Implementation of the ATLAS trigger within the ATLAS Multi-Threaded Software Framework: AthenaMT

IEEE eScience

29 October - 1 November 2018, Amsterdam

Tim Martin, University of Warwick
On behalf of the **ATLAS Collaboration**
`Tim.Martin@cern.ch`

Slides: `http://cern.ch/go/97FW`

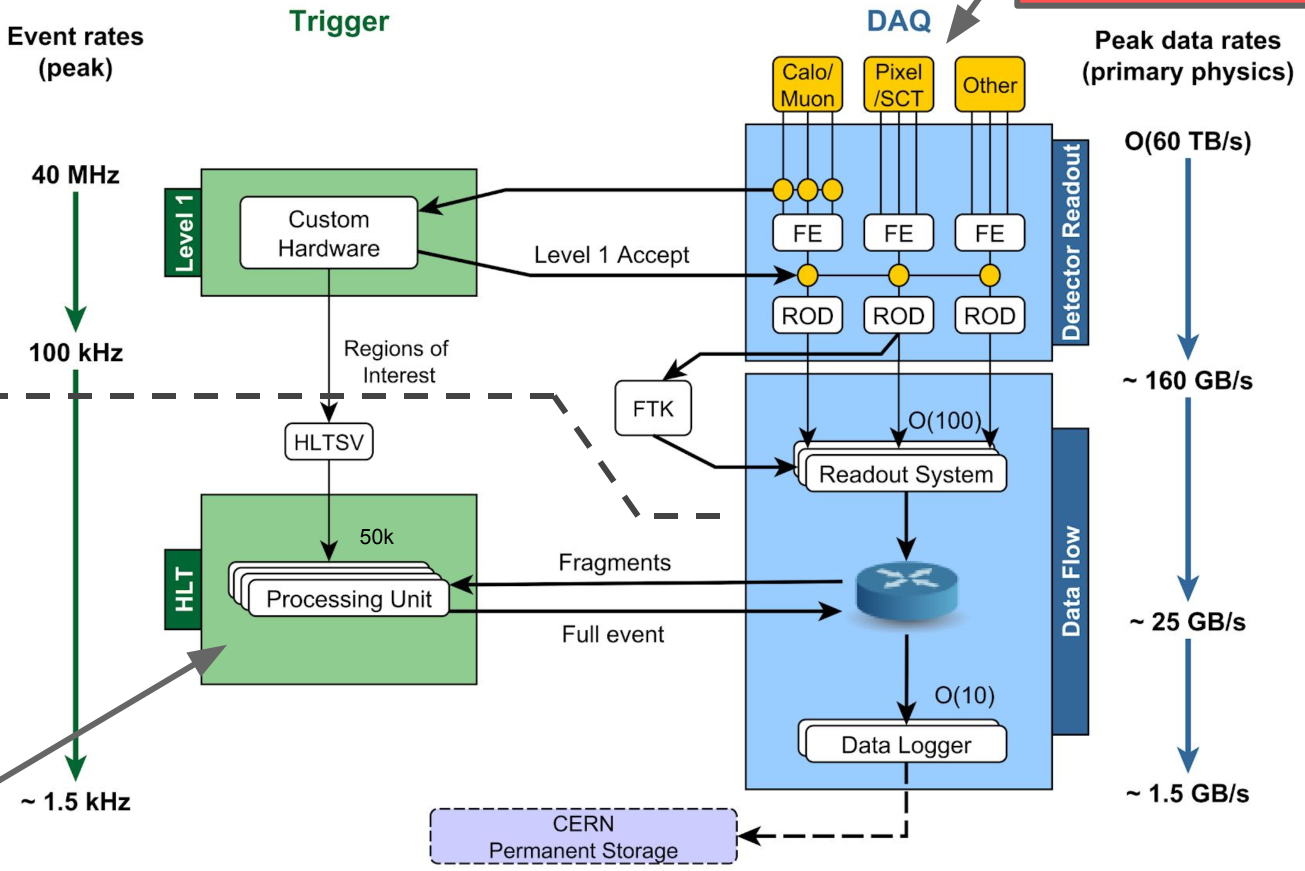
ATLAS Trigger & Data Acquisition

Detector Electronics

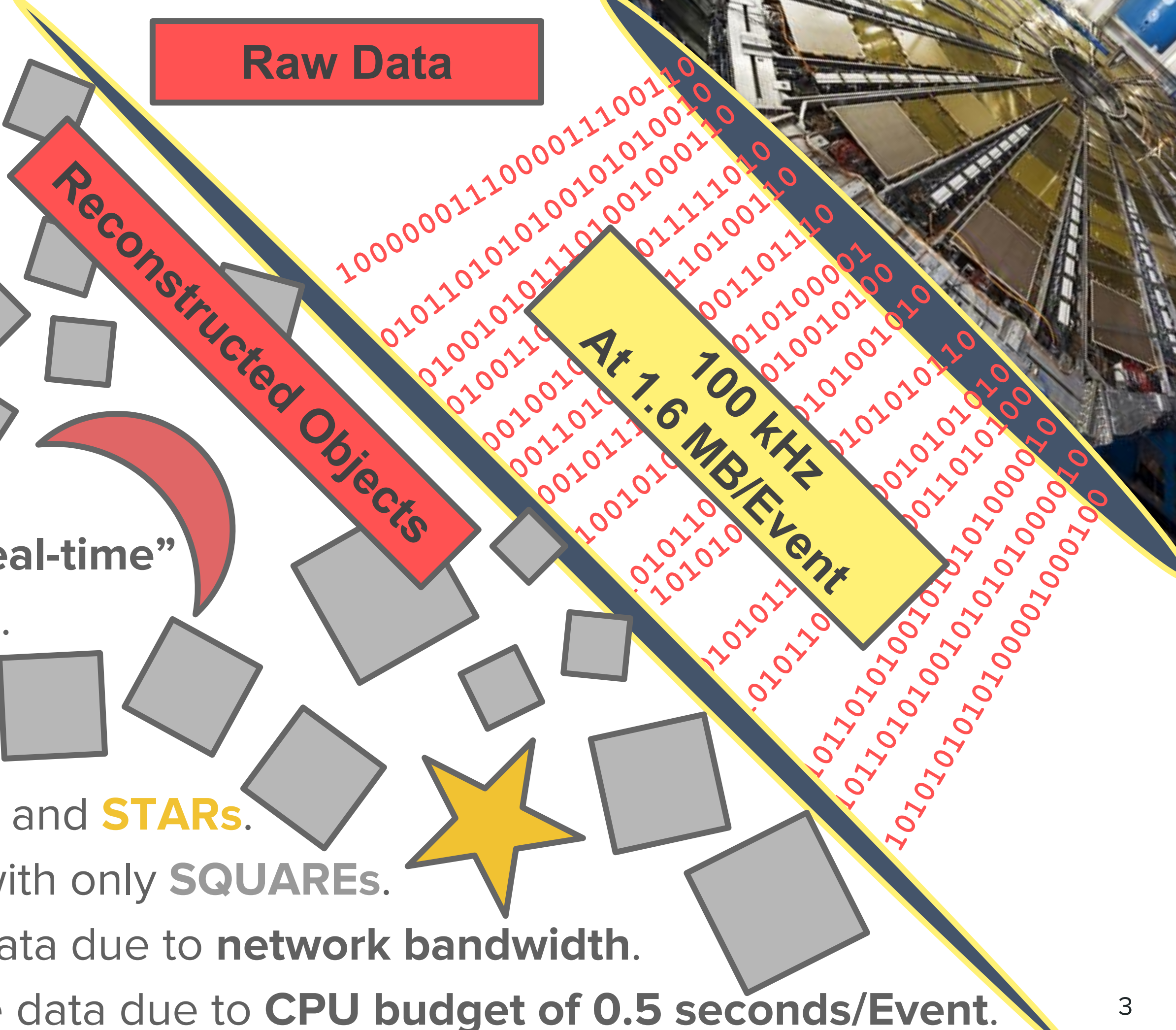
Hardware Dominant Domain

Software Dominant Domain

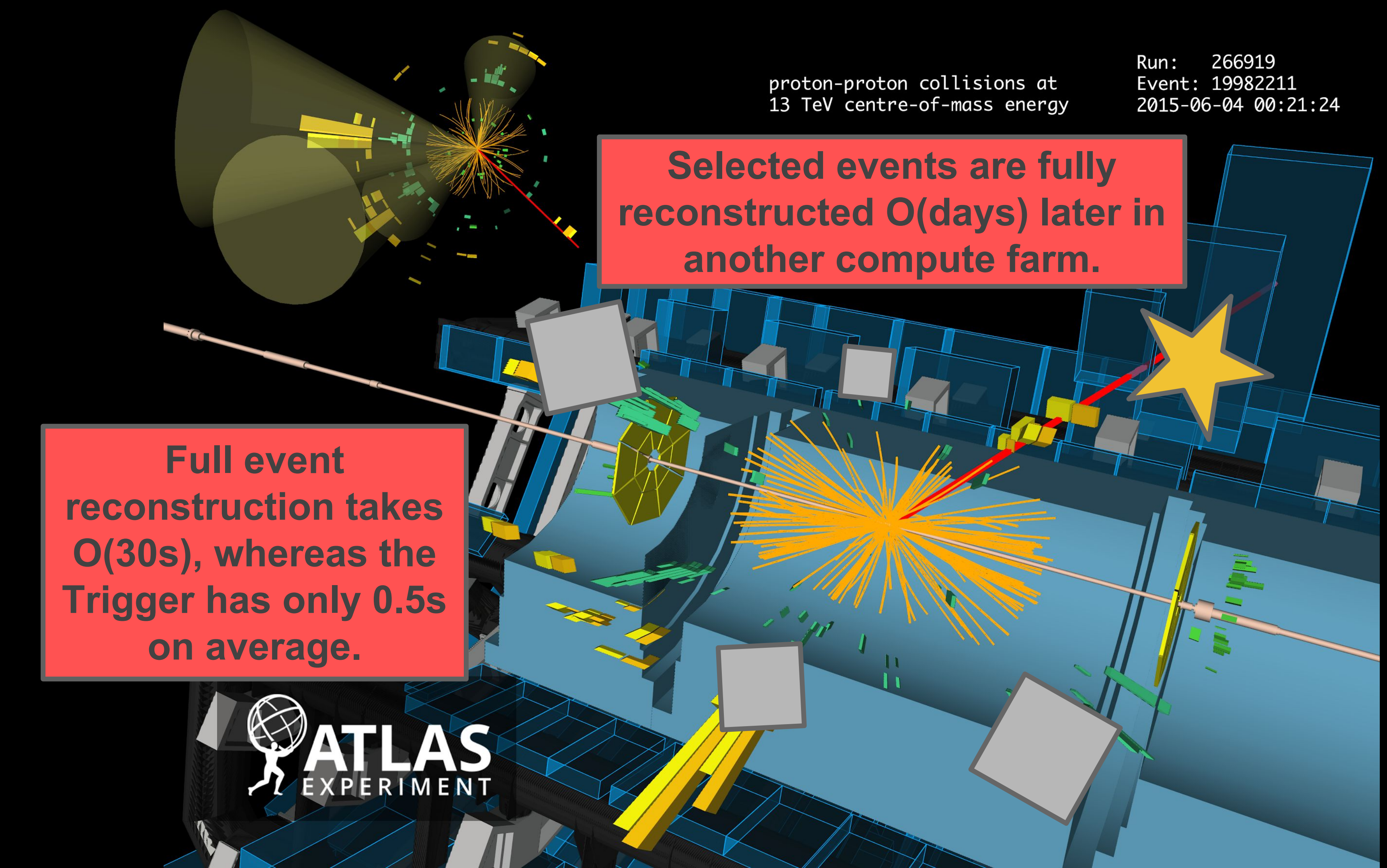
Focus of today's presentation



An Abstraction



- Process **100 kHz** events in **“real-time”**
- Every event will **PASS** or **FAIL**.
- **FAIL** events are lost **forever**.
- Only **1%** of events can **PASS**.
- We **want** events with **MOONs** and **STARs**.
- We **don’t care** about events with only **SQUAREs**.
- We **cannot look** at all of the data due to **network bandwidth**.
- We **cannot reconstruct** all the data due to **CPU budget of 0.5 seconds/Event**.

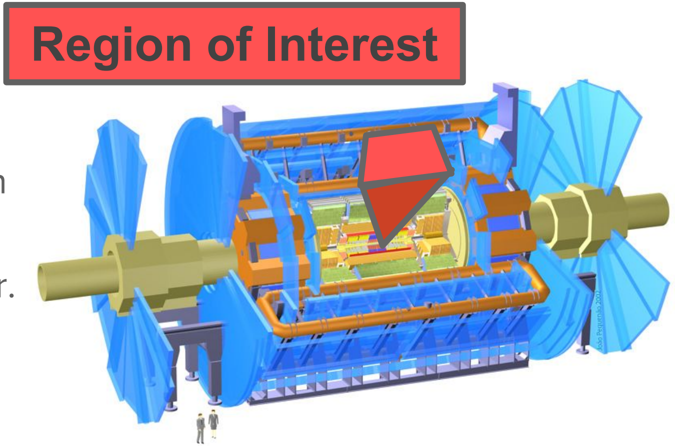


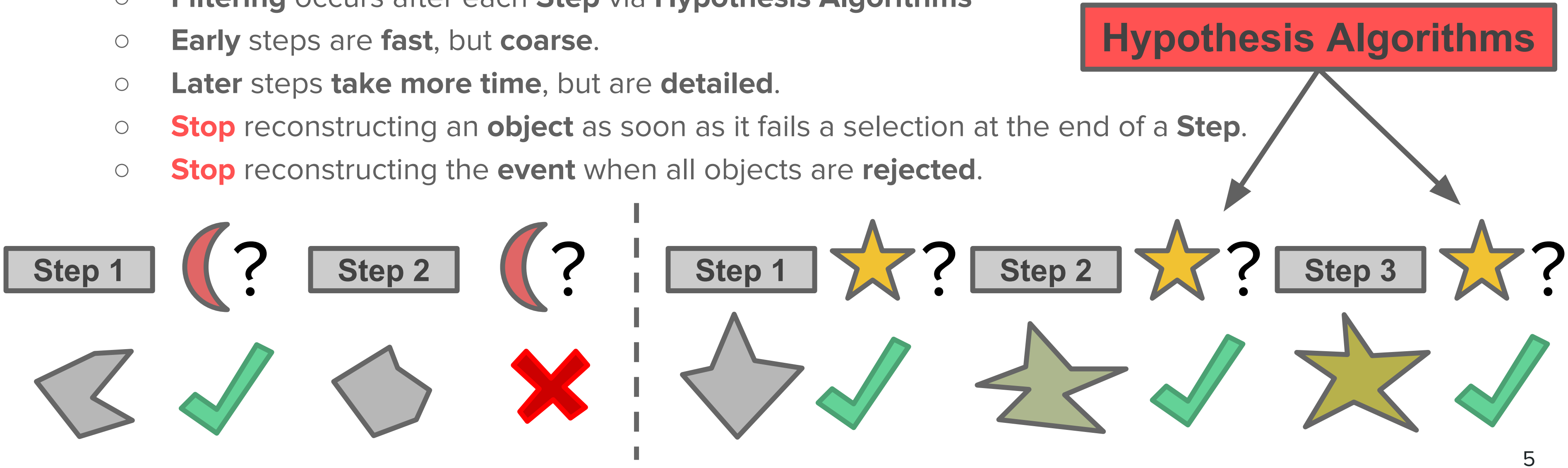
Selected events are fully reconstructed O(days) later in another compute farm.

Full event reconstruction takes O(30s), whereas the Trigger has only 0.5s on average.

Key Principles of the Trigger

- **Regional Reconstruction**
  - We **cannot** look at all **1.6 MB** of every event due to bandwidth
