



DIRACX HACKATHON • 1 JULY 2026

DiracX production system

Chris Burr

CERN

Image: [CERN-EX-66954B](#) © 1998–2026 CERN





Introduction

- These slides aim to build a common understanding of:
 - What a production is?
 - How a production is split into transformations?
 - How to make use of these blocks in your own workflows.

- These slides aim to build a common understanding of:
 - What a production is?
 - How a production is split into transformations?
 - How to make use of these blocks in your own workflows.
- After the hackathon, an Architecture Design Review (ADR) will be prepared early in summer.
 - Once agreed upon, start development!

DIRAC vs DiracX

- Here I show the intended production system in DiracX.
- The underpinnings are almost identical to the DIRAC Transformation System.
 - With lots of little tweaks based on experience and lessons learned.

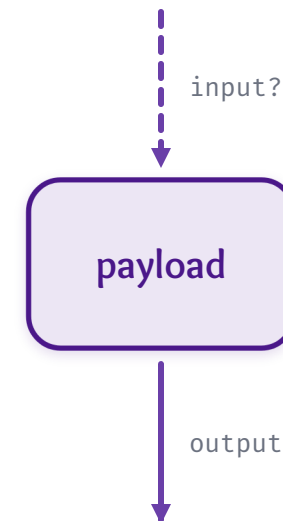




Basic Workflow Management in DiracX

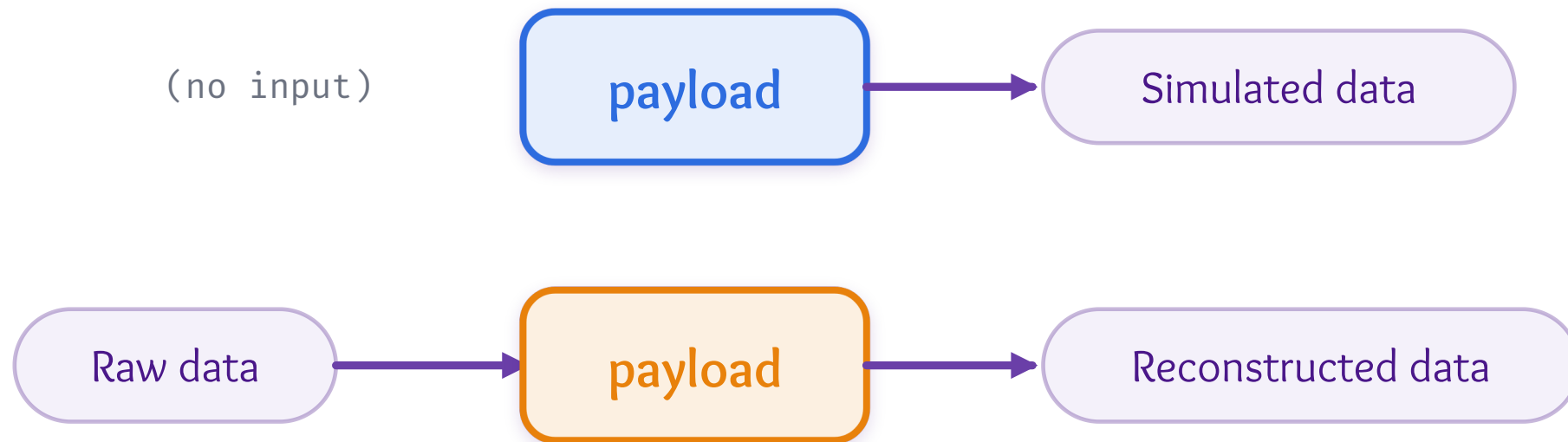
Start with the basics

- We have a black box that:
 - Might have input(s)
 - Produces output(s)
- For now we ignore any source of parallelism



