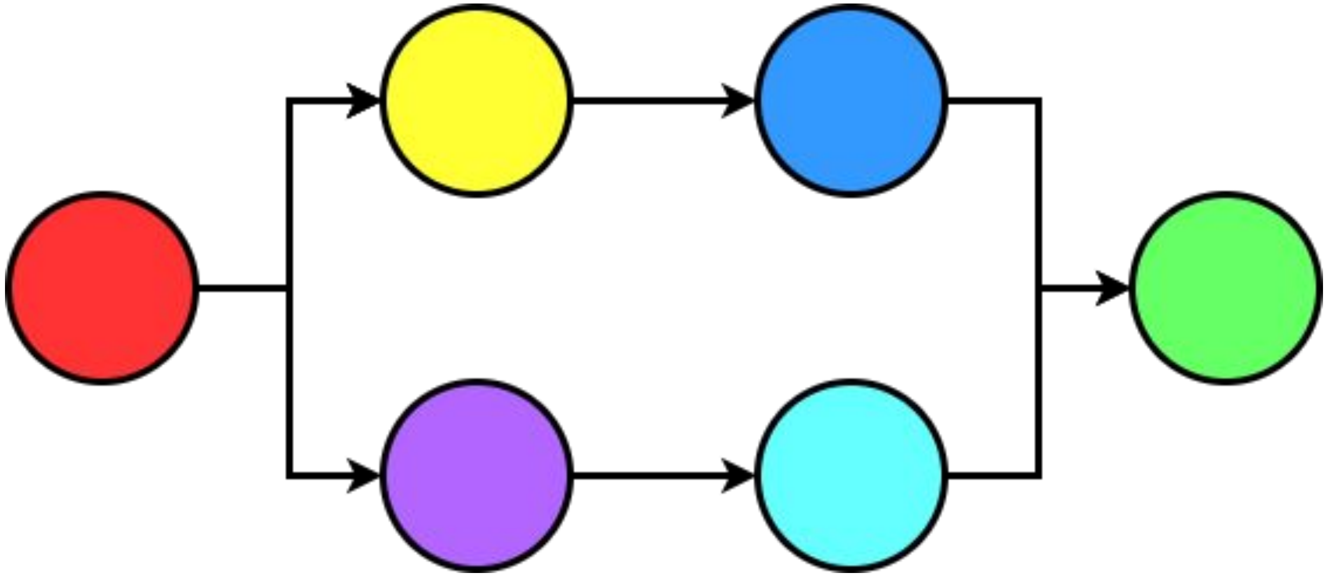
A Fault Tolerance mechanism for Hybrid Scientific Workflows

Alberto Mulone Doriana Medić Marco Aldinucci

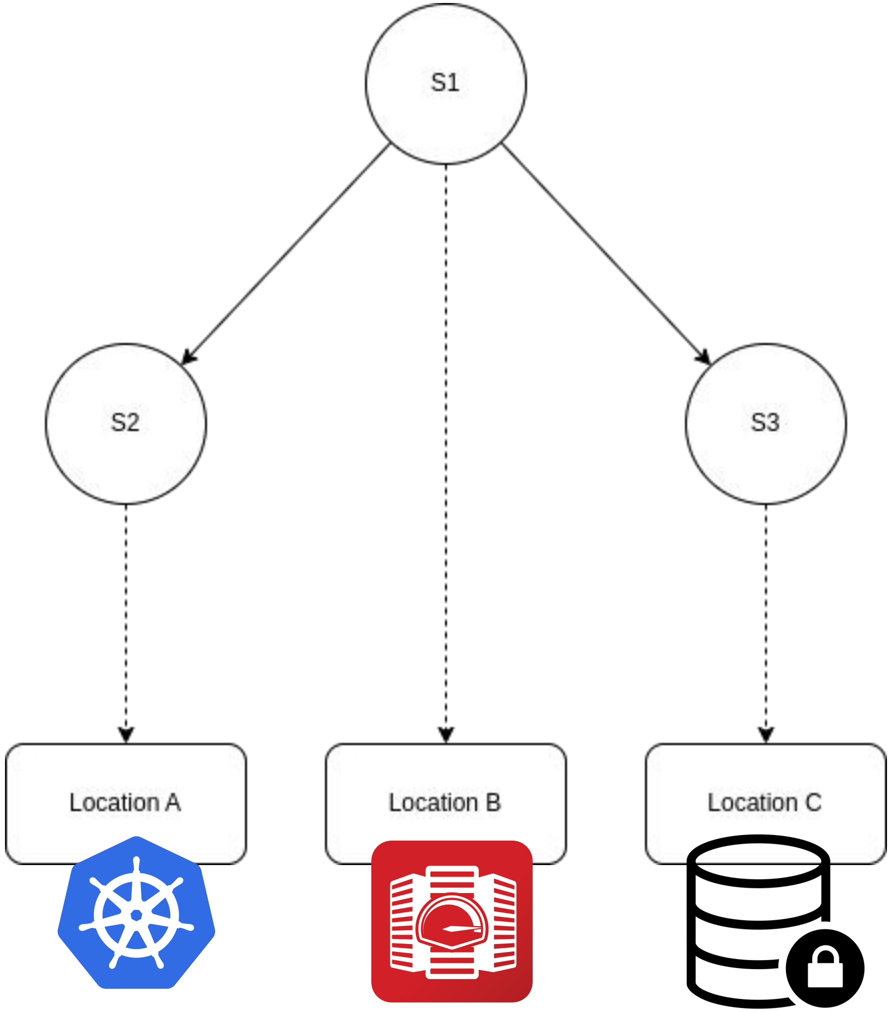
Alberto Mulone, PhD student, University of Turin
HiPES - EuroPar, Madrid, 26 August 2024