  - Restrict to running **reconstruction algorithms** within **Regions of Interest**, identified in the 1st level hardware trigger.
- **Early Rejection**
  - Split reconstruction up into multiple **Steps**.
  - **Filtering** occurs after each **Step** via **Hypothesis Algorithms**
  - **Early** steps are **fast**, but **coarse**.
  - **Later** steps **take more time**, but are **detailed**.
  - **Stop** reconstructing an **object** as soon as it fails a selection at the end of a **Step**.
  - **Stop** reconstructing the **event** when all objects are **rejected**.





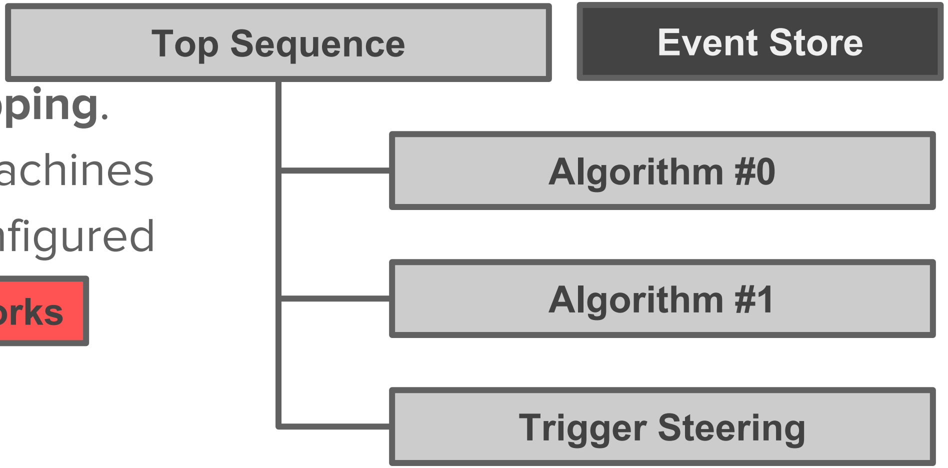
Software Framework: Athena

O(4M) lines of C++ & O(1M) lines of python.

- The **ATLAS Software framework**, **Athena**, is built on top of the inter-experiment **Gaudi** framework (shared e.g. with the **LHCb** collaboration).
    - **C++ Algorithms**, **Services**, **Tools** etc. with **Python configuration**.
- For each event a **sequential list** of algorithm executed. Algorithms are assumed to depend only on other algorithms scheduled earlier in the list.
- One common singleton **Event Store** handles **transient** and **persistent** data access.
- High Level Trigger uses **custom steering**, regular algorithms need **wrapping**.
- In the High Level Trigger, **multi-core** machines are currently utilised by **forking** the configured HLT process instance.