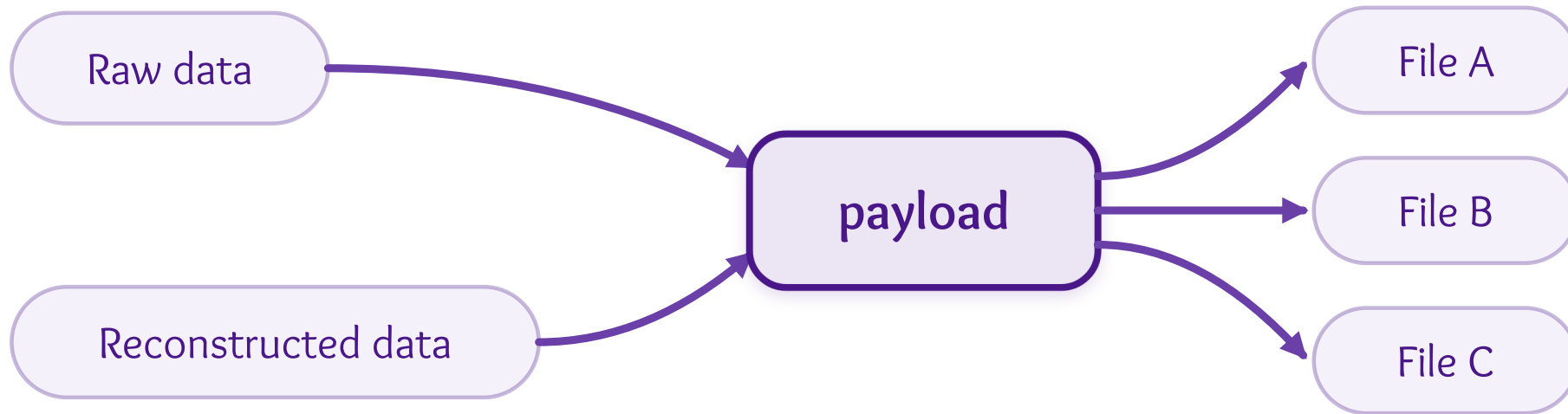
Start with the basics

Simple case: a simulation and a reconstruction



Start with the basics

More complex: correlated inputs, many outputs



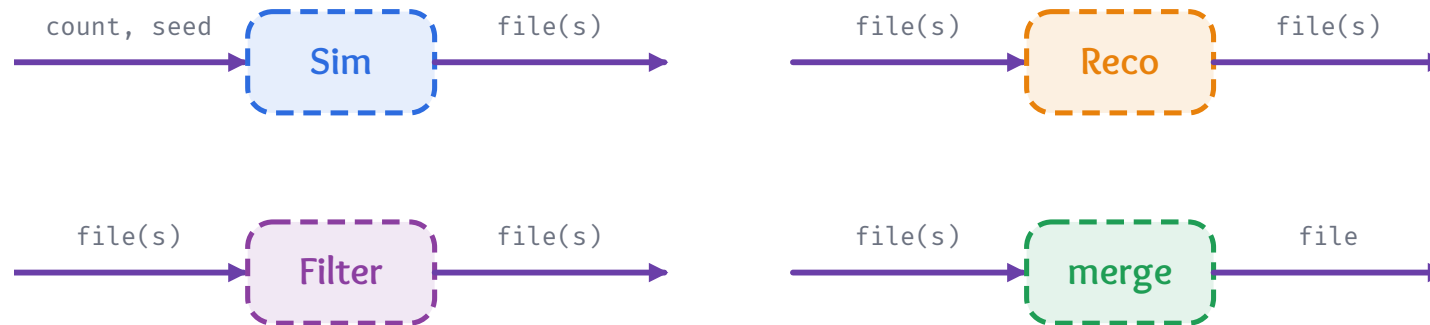
Two steps, one job

- Start with **one job** running two steps end-to-end:
 - **Sim**: simulate events - input is count, seed
 - **Reco**: reconstruct — processes Sim's output
- A step describes **how to run one program** (a command-line tool).



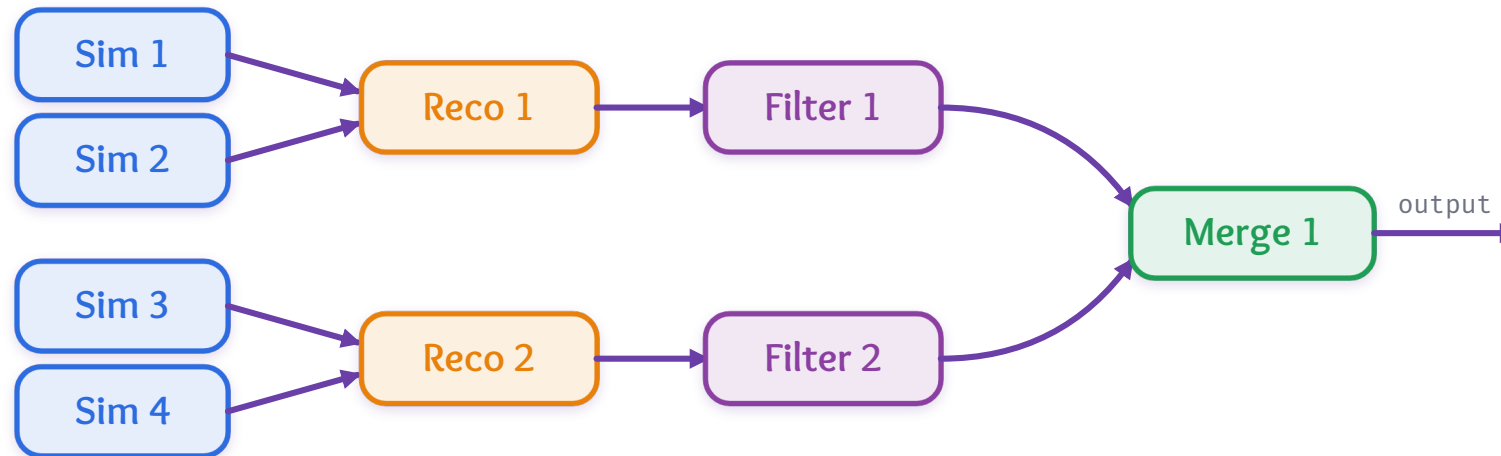
Splitting things up

- One job hits limits — **disk, CPU time, memory.**
- Split into steps which are abstract building blocks