Workflow

A workflow is an abstraction that models a complex and modular working process as a set of steps and their inter-dependencies.

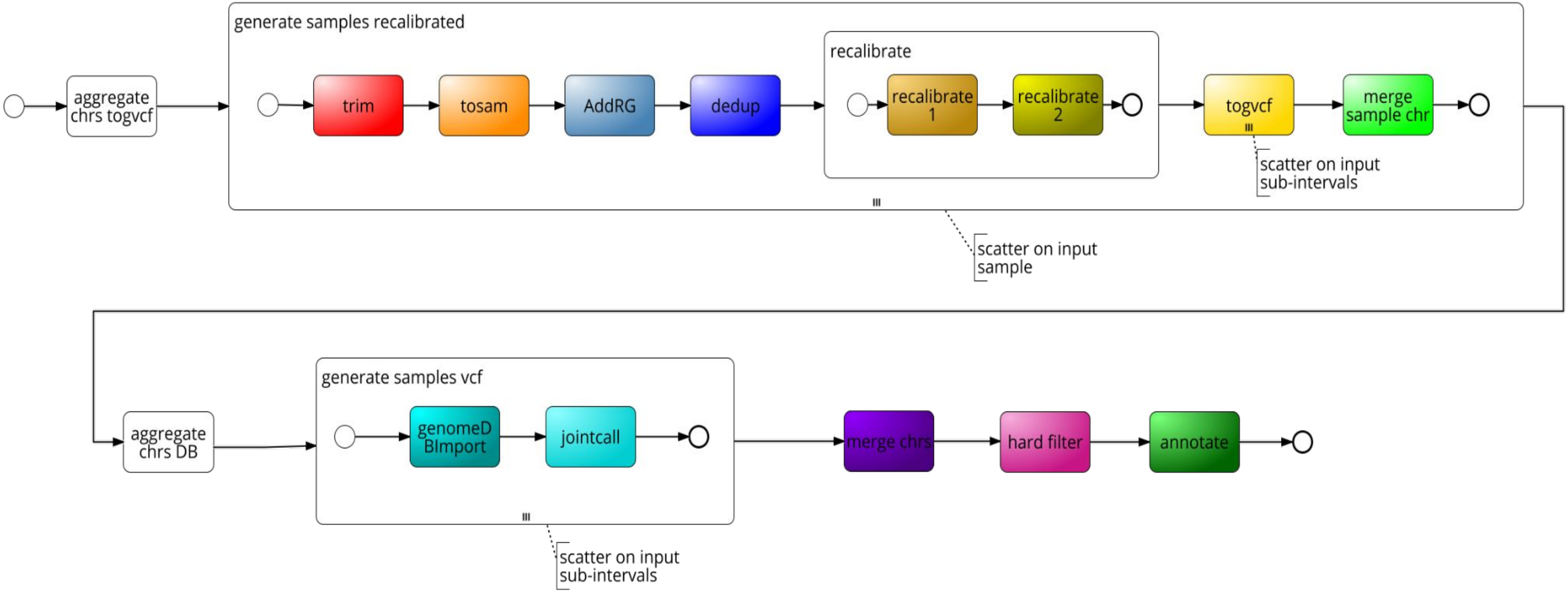


Hybrid workflows



A hybrid workflow is a workflow whose steps can span multiple, **heterogeneous**, and **independent** computing infrastructures.

