scientist** who wants to analyse it
- However: data is **super-sensitive**!

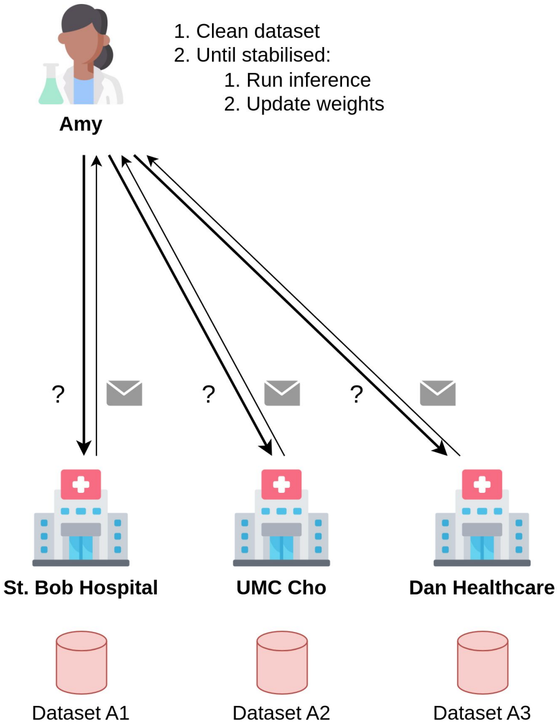
- **How can Amy safely analyse the data?**



```
1. Clean dataset
2. Until stabilised:
        1. Run inference
        2. Update weights
```

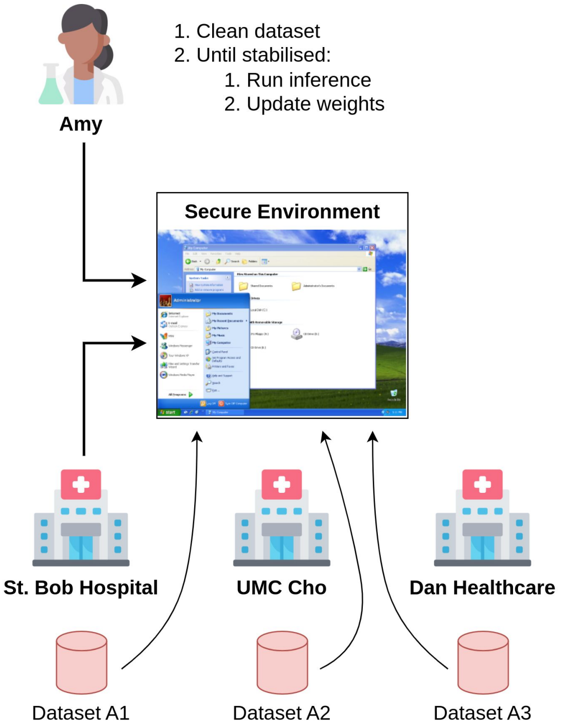
Naive approach

- Amy might **request all domains for access to data**
- Domains then **send back the data in full**
- She performs her steps **locally**
- Evaluation
  - Requires a lot of trust in Amy!
    - No oversight to what she does
  - Very tedious to arrange
    - Manual decision making
    - Manual preprocessing
  - Might be hard to share data securely



Better approach

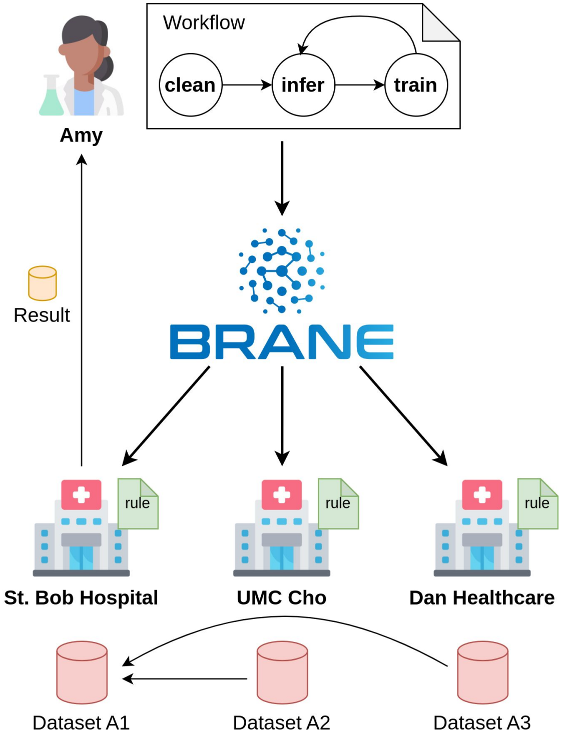
- One hospital creates **secure environment**
  - Hospitals make **data available** in this environment
- Amy can perform her work **“on-site”**
- Evaluation
  - Still requires trust in Amy
    - Better, but still limited oversight
  - Still tedious to arrange
    - Manual decision making still occurs
    - Per-researcher setup
  - ~~Might be hard to share data securely~~