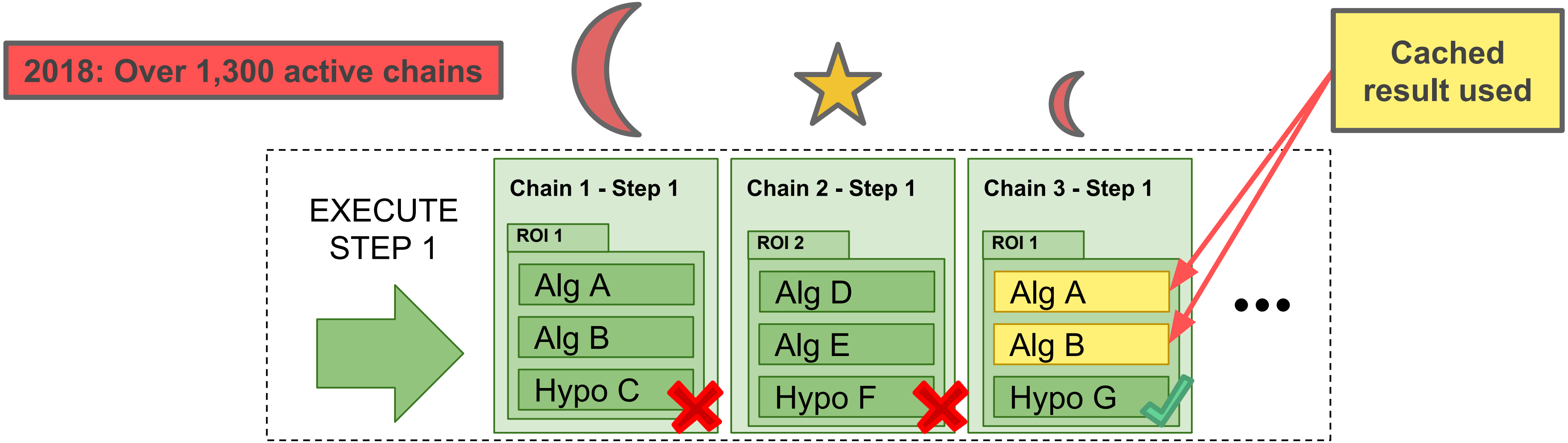
O(30) forks

- Memory is shared with **copy-on-write**. Introduces **overheads** when pages are modified after the fork.



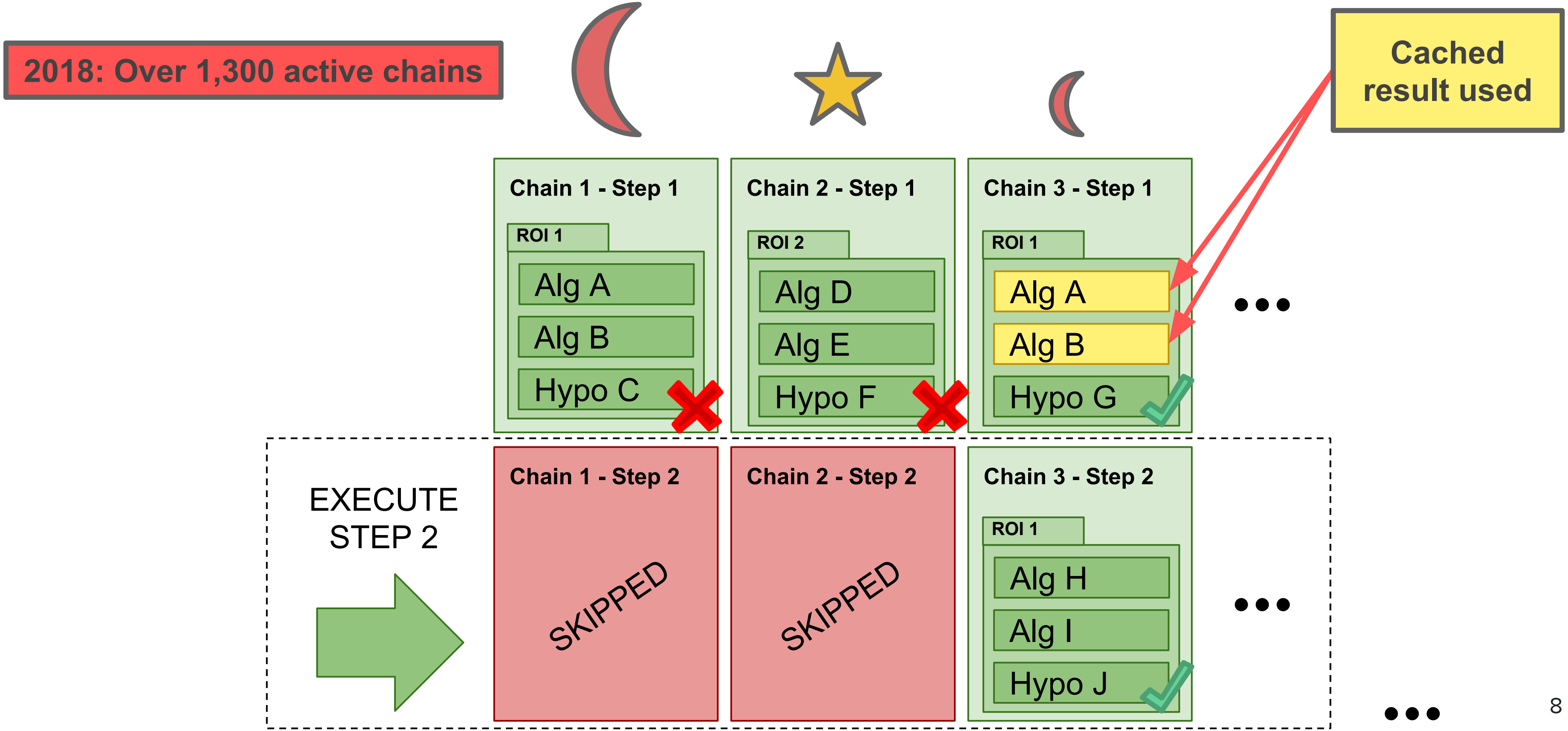
Current Single Threaded Trigger Architecture

- Object selections are encoded in **Chains**. Each step runs in **serial**.



Current Single Threaded Trigger Architecture

- Object selections are encoded in **Chains**. Each step runs in **serial**.



Moving on to Athena Multi-Threaded

Future Athena Multi Threaded (MT)

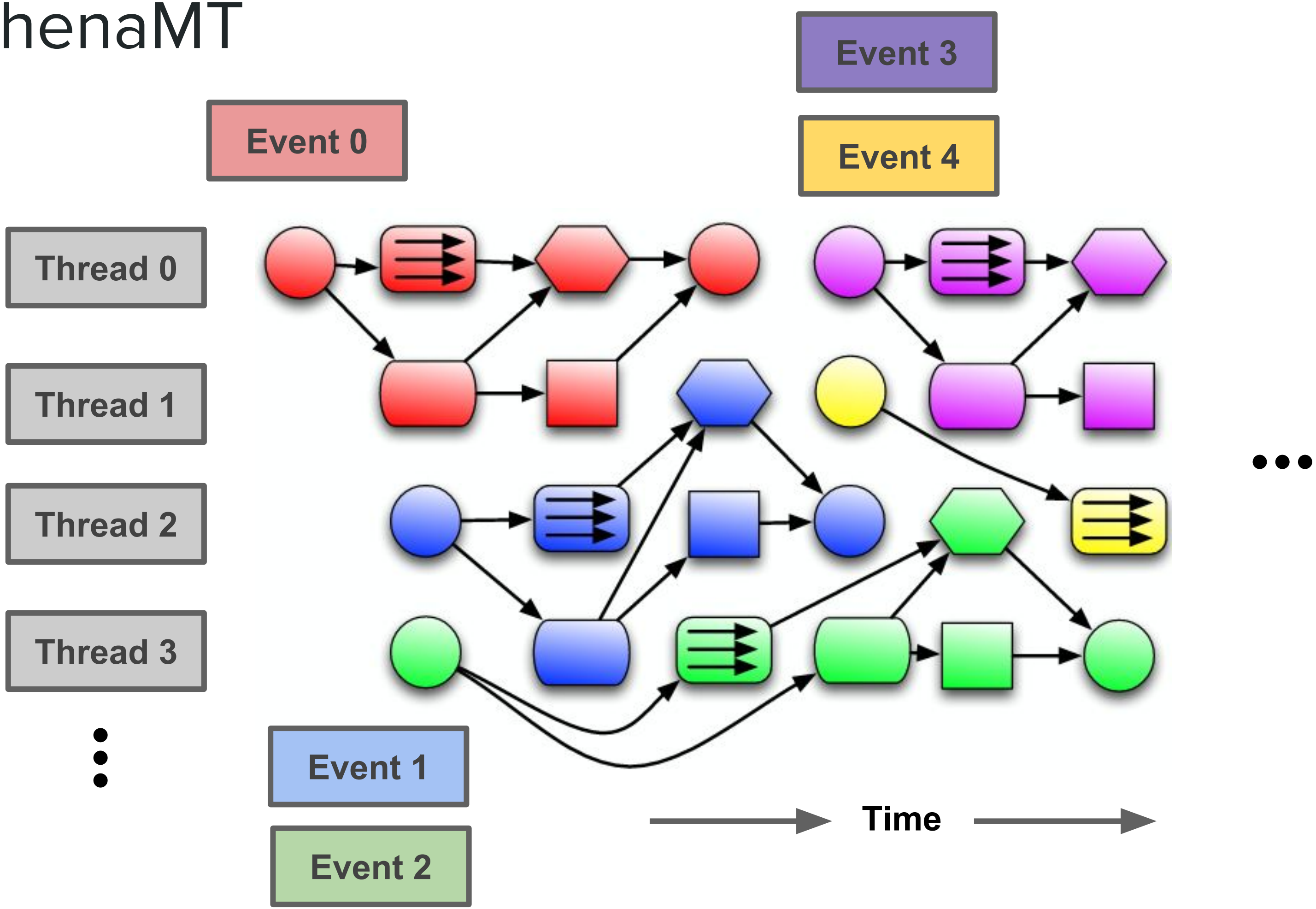
Thread Building Blocks (link)