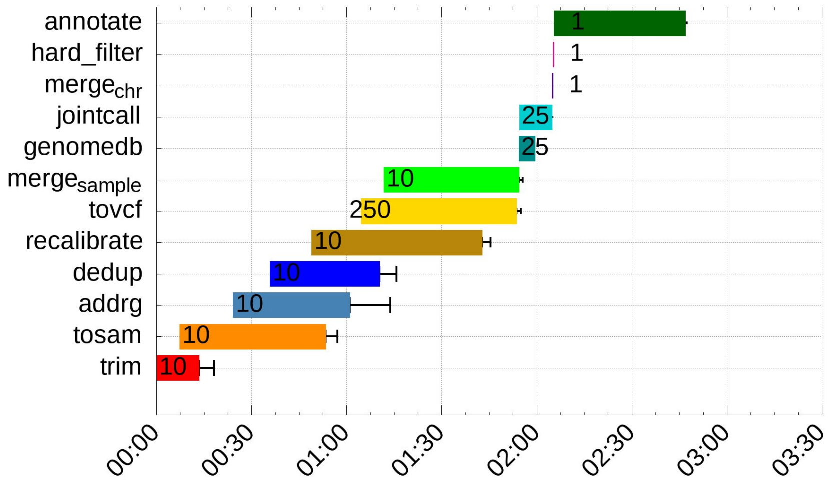
Use case: Variant Calling Pipeline

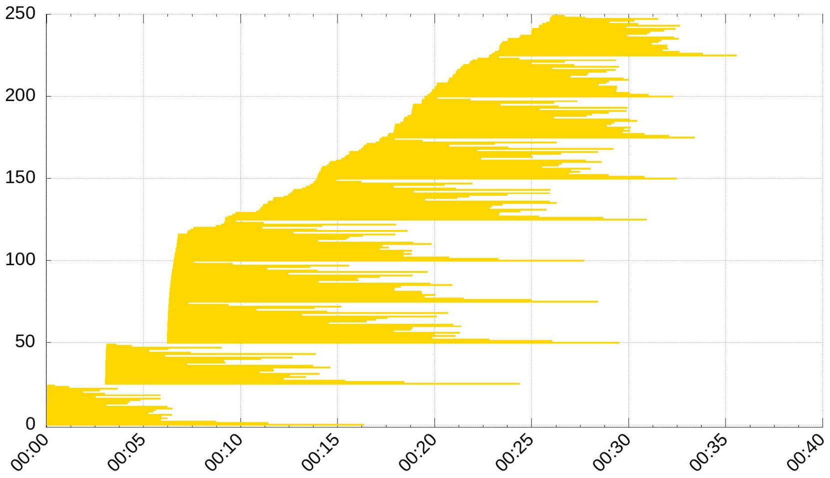


A. Mulone, S. Awad, D. Chiarugi and M. Aldinucci,
"**Porting the Variant Calling Pipeline for NGS data in cloud-HPC environment,**"
2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC),
Torino, Italy, 2023, pp. 1858-1863, doi: 10.1109/COMPSAC57700.2023.00288.