1. Clean dataset
2. Until stabilised:
    1. Run inference
    2. Update weights

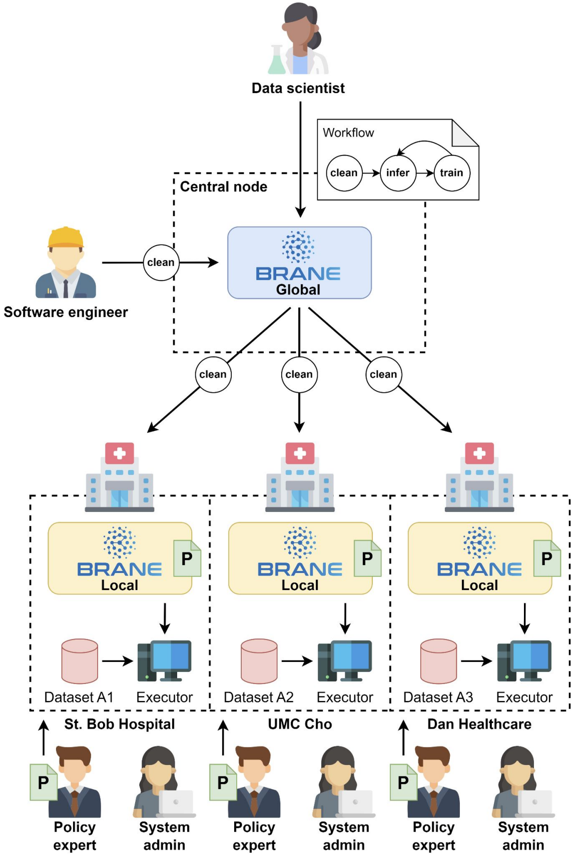
EPI approach

- Amy can **formalise her steps** (workflow)
- Domains can **formalise their regulations** (policy)
- EPI Framework can **perform the computation**
    - While ensuring **regulations are not violated**
- Evaluation
    - Requires minimal trust in Amy
        - Her plan can be analysed before computation
    - Less tedious to arrange
        - Manual decision making only required *sometimes*
        - Policies pre-defined (easily applied)



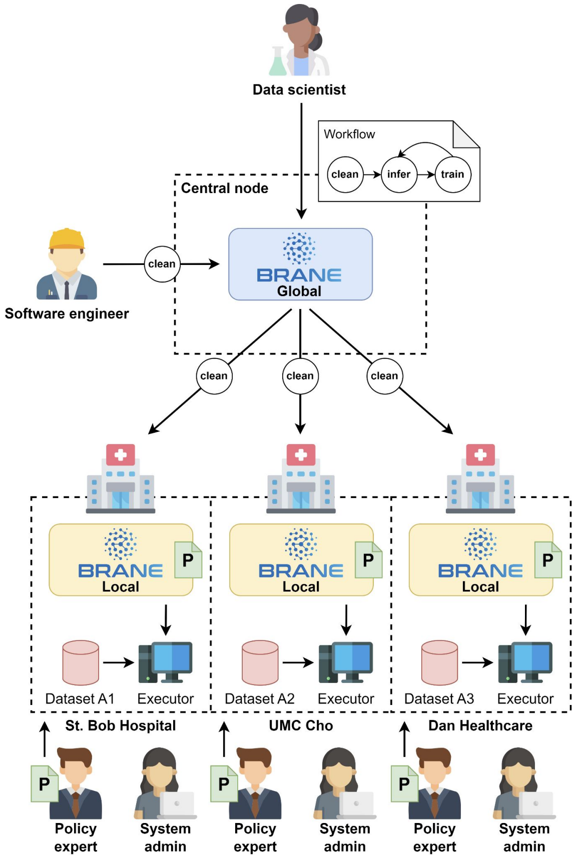
The EPI Framework

- Built to **share data** for **workflow**, **policy**-compliant
- The framework consists of **two components**:
  - The **central component** has the overview and accepts incoming work
  - The **local components** (owned by hospitals) has the data and performs the work



Separation of concerns

- Two **central-side** roles
  - Data **scientists**
  - Software **engineer**
- Two **local-side** roles
  - System **administrator**
    - Manages the components
  - Policy **expert**



II. Workflows

Data scientists & Software engineers

Recipes as workflows

- **Recipes** (as in cooking) are perfect examples of **workflows**
- Defines a **series of steps/tasks** to perform
- Mentions only **relevant details**
  - e.g., it doesn’t say which chef executes a task
- Tasks may be **dependent** on each other

Pasta Broccolo

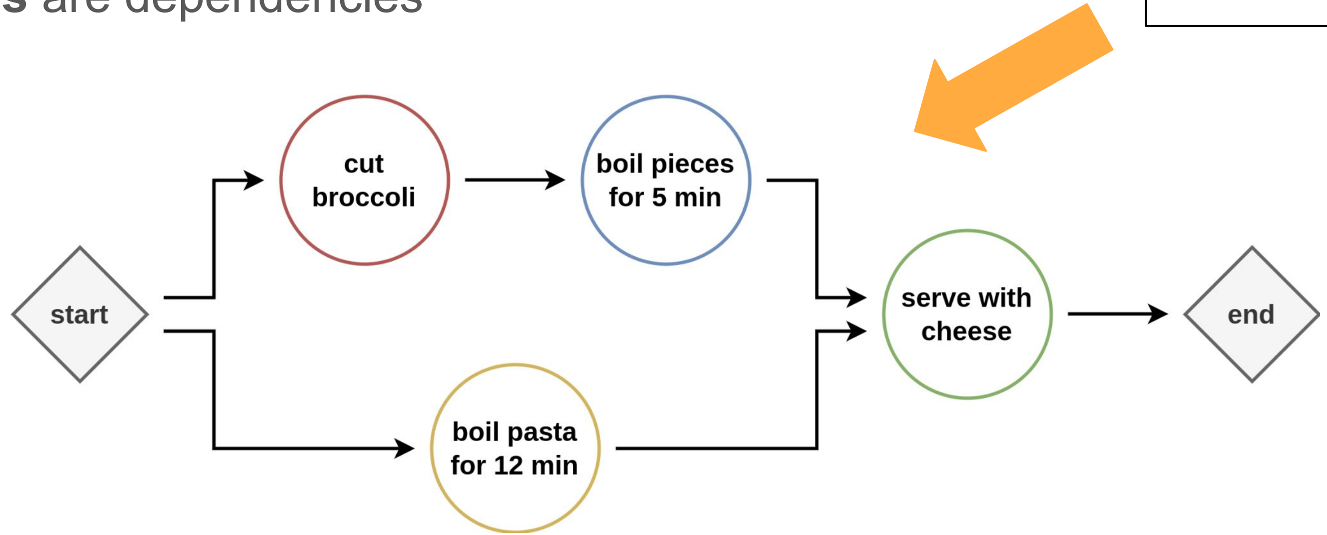
1. **Clean** the broccoli and cut it
2. **Boil** the broccoli pieces for 5 minutes, and the pasta for 12 minutes
3. **Serve** with cheese