- **AthenaMT** is built on the **Gaudi Hive (Intel TBB)** multi-threaded architecture.
- Offers **Intra-Event parallelisation**.
  - The **Algorithm Scheduler** is configured with the **Input-** and **Output-Data Handles** of all algorithms. Builds a **Data Dependency** graph.
  - Multiple algorithms within an event can run in parallel, provided that their input Data Handles (if any) are available.
- Offers **Inter-Event parallelisation**.
  - **Multiple events** may be being processed simultaneously: **"in flight"**.
  - Optimal memory efficiency if all algorithms are **re-entrant**, i.e. **stateless** and able to run on **multiple concurrent events** (alternate: cloneable).
- Offers **In-Algorithm parallelisation**.
  - Algorithm authors may make use of e.g. **parallel for-loops**.

Goal: Maximise memory efficiency & keep all threads busy.

Goal: No Trigger-specific steering layer. No wrappers.

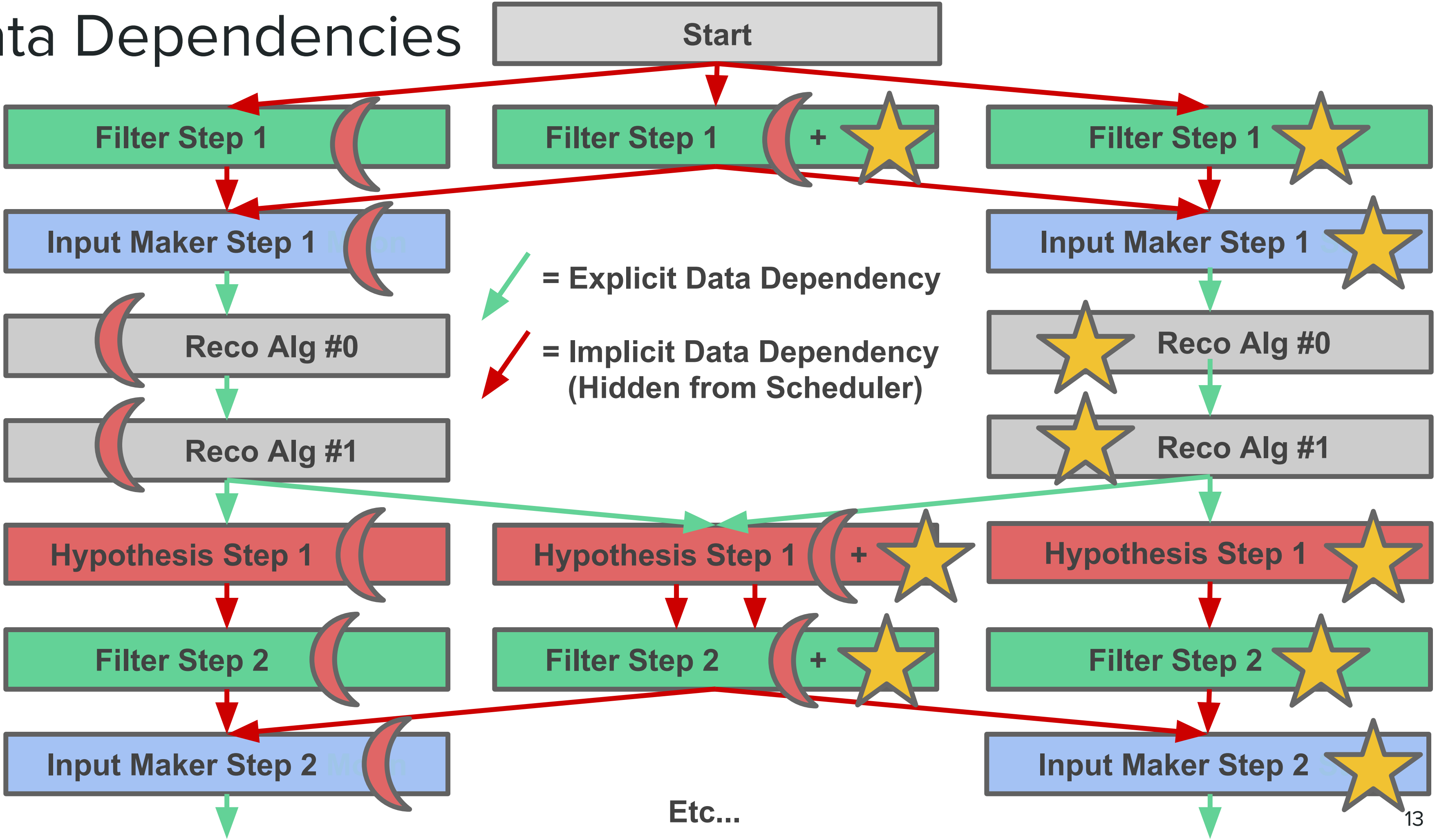
AthenaMT


Event 3
Event 0
Event 4
Thread 0
Thread 1
Thread 2
Thread 3
Event 1
Event 2
Time

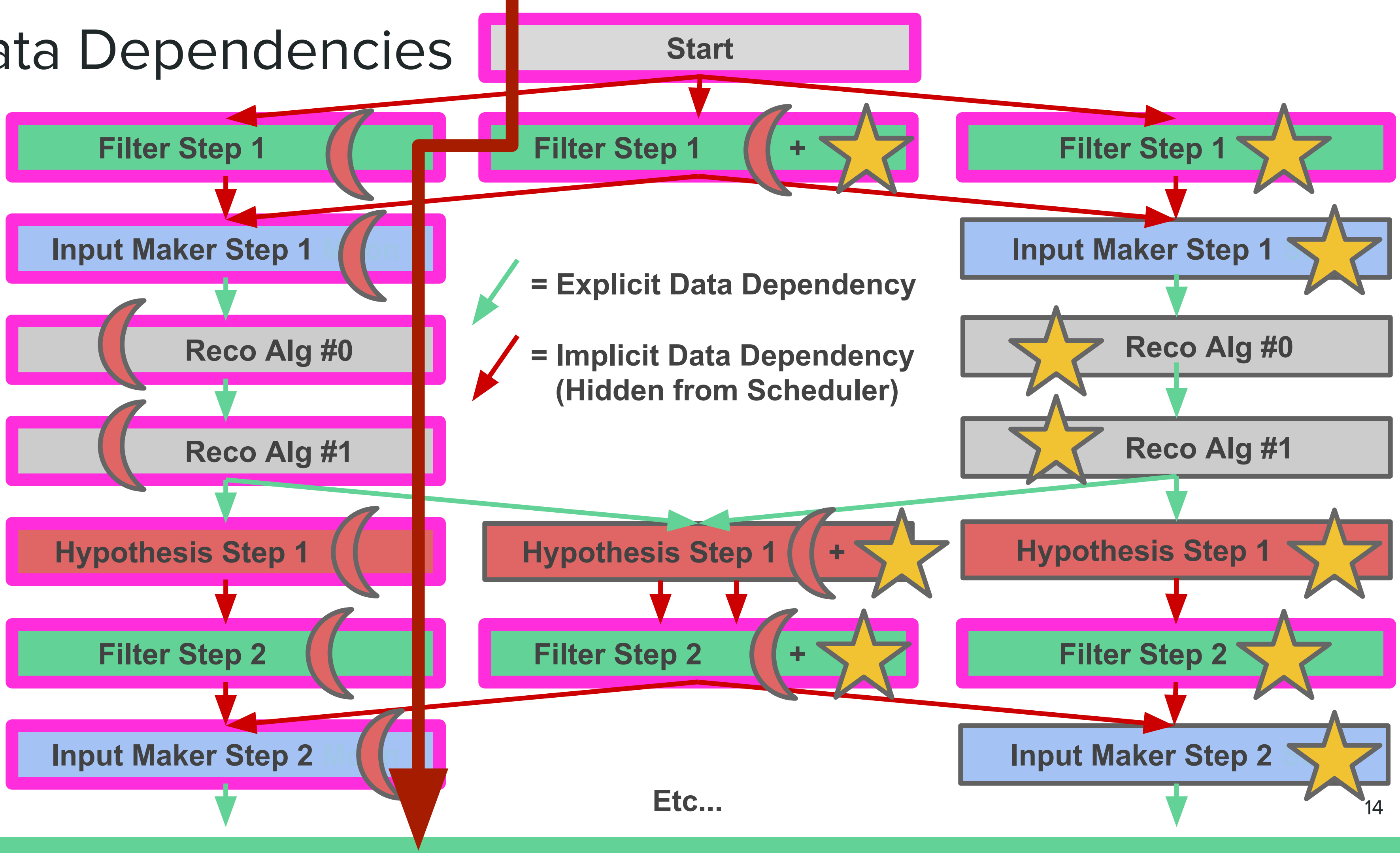
AthenaMT Data Dependencies

- Suppose three **types of selection**:  ,  and  +  . Each **Chain** will follow **one of these three paths**, with the chain's configuration controlling object quality & object size requirements.
- We build a **dependency graph** of the algorithms required to perform the reconstruction. Like in the current system, it is split into different **steps**.
- Three classes of algorithm are used to **control the execution**.
  - **Filter Algorithm** **Always runs** at the start of each step. Responsible for implementing **Early Rejection**. Returns a boolean **Filter Decision** to the **Gaudi MT Scheduler**.
  - **Input Maker Algorithm** Provides concrete starting point for reconstruction algorithms. Responsible for restricting reconstruction to **Regions of Interest.**
  - **Hypothesis Algorithm** Executes **hypothesis testing** for all active **Chains**. Provides input to next Step's Filter(s).