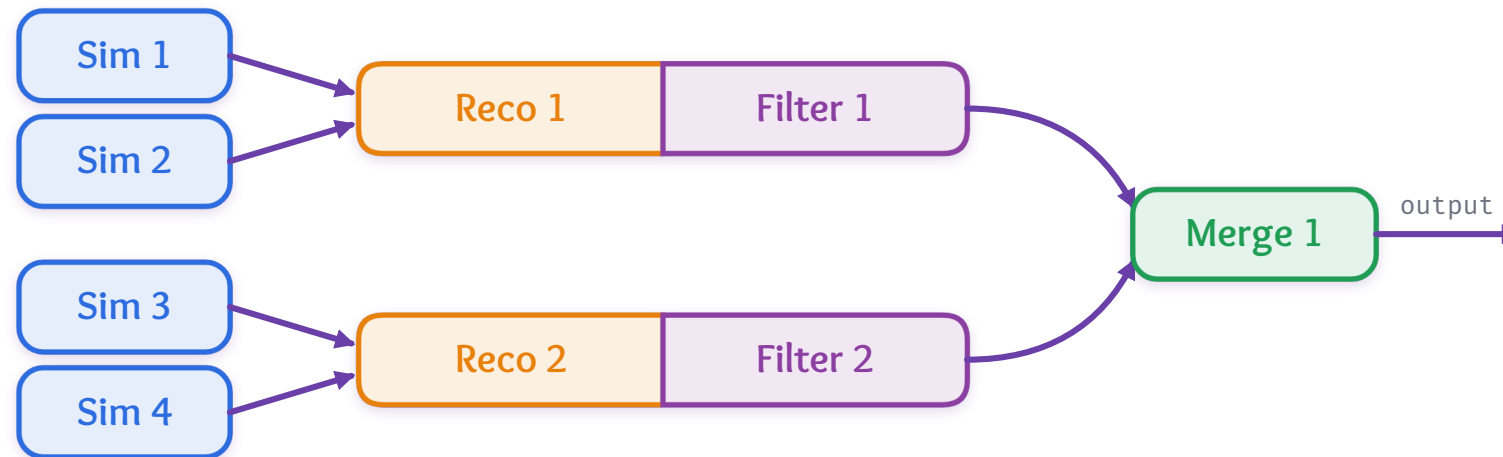
Parallelising the work

- Many Sim jobs feed each Reco → Filter branch.
- The branches **merge** into the final output.
- Each block in the diagram is a job



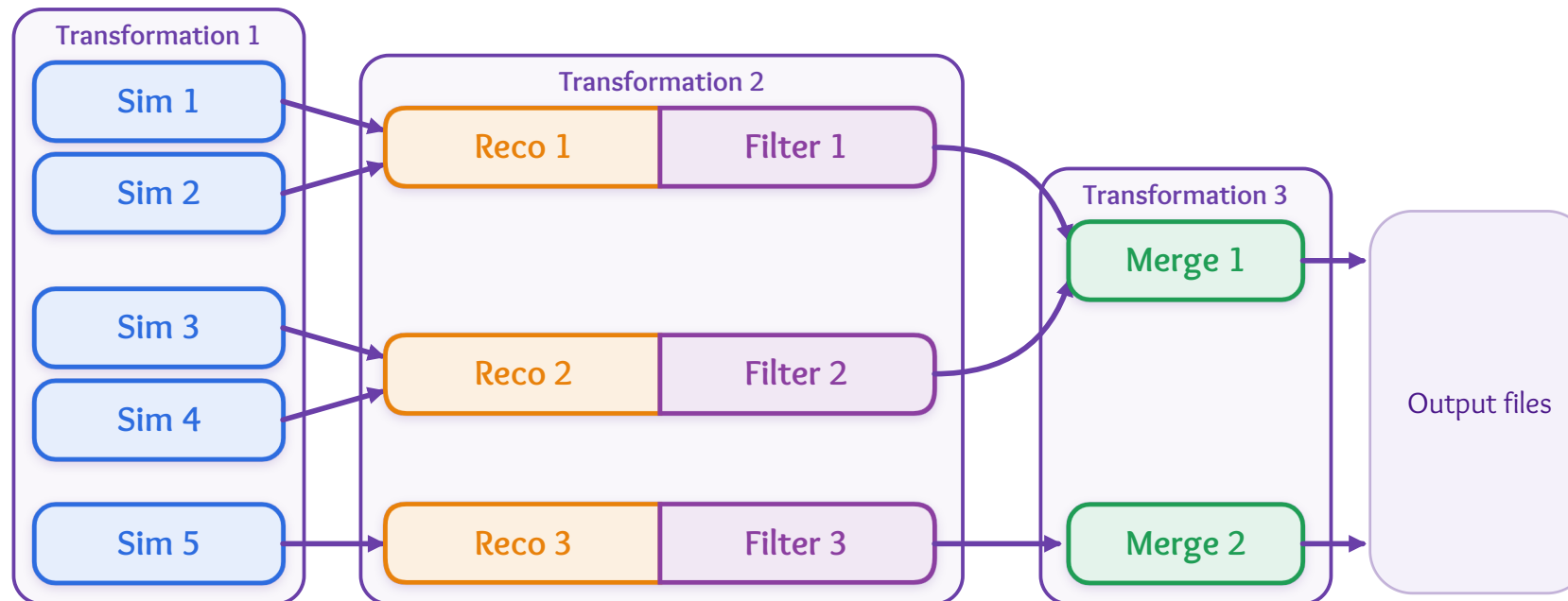
Parallelising the work

- No need to separate the **Reco** and **Filter** steps into separate jobs.
- Could be streamed between them or ran sequentially.
- This diagram repeats N times times to produce more simulation.