Formalising recipes

- Recipes (workflows) can be represented as **graphs**
- **Nodes** are tasks
- **Edges** are dependencies

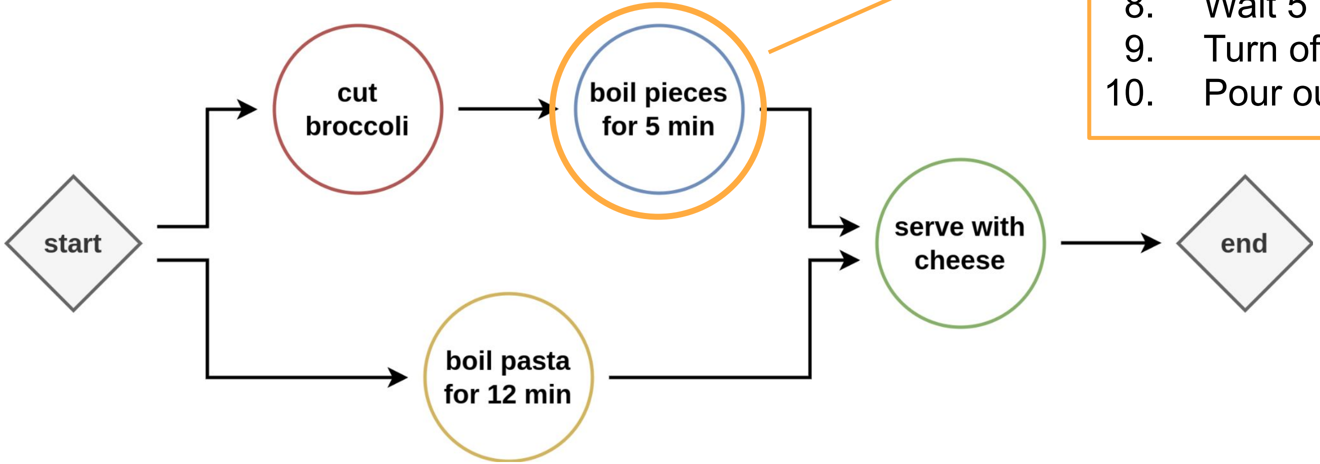
Pasta Broccolo

1. **Clean** the broccoli and cut it
2. **Boil** the broccoli pieces for 5 minutes, and the pasta for 12 minutes
3. **Serve** with cheese



Formalising recipes

- Recipes (workflows) can be represented as **graphs**
- **Nodes** are tasks
- **Edges** are dependencies

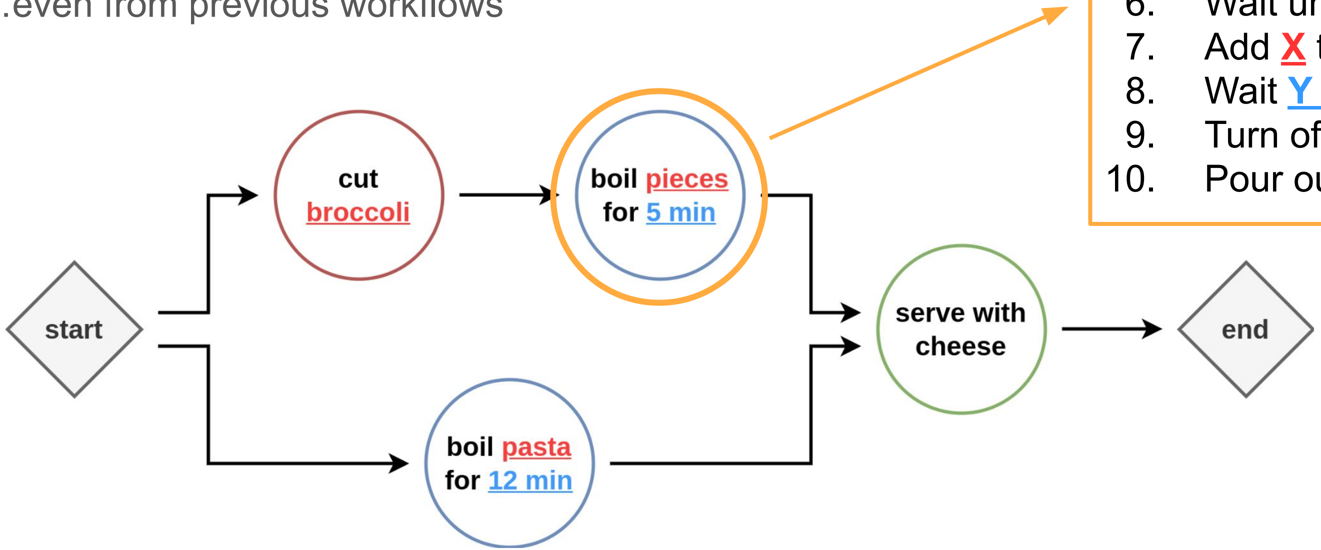


Boiling broccoli pieces

1. Get pan
2. Pour water in pan
3. Put pan on furnace
4. Put on lid
5. Light fire
6. Wait until bubbles
7. Add broccoli to pan
8. Wait 5 minutes
9. Turn off fire
10. Pour out water

Generalising tasks

- However, it’s nice if some tasks can be **parameterized**
- This means we can **re-use tasks**!
  - …even from previous workflows