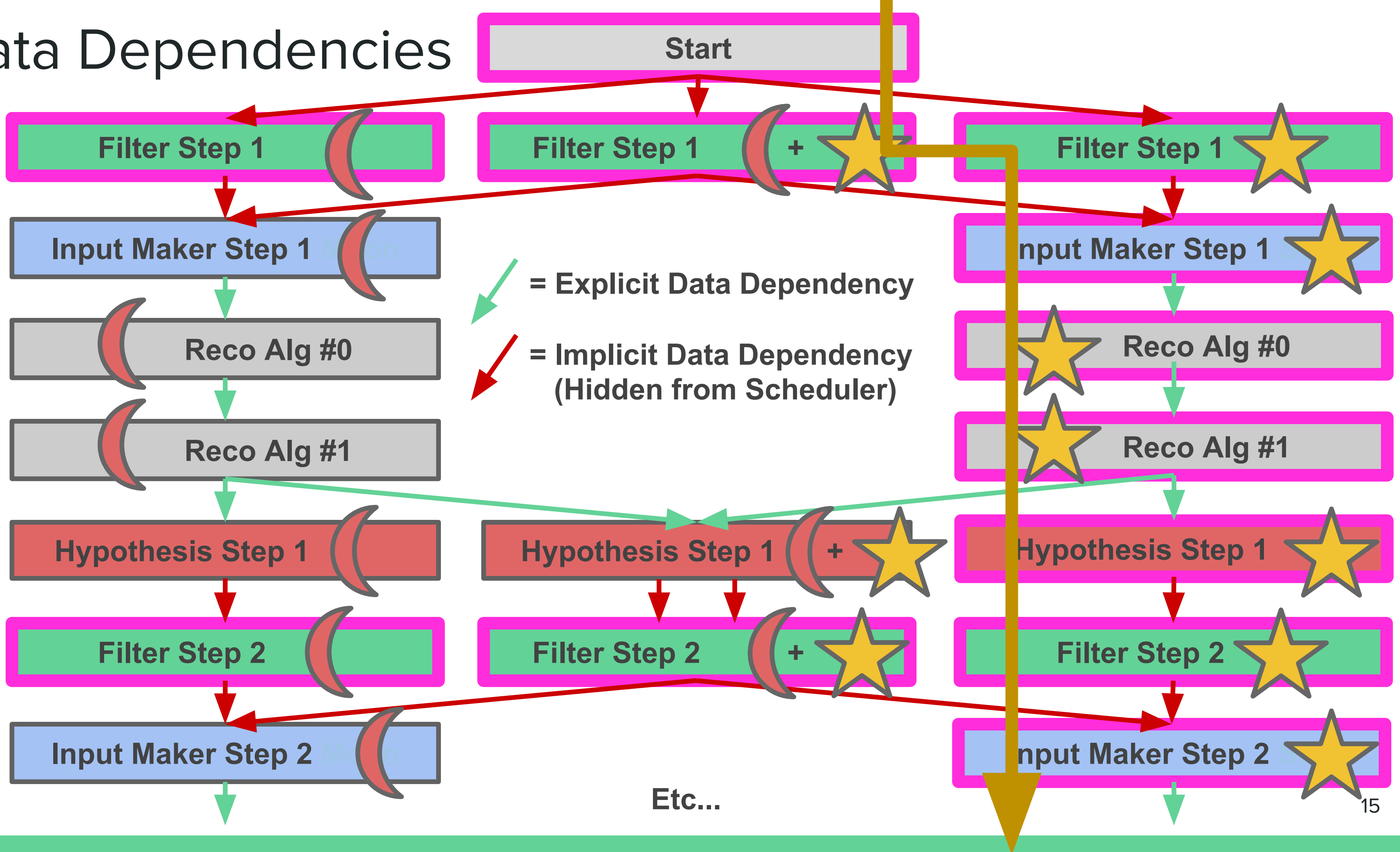
Data Dependencies



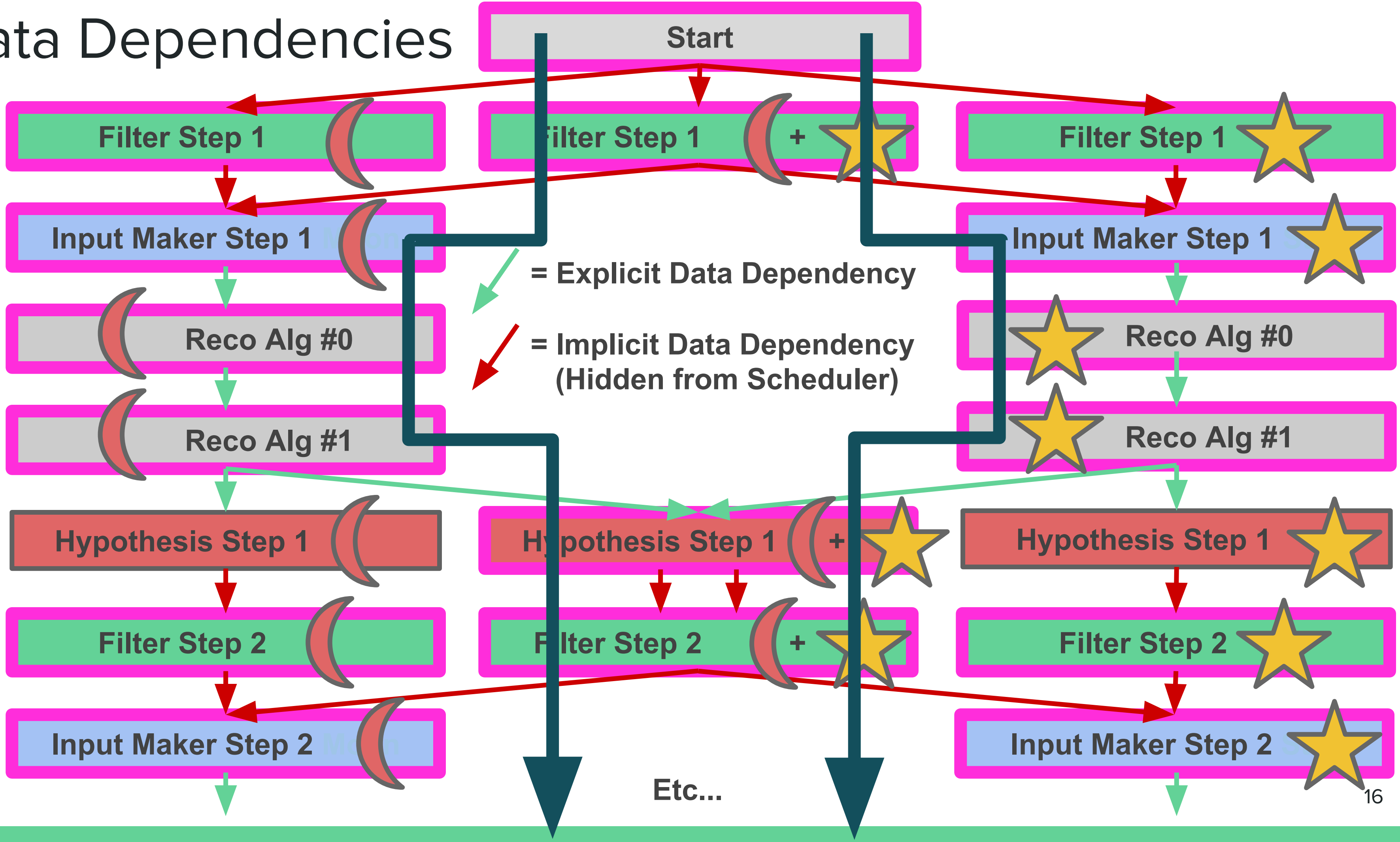
Data Dependencies



Data Dependencies



Data Dependencies



Control Flow

- The **Data Dependency** graph on the previous slide is **not enough** on its own.
  - We need a mechanism to **stop** **Filter Step 2** from running **before** **Hypothesis Step 1** has **executed and returned**.
- Introduce a second **Control Flow** graph, built from two types of **node (Sequencers)**.