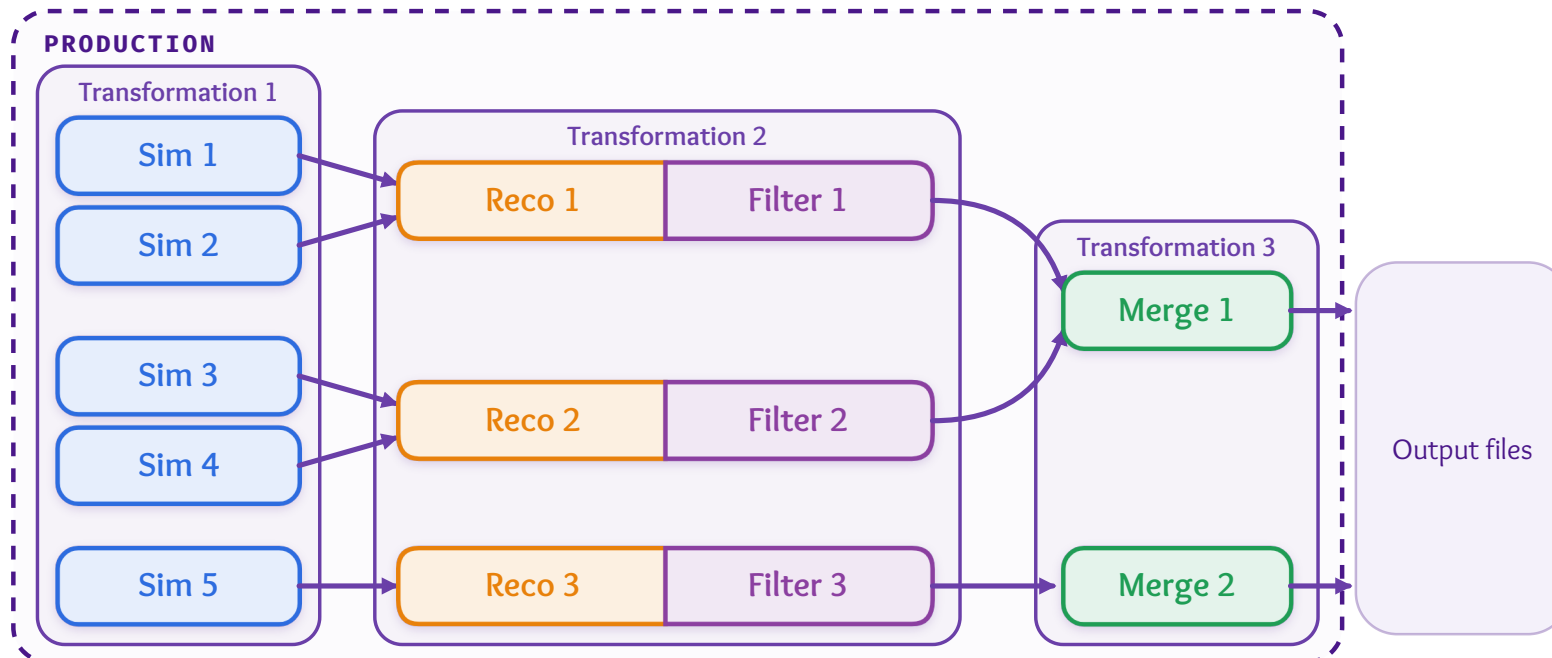
Transformation

- Transformations contain many jobs which each do the same thing
- Transformations can be chained together



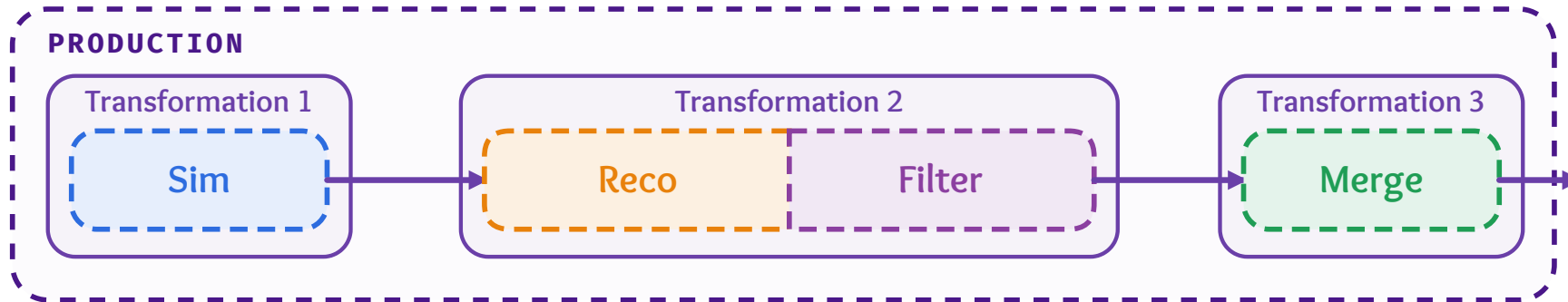
Productions

- A Production groups one or more transformations together
 - Might also include data management transformations (e.g. archive output to tape)
 - I'll come to those later



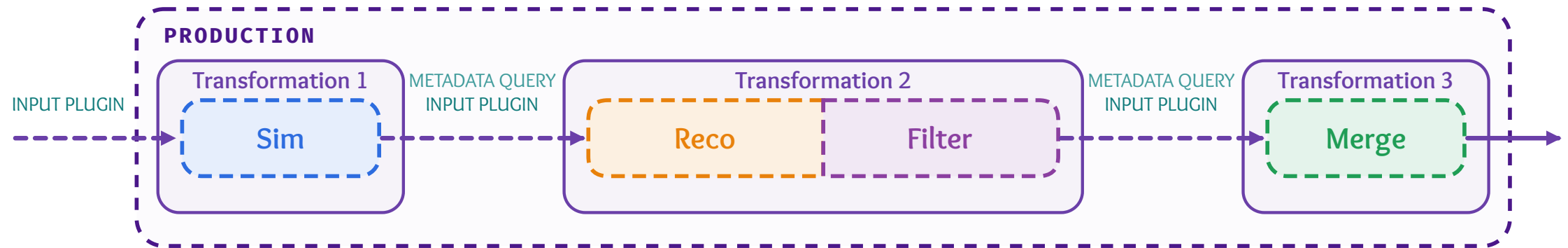
Becoming abstract again

- When defining a transformation the jobs don't exist
 - In workflow management terms, they form a dynamic DAG