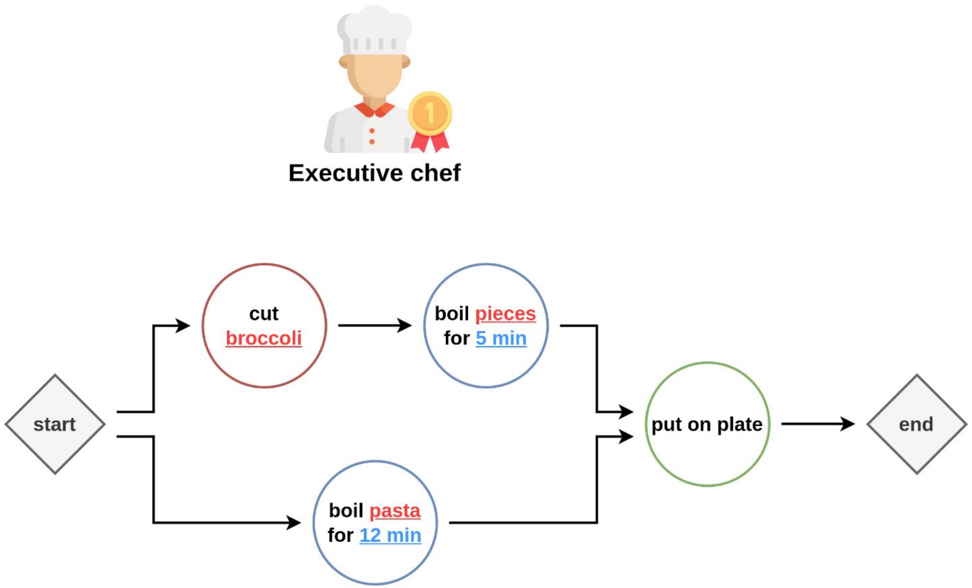


Boiling X

1. Get pan
2. Pour water in pan
3. Put pan on furnace
4. Put on lid
5. Light fire
6. Wait until bubbles
7. Add X to pan
8. Wait Y minutes
9. Turn off fire
10. Pour out water

Separation of concerns

- If we can re-use tasks, we can now write them **beforehand**
- Which means now **specialized people** can do **different things** in **parallel**!



Executive chef

start

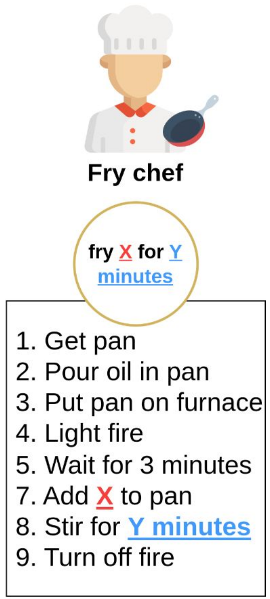
cut broccoli

boil pieces for 5 min

boil pasta for 12 min

put on plate

end



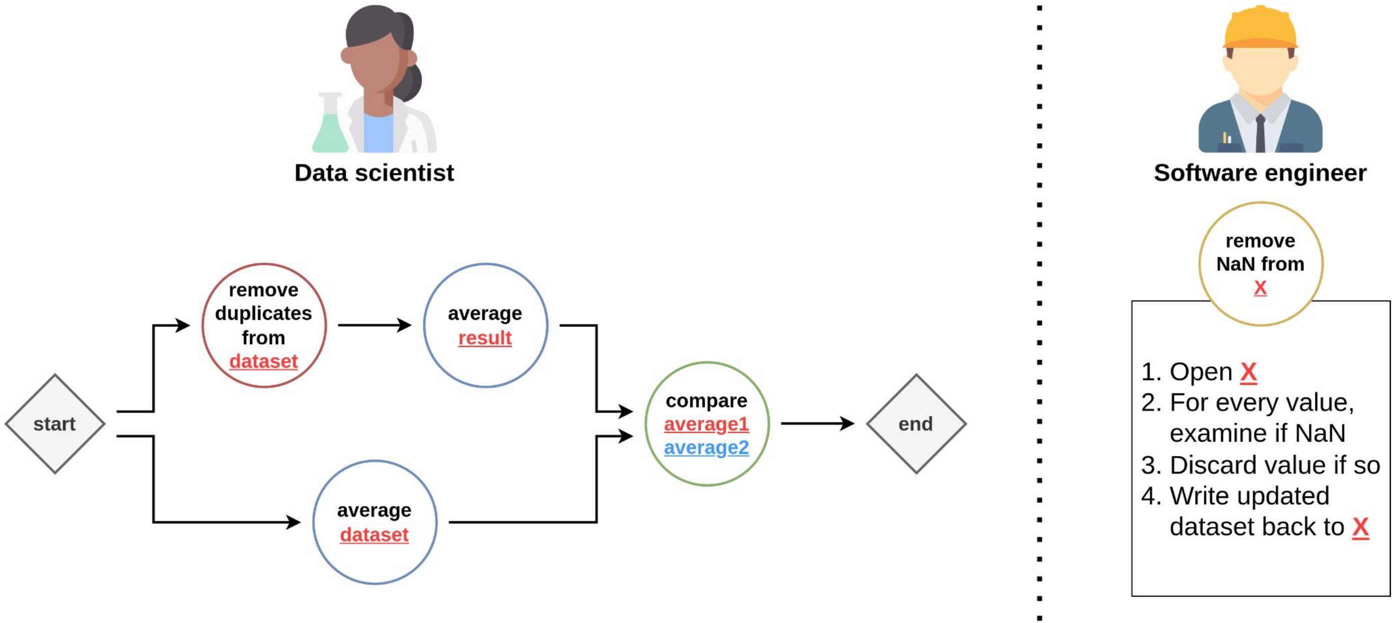
Fry chef

fry X for Y minutes

1. Get pan
2. Pour oil in pan
3. Put pan on furnace
4. Light fire
5. Wait for 3 minutes
7. Add X to pan
8. Stir for Y minutes
9. Turn off fire