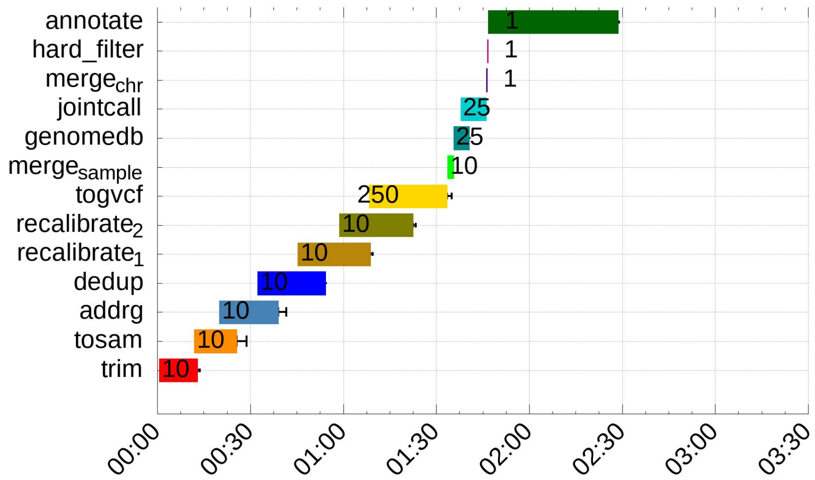
Use case: Variant Calling Pipeline

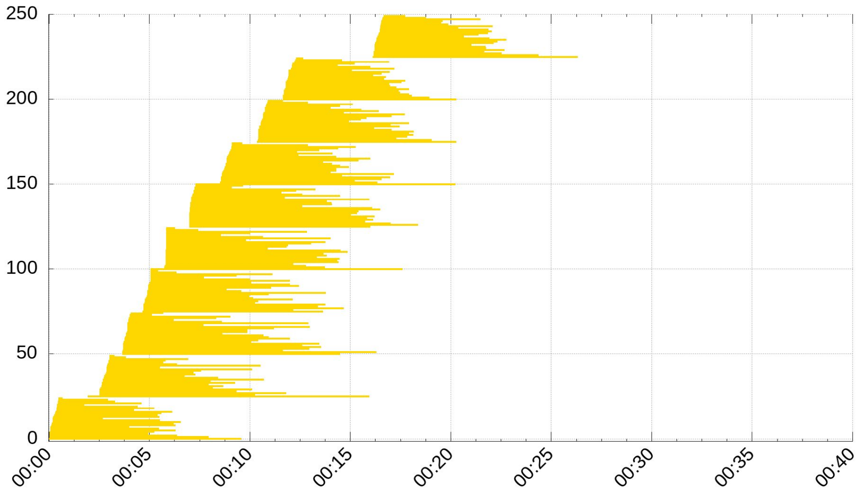
VM cloud
(96 cores)