Connecting it all together

- Each transformation has an **input plugin**: controls when to create tasks
- Transformations with input data have an **metadata query**





Transformation state

- Transformations have **transformation input** (typically a list of LFNs)

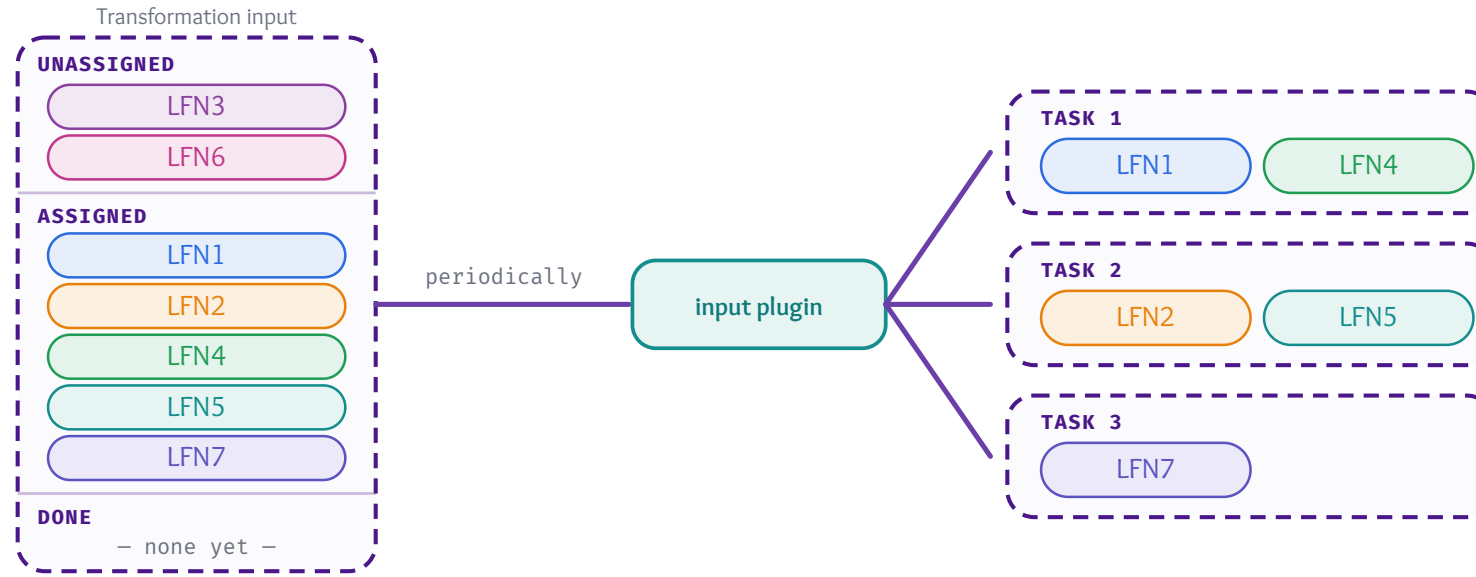


Transformation state

- Transformations have **transformation input** (typically a list of LFNs)
- Periodically uses the **input plugin** to decide when to create a **task**

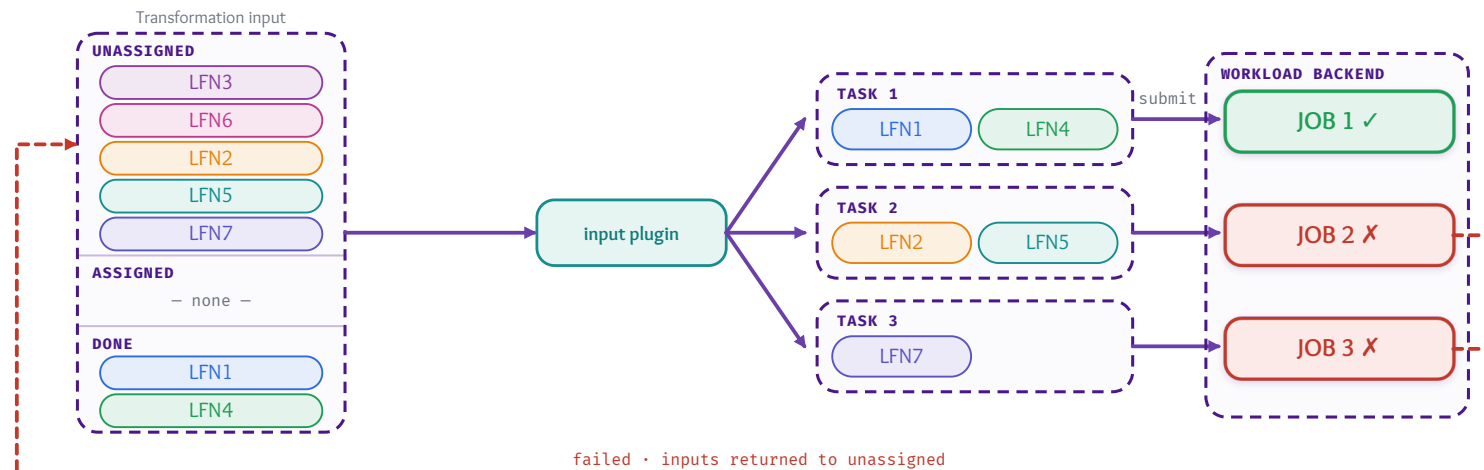
Transformation state

- Transformations have **transformation input** (typically a list of LFNs)
- Periodically uses the **input plugin** to decide when to create a **task**
- One or more **transformation input(s)** are assigned to a **task**