Formalising programs

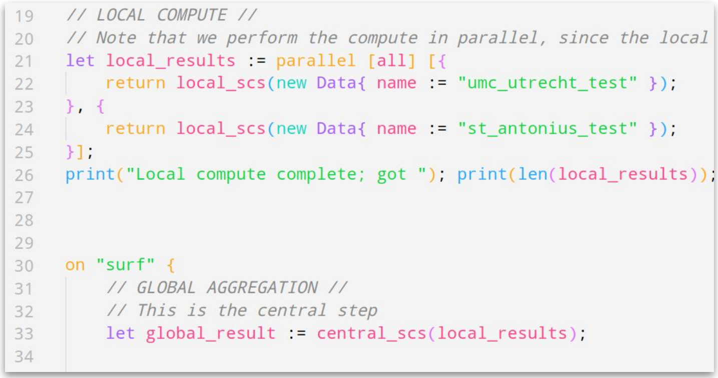
- We can **formalize programs** in exactly the same way
  - Also only describe high-level details
- Workflows written by **scientists**, tasks written by **engineers**



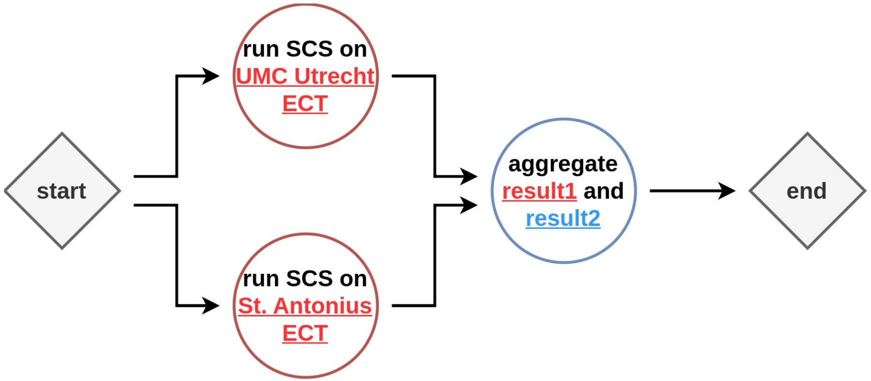
1. Open X
2. For every value, examine if NaN
3. Discard value if so
4. Write updated dataset back to X

Formalising workflows, practically

- The EPI Framework uses **BraneScript**
- Tasks are represented as **functions**, dependencies are **derived**
- Other methods supported in the future

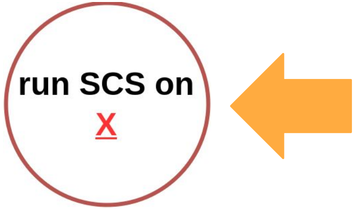


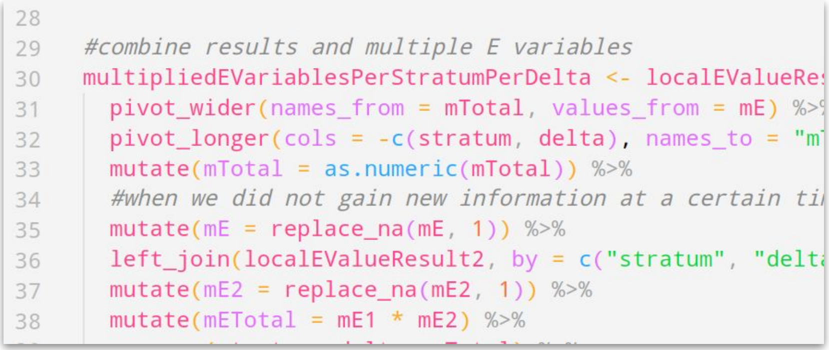
```
// LOCAL COMPUTE //
// Note that we perform the compute in parallel, since the local
let local_results := parallel [all] [{
    return local_scs(new Data{ name := "umc_utrecht_test" });
}, {
    return local_scs(new Data{ name := "st_antonius_test" });
}];
print("Local compute complete; got "); print(len(local_results));



on "surf" {
    // GLOBAL AGGREGATION //
    // This is the central step
    let global_result := central_scs(local_results);

```

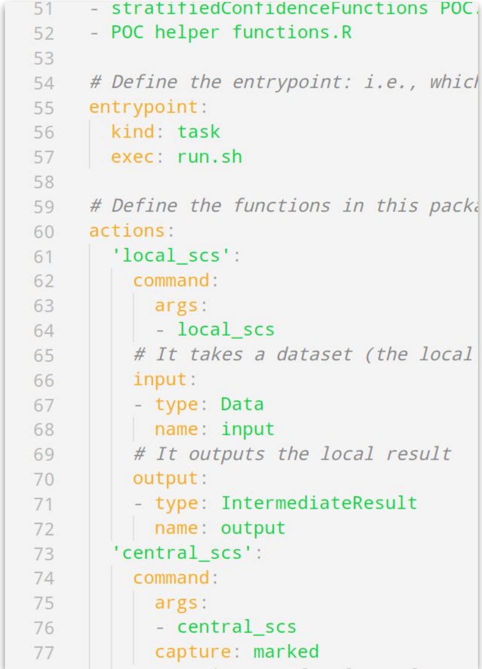

start
run SCS on UMC Utrecht ECT
run SCS on St. Antonius ECT
aggregate result1 and result2
end

Formalising tasks, practically

- Tasks are represented as **functions**
- Functions are grouped in **packages**
- Packages are implemented as **Docker containers**
  - Inputs read from environment variables
  - Output written to stdout
- Run **isolated** (input and output dictated by Brane)