OR

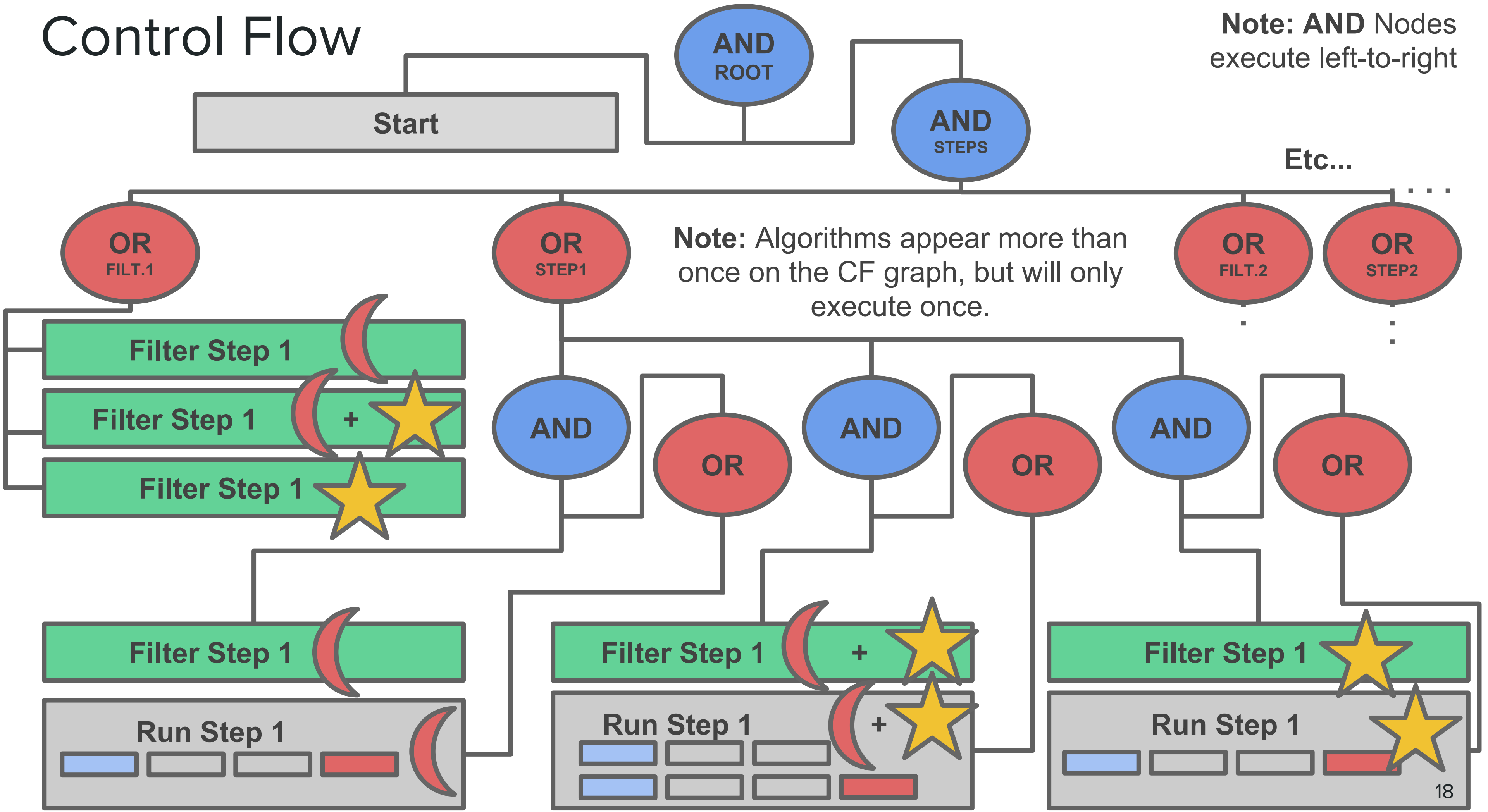
- The **OR** node **cannot exit early**. It will schedule **all of its children to execute** in parallel, and return **the logical OR of its children's** filter decisions upon completion.

AND

- The **AND** node **can exit early**. It will schedule its **children to run sequentially**, one after the other. Should a child return **False**, the sequencer will **halt execution** and return **False** to its parent. Otherwise, it will return the filter decision of its final child.

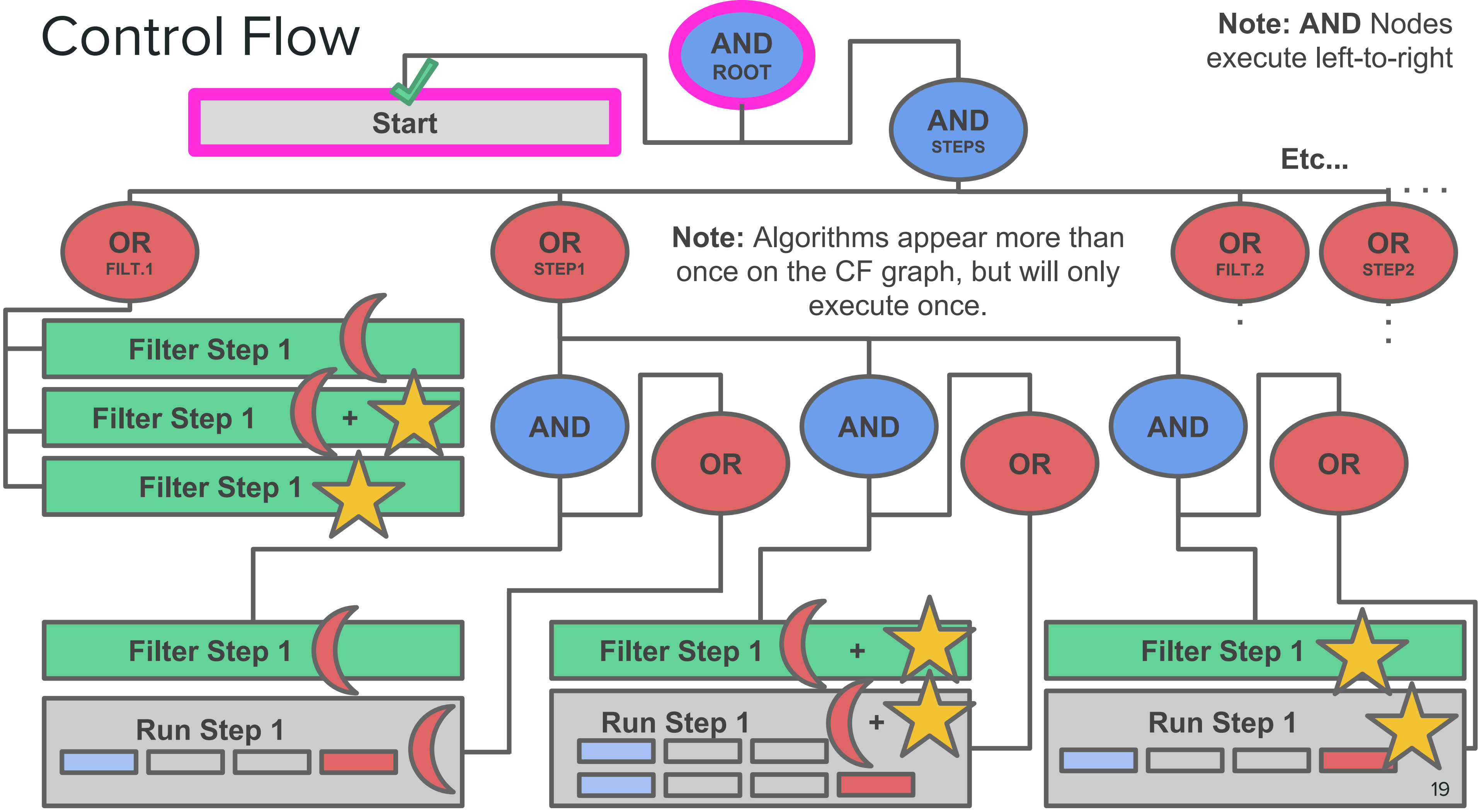
All algorithms must be "unlocked" in the Control Flow graph before they can be Scheduled to run.