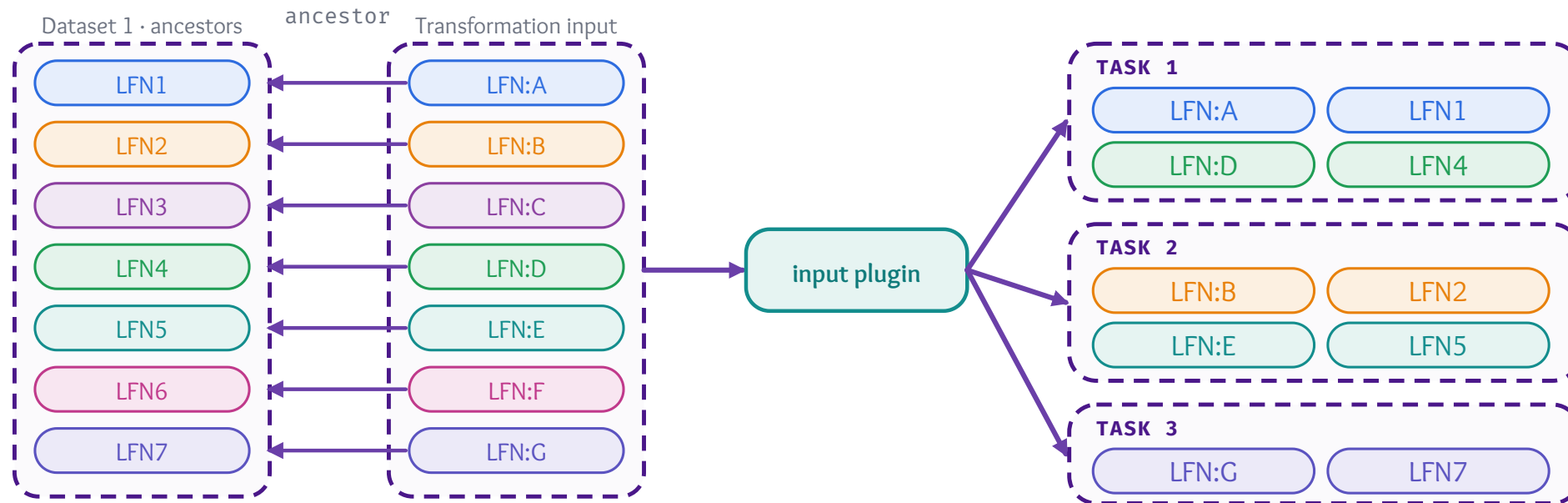
Transformation tasks

- A transformation's **task** is submitted to a backend
- **Tasks** are not retryable -> added back to the pool of unassigned **transformation input(s)**
 - An input is only ever successfully processed by one **task**
- Next time the **input plugin** is run, a new **task** might be created
 - Potentially with different inputs



Grouping correlated inputs

- The **input plugin** can inject additional inputs
- For example also include an ancestor file





Data management transformations

- The other side of the transformation system is data management.
 - Two primitives: **Copy** and **Delete**
 - Tasks know the final data distribution state you want to obtain