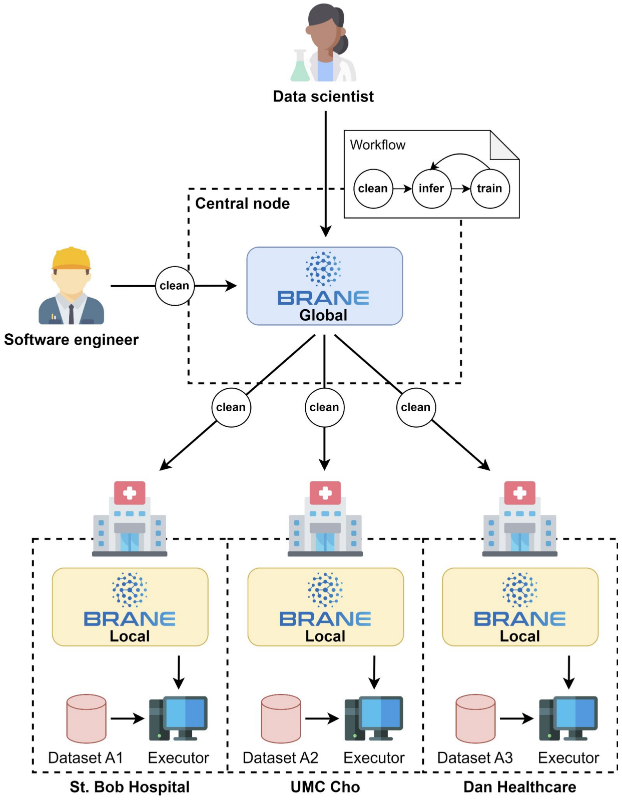

run SCS on
X


```

#combine results and multiple E variables
multipliedEVariablesPerStratumPerDelta <- localEValueRes
  pivot_wider(names_from = mTotal, values_from = mE) %>%
  pivot_longer(cols = -c(stratum, delta), names_to = "m
  mutate(mTotal = as.numeric(mTotal)) %>%
  #when we did not gain new information at a certain ti
  mutate(mE = replace_na(mE, 1)) %>%
  left_join(localEValueResult2, by = c("stratum", "delta
  mutate(mE2 = replace_na(mE2, 1)) %>%
  mutate(mETotal = mE1 * mE2) %>%
```


```
  - stratifiedConfidenceFunctions POC.
  - POC helper functions.R

# Define the entrypoint: i.e., which
entrypoint:
  kind: task
  exec: run.sh

# Define the functions in this packa
actions:
  'local_scs':
    command:
      args:
      - local_scs
    # It takes a dataset (the local
    input:
    - type: Data
      name: input
    # It outputs the local result
    output:
    - type: IntermediateResult
      name: output
  'central_scs':
    command:
      args:
      - central_scs
      capture: marked
```

Takeaways

- Workflows **formalise** analyses of data scientists
  - Encoded as a series of **tasks** with **dependencies**
  - Represented as a **graph**
- Doing so, they are **high-level programs**
  - Only details relevant for the scientist are expressed
  - Others are inferred by the framework's "expertise"



III. Policies

Policy experts

Back in the kitchen…

- Policies are **constraints** on what *should* happen
  - Directly, they prohibit some things happening in the workflow
- Can be from **various sources**
  - Laws (GDPR), organisational policies, contracts, etc
- Can be on **different levels**
  - Directly prohibit actions, impose conditions, …

Kitchen rulez

1. Your workspace must be hygienic
2. You must wash your hands before touching food
3. Listen to your boss
4. Thou shalt not put pineapple on pizza

The problem with policies

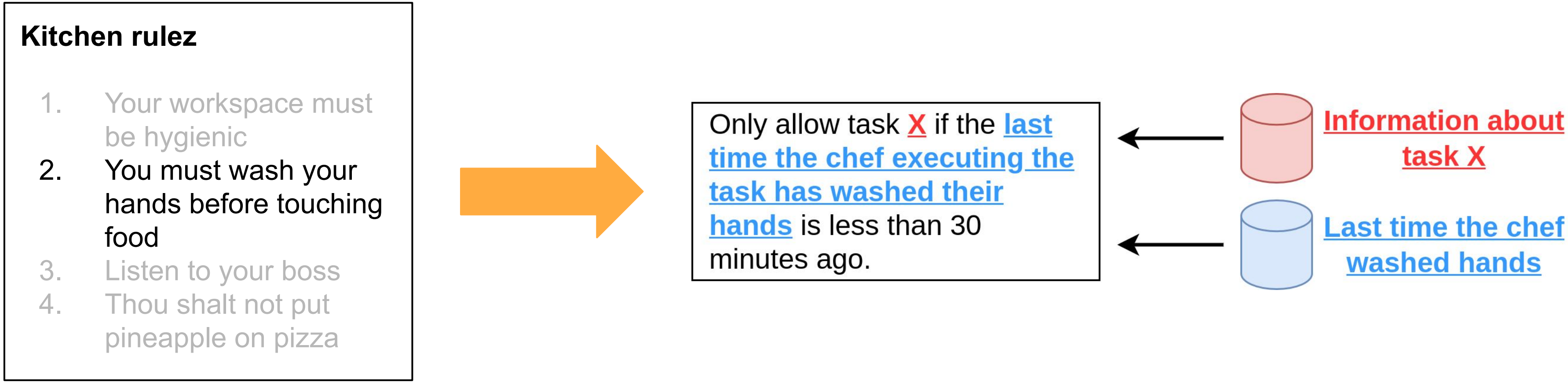
- Policies, however, tend to be **very vague**
  - Especially laws, to allow a judge to interpret
- **Various sources** of vagueness
  - **Abstract** terms (e.g., “What does ‘hygienic’ mean?”)
  - **Incomplete** definition (e.g., “How often do I need to wash my hands?”)
  - Needing **context** information (e.g., “Who is my boss?”)