Control Flow



Control Flow

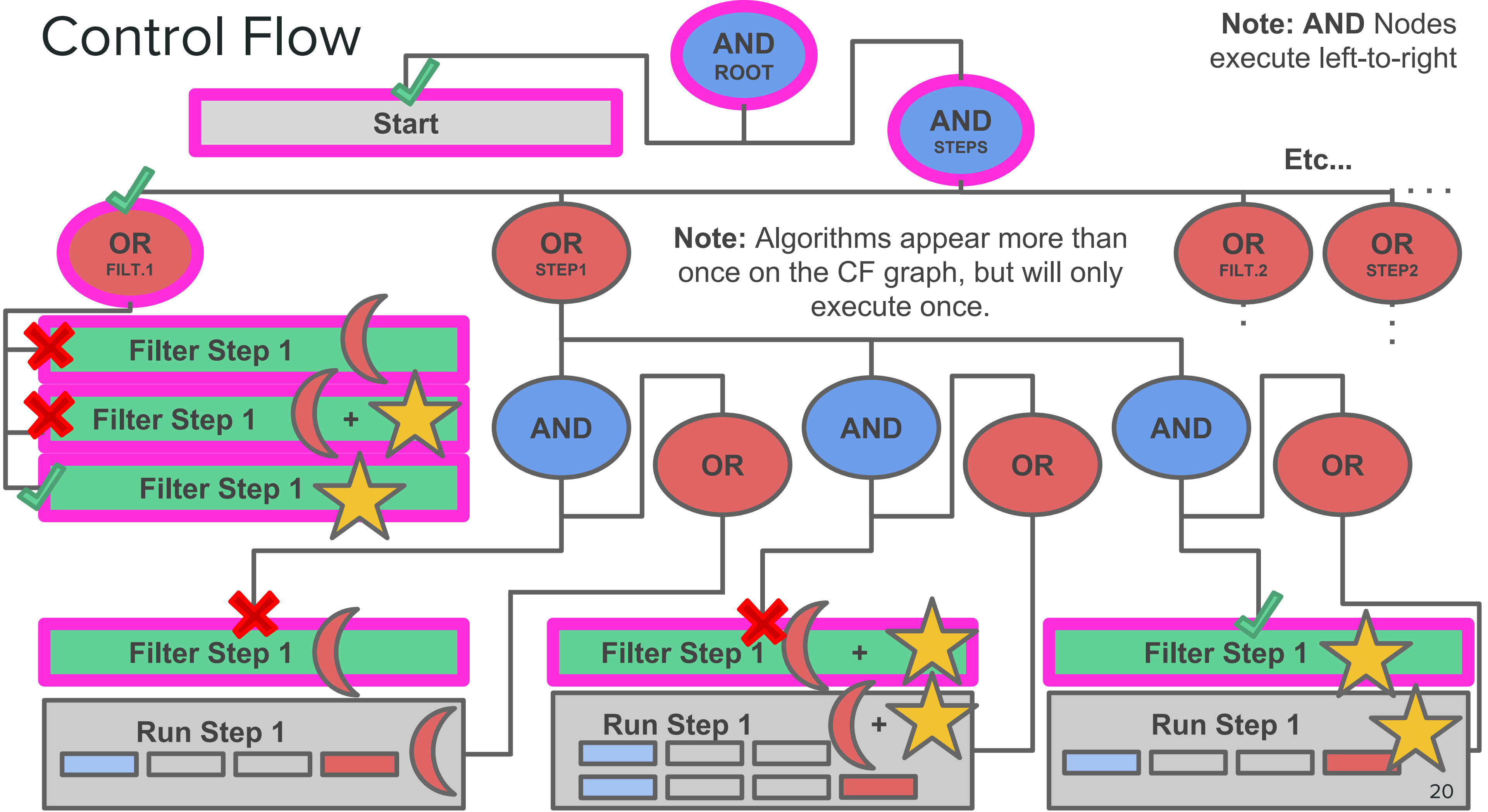
Note: AND Nodes execute left-to-right



Note: Algorithms appear more than once on the CF graph, but will only execute once.

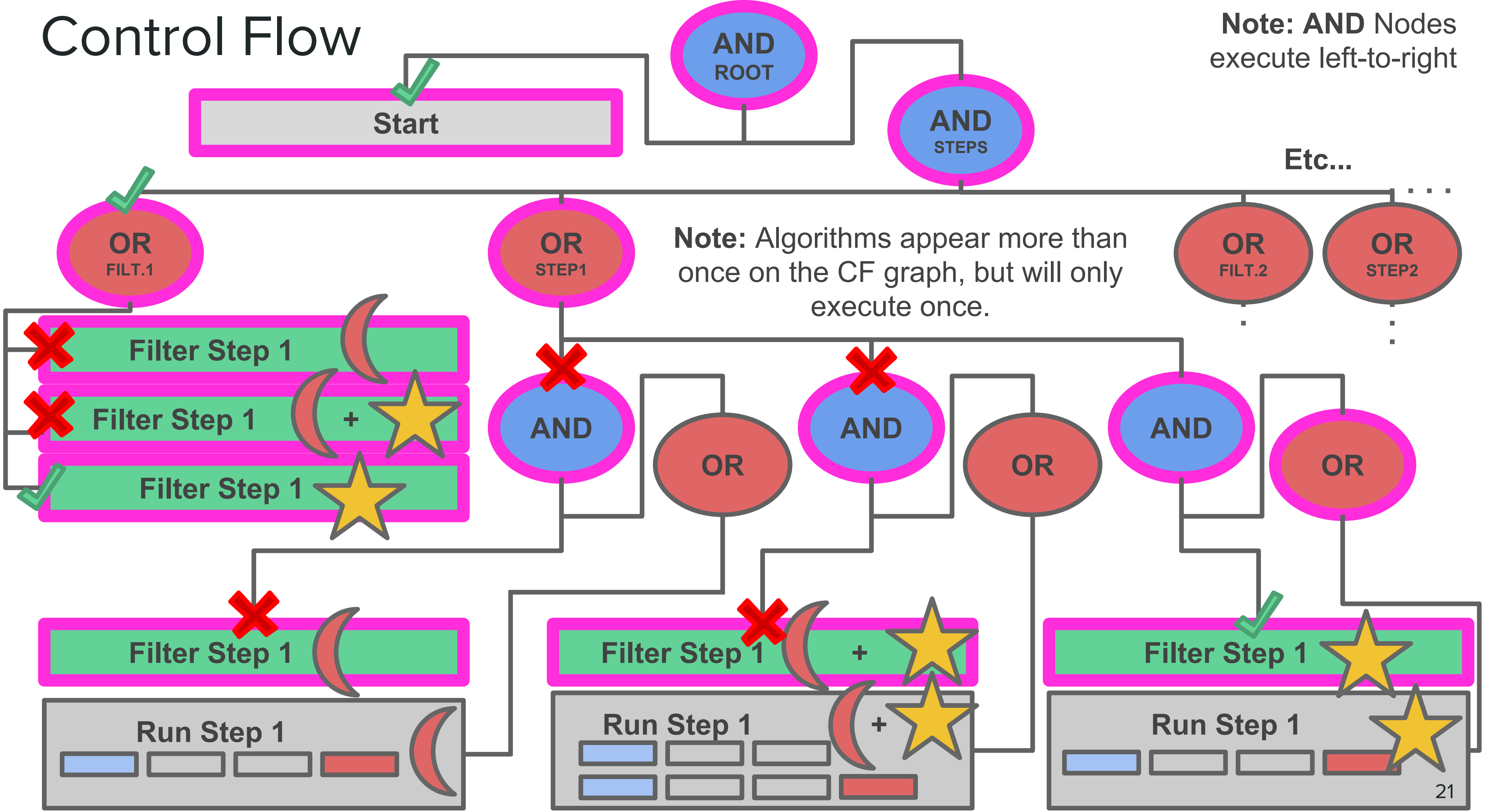
Control Flow

Note: AND Nodes execute left-to-right



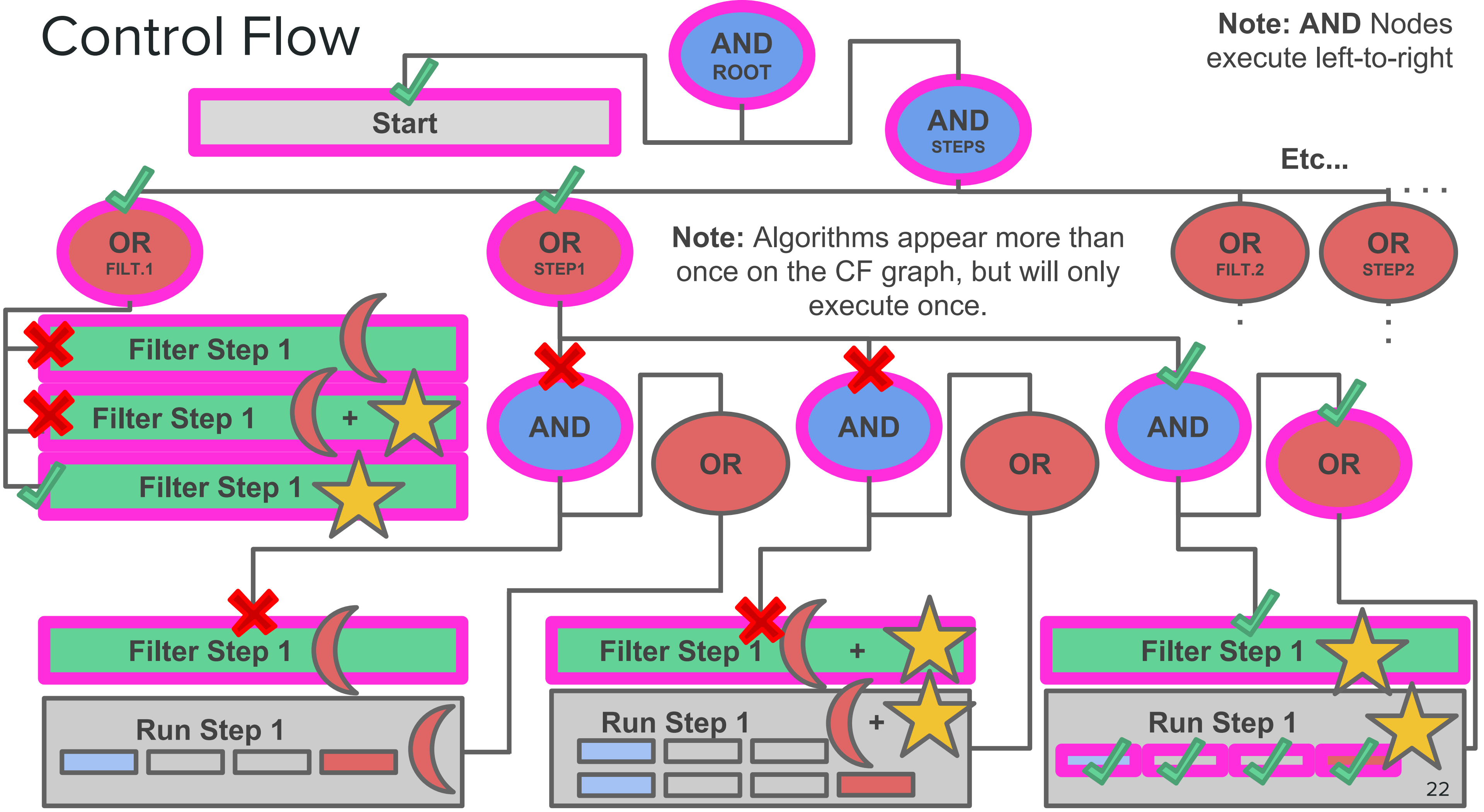
Note: Algorithms appear more than once on the CF graph, but will only execute once.