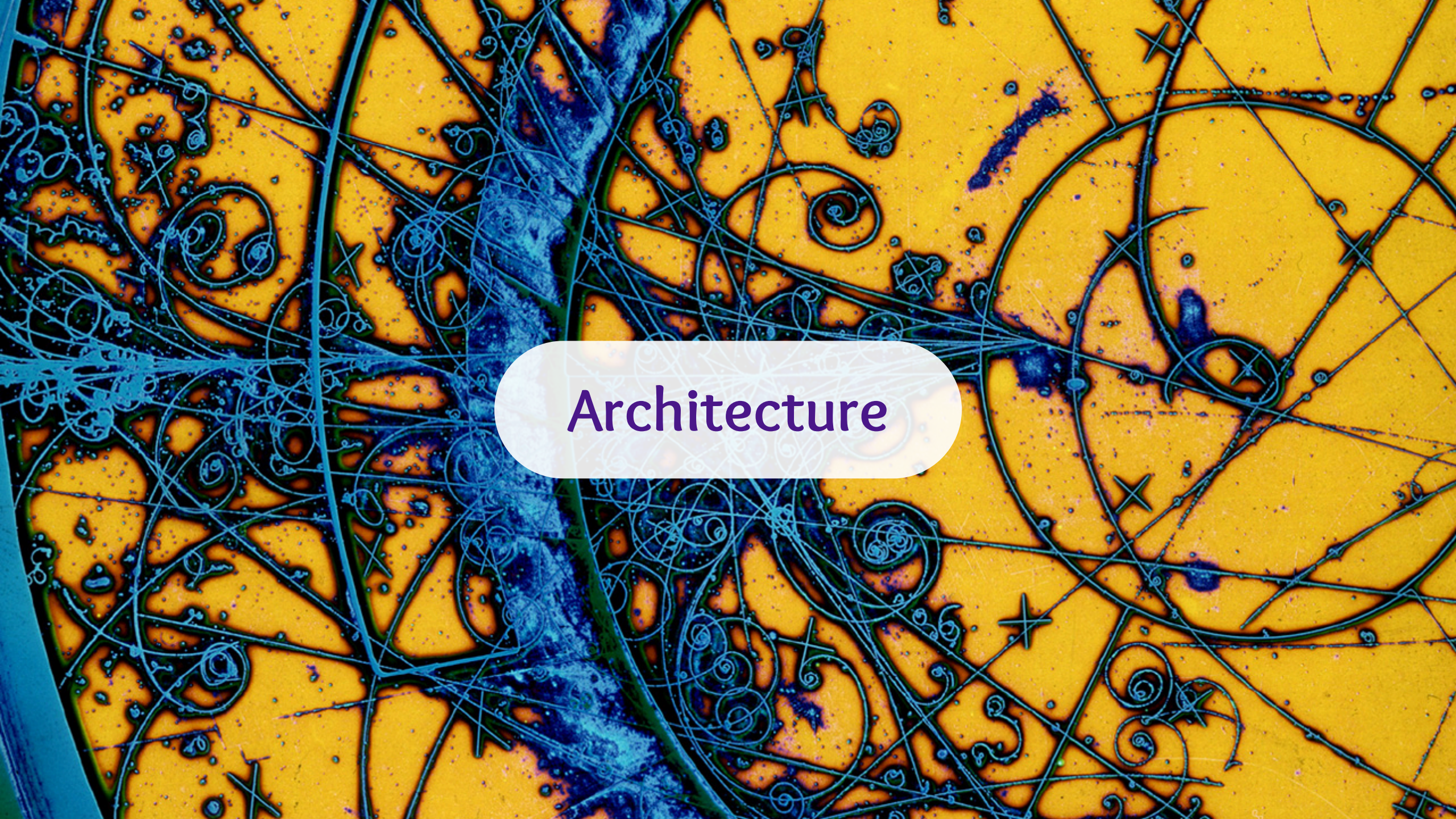
Data management transformations

- The other side of the transformation system is data management.
 - Two primitives: **Copy** and **Delete**
 - Tasks know the final data distribution state you want to obtain
- Similarities to workload management transformations:
 - Input metadata query to create tasks
 - Tasks have input LFNs



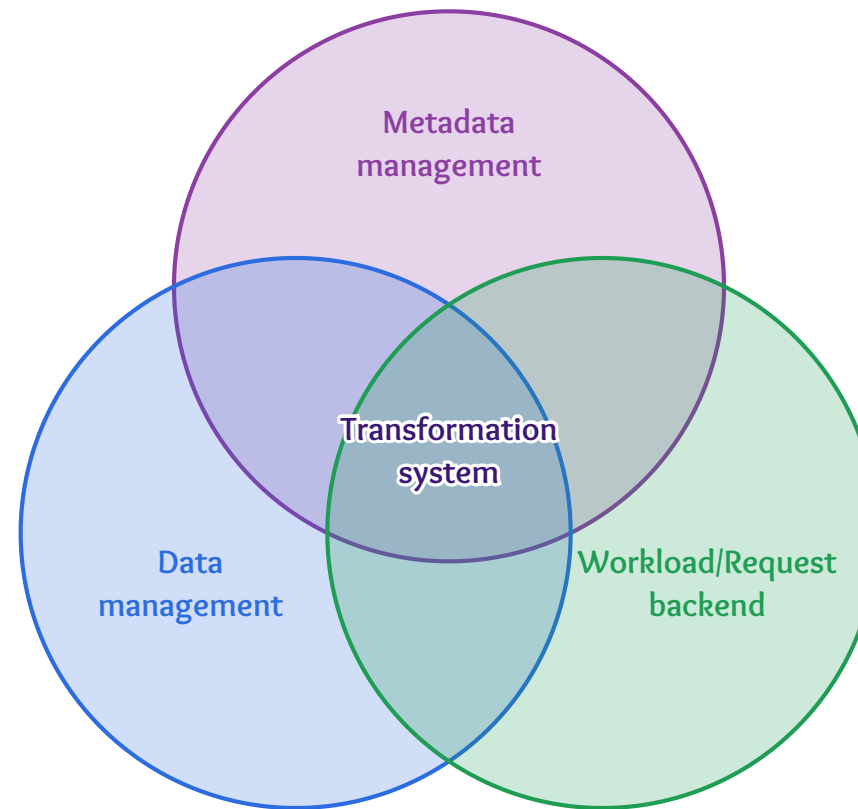
Data management transformations

- The other side of the transformation system is data management.
 - Two primitives: **Copy** and **Delete**
 - Tasks know the final data distribution state you want to obtain
- Similarities to workload management transformations:
 - Input metadata query to create tasks
 - Tasks have input LFNs
- Differences:
 - No Jobs (uses Requests instead)
 - No CWL
 - No "output"



Architecture

The three components of the transformation system





Metadata management

- Extremely experiment specific
- Basics:
 - Steers what LFNs are picked up by a transformation[†]
 - Output of transformations are registered
- Can also provide:
 - ancestry information for correlated input
 - descendent information for additional safety checks

[†] LFNs can be added manually for special cases.



Data management

- Provides information about
 - LFN availability
 - Available storage at sites

- Provides information about
 - LFN availability
 - Available storage at sites
- Example uses of transformation **input plugins**:
 - Group LFNs to all have replicas at the same locations
 - Don't create tasks for LFNs that are at locations with downtime



Workload/Request backend

- Workload/Request backend takes care of actual **task** execution



Workload/Request backend

- Workload/Request backend takes care of actual **task** execution
- For workload transformations:
 - Each workload task has:
 - Matching criteria
 - Zero or more input LFNs and input sandboxes
 - An associated workflow
 - Workload backend is responsible for:
 - Scheduling the task to a worker node
 - Starting the DiracX job wrapper