Kitchen rulez

1. Your workspace must be hygienic
2. You must wash your hands before touching food
3. Listen to your boss
4. Thou shalt not put pineapple on pizza

Constraining recipes

- Luckily, we can scope kitchen policies to **recipes** (workflows)
    - These are already very concrete and formal (executable/computable)
- Thus we can express policies as **rules over recipes**
    - Plus extra context
- Doing so forces us to **get concrete**!



Kitchen rulez

1. Your workspace must be hygienic
2. You must wash your hands before touching food
3. Listen to your boss
4. Thou shalt not put pineapple on pizza

Only allow task X if the **last time the chef executing the task has washed their hands** is less than 30 minutes ago.

Information about task X

Last time the chef washed hands

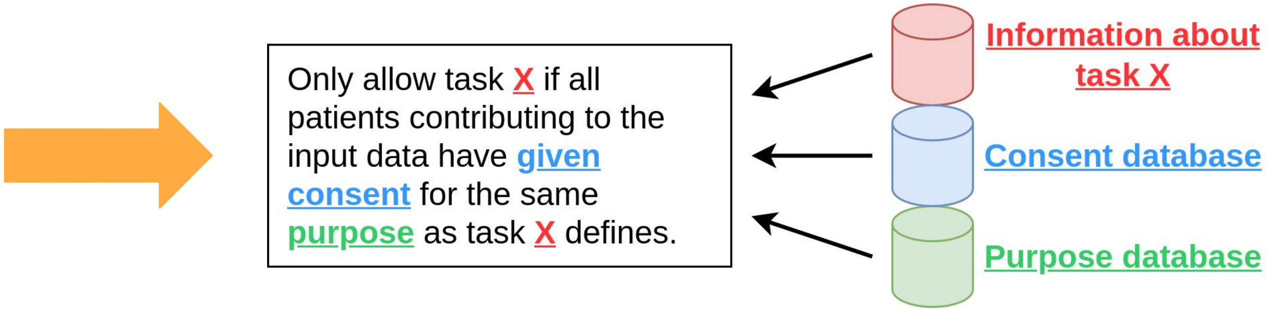
Constraining workflows

- The same therefore applies to EPI Framework **workflows**!
- Can be defined by hospitals **individually**
  - I.e., hospitals are in charge of their own behaviour
- Written by **policy experts**

Policy expert

GDPR (approximately)

1. You can only use data for the purpose patients have given consent for.
2. Patients must be able to revoke their consent
3. …



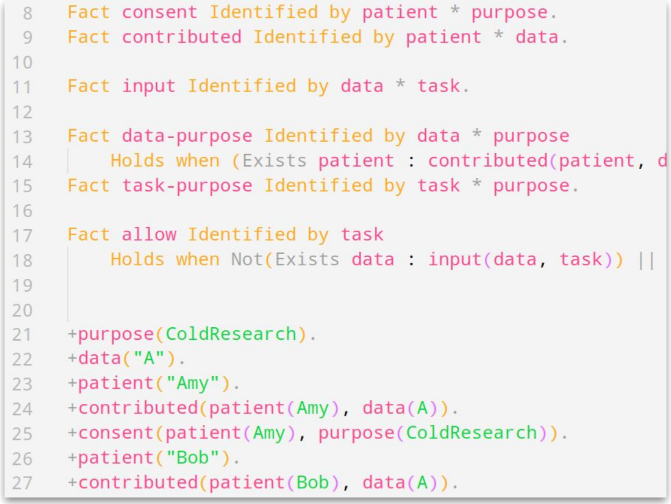
Only allow task X if all patients contributing to the input data have given consent for the same purpose as task X defines.

Expected workflow policy types

- Policies fundamentally control **data**
- Tasks **access data** as part of a workflow
- As such, policies will **allow/deny tasks to process their data**
- Examples:
  - **Laws** (GDPR)
    “Tasks using my data are only allowed if no patients sourcing that data has retracted consent”
  - **Organisational policies**
    “The result of tasks processing this dataset may only be seen by people with this role”
  - **Agreements/Contracts**
    “This dataset is allowed to be transferred to St. Bob if used for this task and this workflow”

Policies as reasoners

- The complexity of policies shows **policies need reasoning**
  - Specifically: policies need to be **logical rules**
- Logic programming languages already exist
  - Datalog[1]
  - eFLINT[2]
  - SEASO[3]
  - …
- However, can be **any language**
  - All that matters is that an **allow/deny** is produced



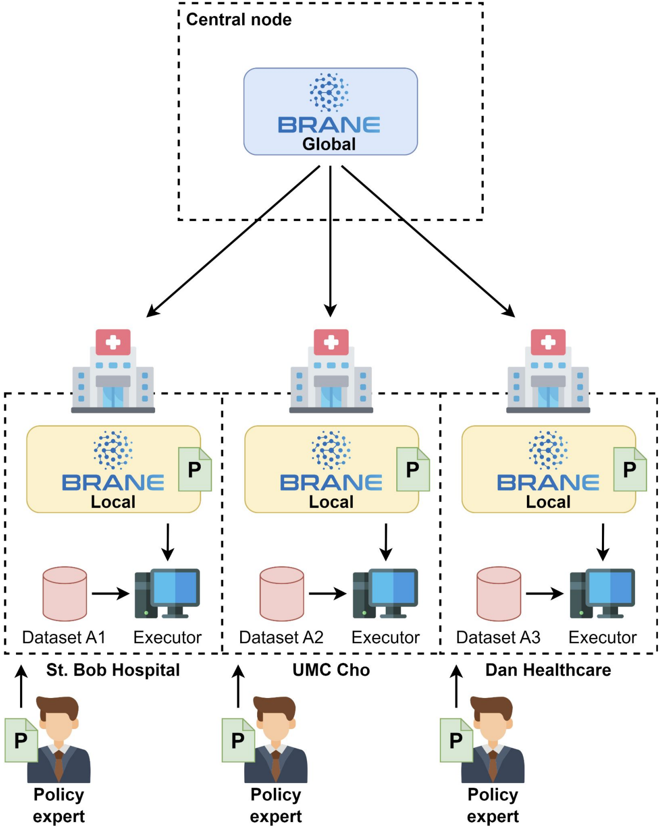
```
Fact consent Identified by patient * purpose.
Fact contributed Identified by patient * data.

Fact input Identified by data * task.

Fact data-purpose Identified by data * purpose
    Holds when (Exists patient : contributed(patient, d
Fact task-purpose Identified by task * purpose.

Fact allow Identified by task
    Holds when Not(Exists data : input(data, task)) ||



+purpose(ColdResearch).
+data("A").
+patient("Amy").
+contributed(patient(Amy), data(A)).
+consent(patient(Amy), purpose(ColdResearch)).
+patient("Bob").
+contributed(patient(Bob), data(A)).
```