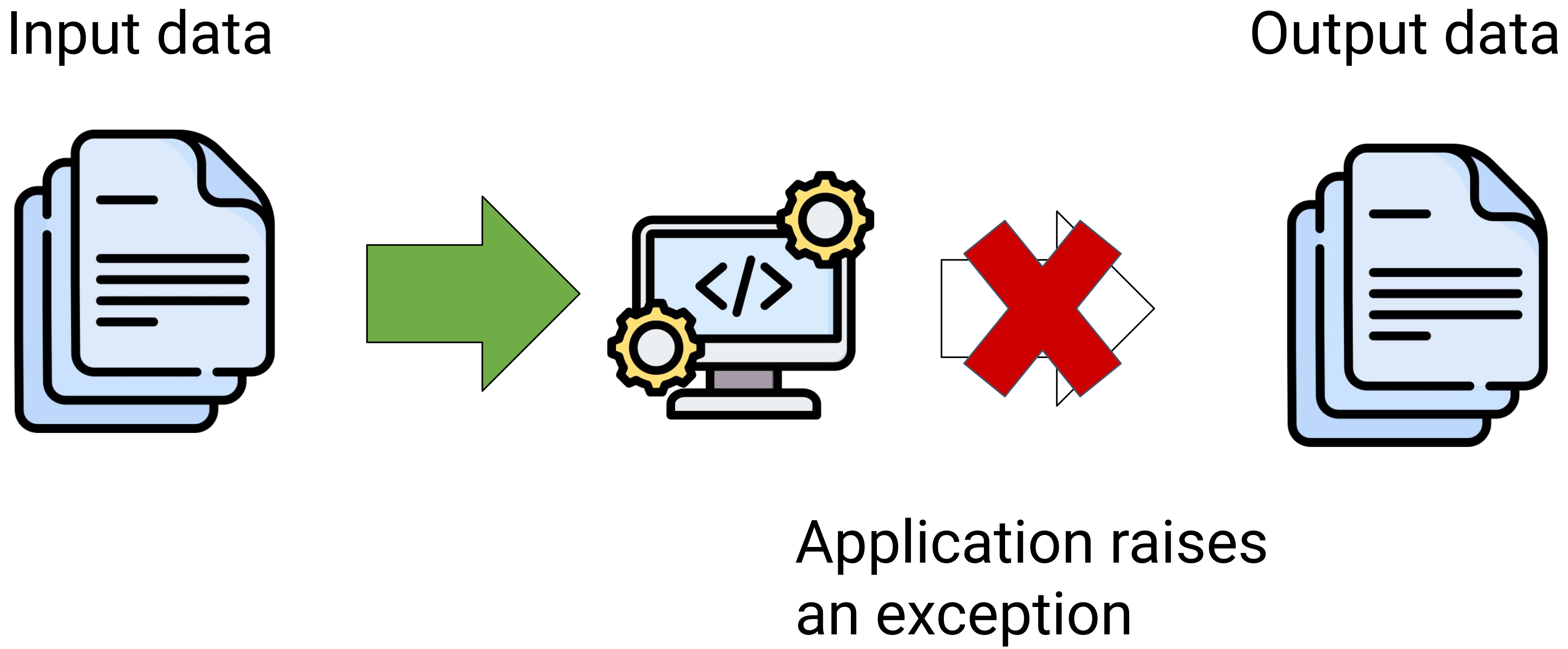


Hybrid
execution
(cloud+HPC)