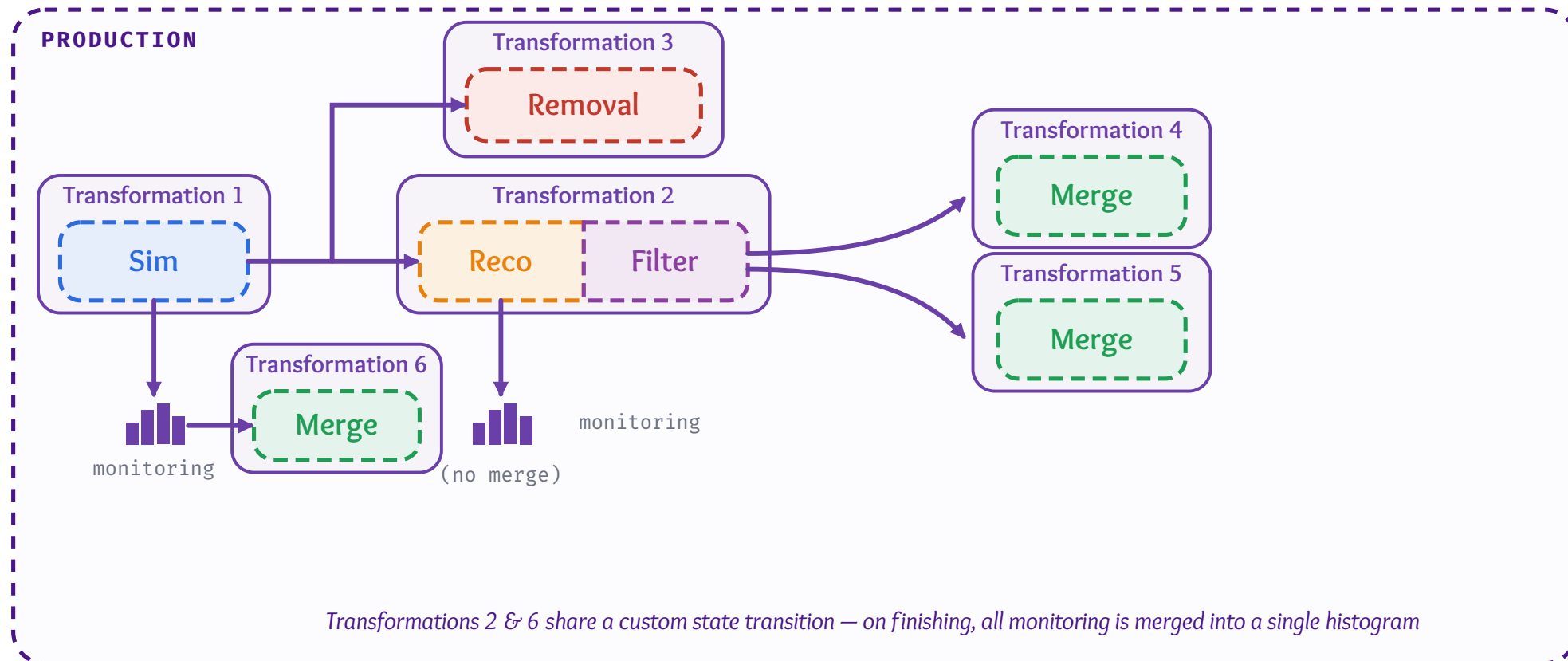
Workload/Request backend

- Workload/Request backend takes care of actual **task** execution
- For workload transformations:
 - Each workload task has:
 - Matching criteria
 - Zero or more input LFNs and input sandboxes
 - An associated workflow
 - Workload backend is responsible for:
 - Scheduling the task to a worker node
 - Starting the DiracX job wrapper
- For data management transformations this is handled by **Requests**



Wrapping up

Putting it all together





Higher level concepts

- Here I've only talked about the low-level system
- Examples in LHCb:
 - "Analysis productions": [CHEP-2026](#)
 - "mc-requests" and "LbMCSubmit": [CHEP-2023](#) and [CHEP-2024](#)
 - Fluka simulations: [FLUKA.CERN collaboration meeting](#)

"Analysis Productions" model: Declaring workflows

- The idea is to have a high-level declarative way of declaring any workflow
- With a escape hatch to allow for deeper customization
- Or just write a DIRAC workflow directly (mostly for expert users)

```
sim-version: 09
name: My Analysis
WG: Charm
samples:
  - event-types:
    - 23103006
    - 27165175
    - 30000000
  data-types:
    - 2016
    - 2017
    - 2018
num-events: 2_500_000
fast-mc:
  redecay: yes
```



"Analysis Productions" model: Submission

- Submission is then Git-style CI/CD driven



"Analysis Productions" model: Submission

- Submission is then Git-style CI/CD driven
- Can run tests locally (optional)



"Analysis Productions" model: Submission

- Submission is then Git-style CI/CD driven
- Can run tests locally (optional)
- Prior to submission, a test is ran automatically
 - Communicate back to users informaton in a friendly way
 - Measure resource requirements
 - Approval rules added automatically for restricted productions



"Analysis Productions" model: Submission

- Submission is then Git-style CI/CD driven
- Can run tests locally (optional)
- Prior to submission, a test is ran automatically
 - Communicate back to users informaton in a friendly way
 - Measure resource requirements
 - Approval rules added automatically for restricted productions
- When merged the production runs



"Analysis Productions" model: Submission

- Submission is then Git-style CI/CD driven
- Can run tests locally (optional)
- Prior to submission, a test is ran automatically
 - Communicate back to users informaton in a friendly way
 - Measure resource requirements
 - Approval rules added automatically for restricted productions
- When merged the production runs

We won't have time for this now, at the next workshop...



Questions?