eflint seaso

[1] https://www2.cs.sfu.ca/CourseCentral/721/jim/DatalogPaper.pdf
[2] https://gitlab.com/eflint
[3] https://github.com/sirkibsirkib/seaso

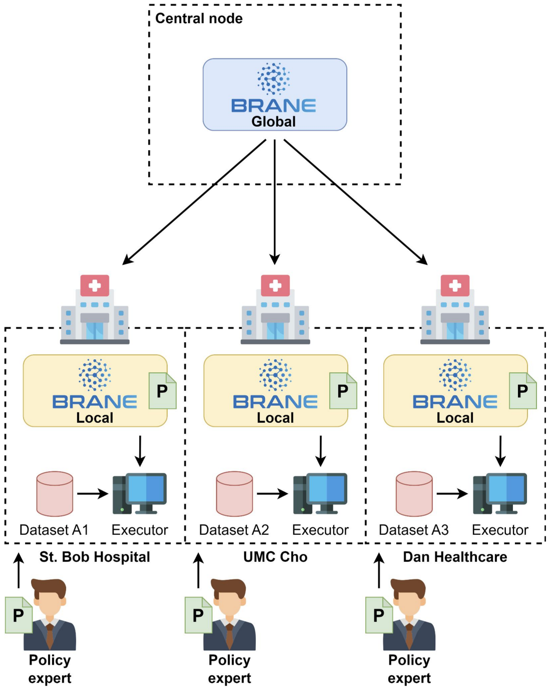
Policies as brain

- Policies determine a **hospital's actions**
- Completely in **hospital control**
  (i.e., hospitals have autonomy)
- This also affords **dynamic updates**
  - Based on situations, new laws, …



Takeaways

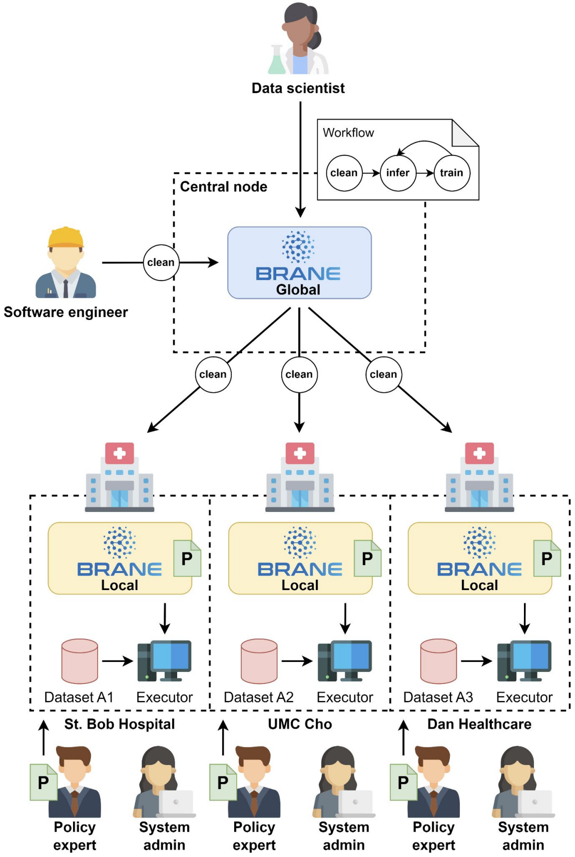
- Policies are **formalisations** of various kinds of rules
    - Formalising laws, organisations policies, agreements, …
    - They are **concretised** versions of the original rule
- Policies **constrain** which tasks are allowed
    - And therefore workflows
- Expressed as **logic rules** (or whatever is needed)



IV. Conclusion

Takeaways

- The **EPI Framework** is a **data-sharing platform**
  - Designed for research context
- Built to **understand the work at hand** (workflow) and **test it to policy**
  - Ensures compliance of the scientist's actions
- **Separation of concerns** to harness complexity



What next?

- If you’re curious, **check the wiki**!
  - https://wiki.enablingpersonalizedinterventions.nl
- System requirements
  - Local: Windows, macOS or Linux machine with Docker[1]
  - Complete: Local machine + Linux server running Docker[1]

Come see the demo in the second hour! :)

[1] https://docker.com

BRANE

Tim Müller (t.muller@uva.nl)

https://enablingpersonalizedinterventions.nl

https://github.com/epi-project/brane

https://wiki.enablingpersonalizedinterventions.nl

The icons (not logos) in this presentation are from: Freepik, Ultimatearm, Vector Valley

V. Bonus slides

The dream of policies

- *“hospitals have autonomy”* (slide 27) is **problematic**!
- It’s crucial that **hospitals are autonomous**…
- …but therefore, we **can’t force them** to (not) do things!
  - They can always leak the data we share somehow
- As such, writing a policy **does not (necessarily!) enforce it**
  - Specifically: **a hospital is guaranteed control *until* shared**
- Policies thus need to **consider trust** in receiving parties