Control Flow



Control Flow

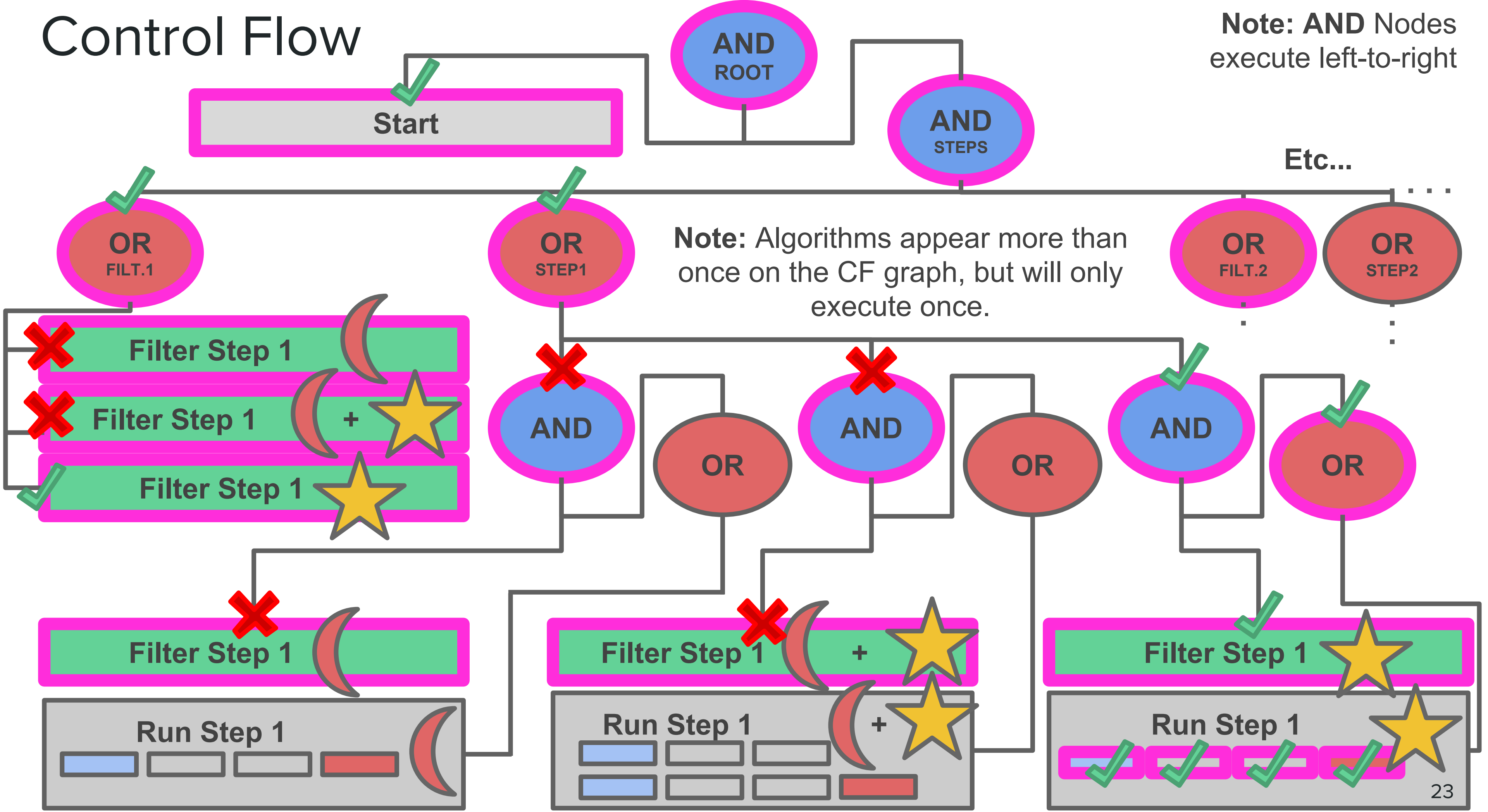
Note: AND Nodes execute left-to-right



Note: Algorithms appear more than once on the CF graph, but will only execute once.

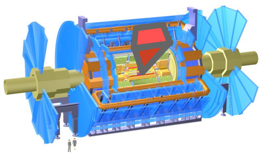
Control Flow

Note: AND Nodes execute left-to-right

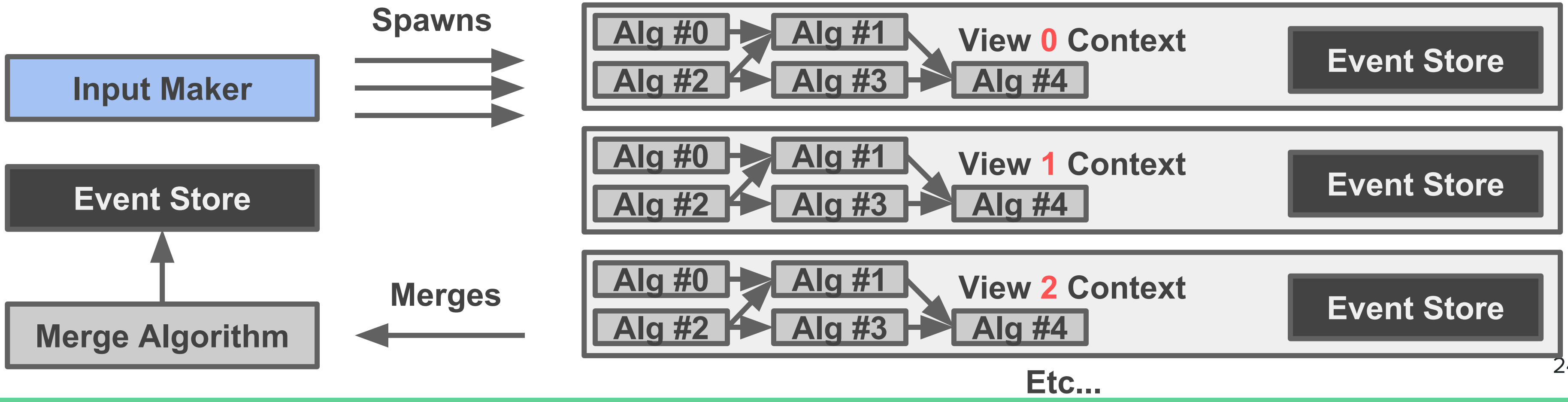


Note: Algorithms appear more than once on the CF graph, but will only execute once.

Regional Reconstruction: Event Views



- **Gaudi Hive** will allow **each algorithm** to execute **at most once per event**.
- But **Regional Reconstruction** requires algorithms to run **once per Region of Interest**.
- **ATLAS** Extension: **Event Views**.
  - **Spawn** one **Event View** per **Region of Interest**, Schedule algorithms **per View**.
  - **Event View** Implements the **Event Store** interface.
  - Completely **transparent** to the **algorithm**. It does not know it's in a view. (...But it can find out)
  - On **completion**, **merge** back to a single collection within the full event context.



Wrapping Up

- **AthenaMT** project will allow for **greater scalability** to the reality of a multi-core world where **memory per core comes at a premium**.
- The two key principles of trigger processing: **Early Rejection** and **Regional Reconstruction** are implementable in **native Gaudi Hive** using a combination of **Data Dependencies**, **Control Flow** and ATLAS' **Event View** extension.
- Will provide greater **unification** of the framework by **removing Trigger-specific steering** and **wrappers**.
- Working on implementation of **framework** and **physics selections** for use in LHC Run 3 in **2021**.

Note: In this talk: squares are jets, moons are electrons & stars are muons. The size of the shape is a proxy for its p_T

Backup

Control Flow & Data Dependencies - In Words

- In this design, all Filter algorithms run first in a **Step**.
  - Check if any **Chains** which utilise the filter are still active and return **True** if so.
  - If **all Filters** in a **Step** return **False** the parent **OR** node will also return **False**: implements **Early Rejection**.
- Reconstruction algs are **unlocked** by the Filters which still have active **Chains**.
  - Algorithms can be **unlocked by multiple** Filters, they will still only run once.
  - **Input Maker** algorithms have no explicit Input Data Dependencies, they will be scheduled to execute first when a **Step** is unlocked.
  - **Reconstruction Algorithms** consume the explicit outputs of the **Input Maker**.
  - The **Hypothesis Algorithm** is the terminal **Data Dependency**. It tests the Hypothesis of each active **Chain** against the reconstructed objects.
- Once **all unlocked** **Hypothesis Algorithms** return, the next **Stage** is unlocked.
  - All **Filter** algorithms read in the previous stage's **Hypothesis** and checks if any **Chains** are still active.

And so on...