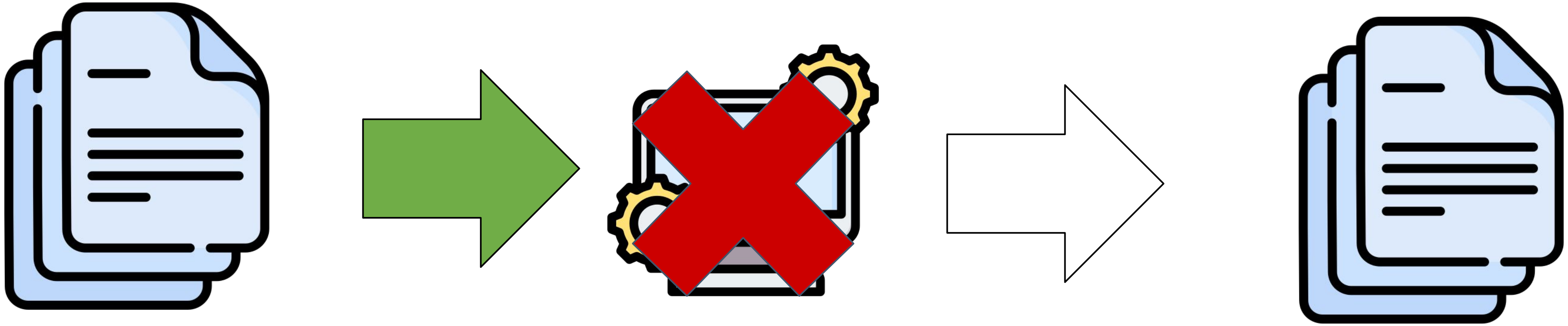
Failures: soft error



Failures: fail-stop error

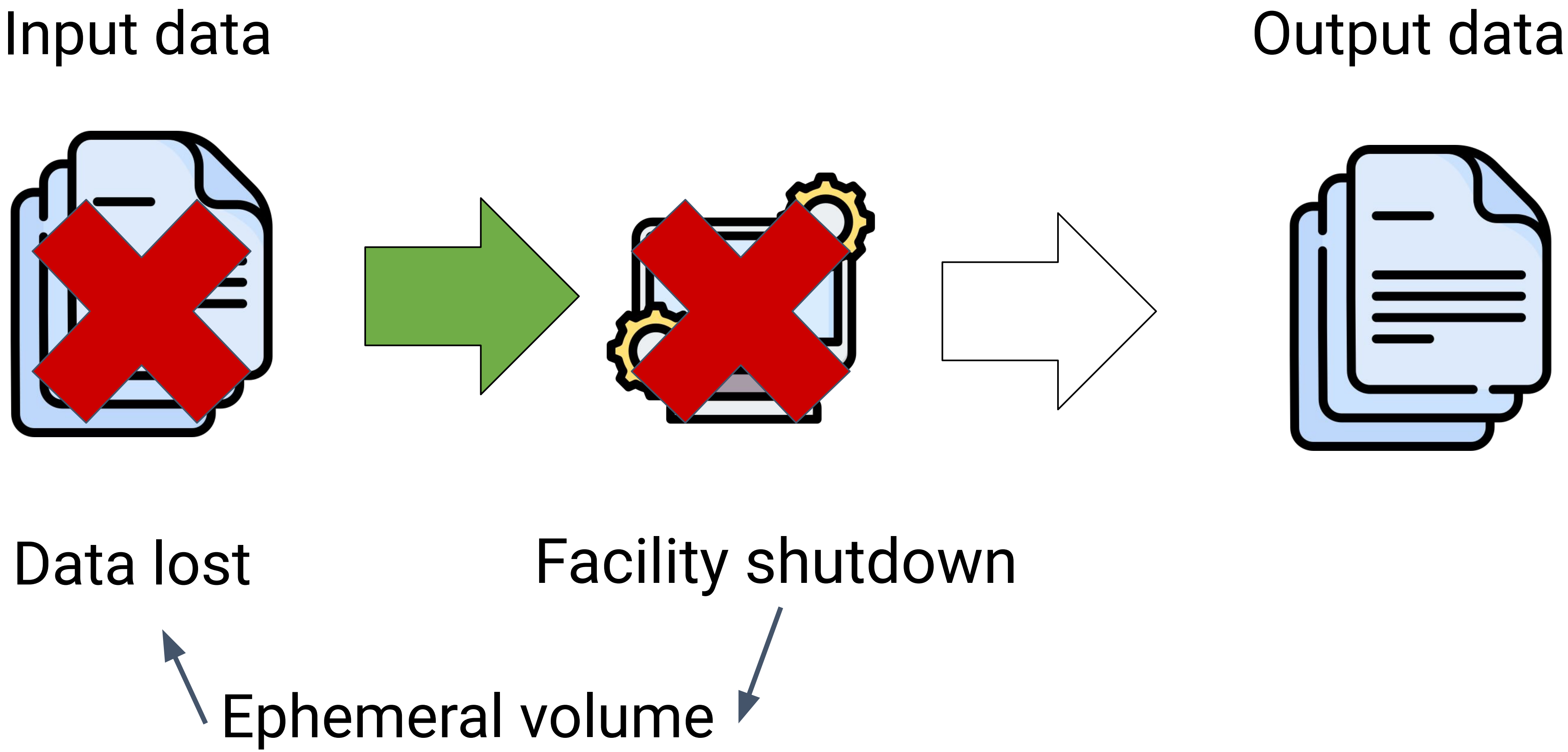
Input data

Output data



Facility shutdown

Failures: fail-stop error



Fault tolerance

A workflow manager is fault tolerant when it can continue the execution correctly even if a failure occurs.

Our contribution

{ if input data are available, then **retry** failed step
otherwise **rollback** of steps which output data are lost

Recovery of the failure is delegated to a new sub-workflow. This sub-workflow is called recovery-workflow.

A synchronization mechanism is necessary to manage dependencies across concurrent recovery-workflows.

Formalization: grammar

Workflow syntax

$$W ::= \langle l,\ D_l,\ t_d \rangle \parallel (W_1,\ W_2)$$

$$t ::= \mu \;/\!/\; t_1.t_2 \;/\!/\; (t_1 \mid t_2) \;/\!/\; (t_1 + t_2) \;/\!/\; \triangleright t \;/\!/\; 0$$

$$\mu ::= exec(s, I,\ O) \;/\!/\; tran(v,\ l_2) \;/\!/\; \underline{tran(v,\ l_1)} \;/\!/\; rec(x)$$

$$v ::= d \;/\!/\; m_s \;/\!/\; m_{d,l} \;/\!/\; ok_{ms} \;/\!/\; ok_d \;/\!/\; err\ (x)$$

$$x ::= s \;/\!/\; D,\ l \;/\!/\; d,\ l \;/\!/\; m_s,\ l$$

Formalization: grammar

Workflow syntax

$$W ::= \langle l, D_l, t_d \rangle \parallel (W_1, W_2)$$

Formalization: grammar

Workflow syntax

$$\mathbf{W ::= \langle l,\ D_l,\ t_d \rangle \parallel (W_1,\ W_2)}$$

where D_l is

$$D_l = \{\ t_W,\ M(s_i),\ d,\ m_{si}\ \}$$

- t_W represents all traces in the workflow
- $M(s_i)$ is the mapping steps-locations
- d is the dataset
- m_{si} are the messages that driver sends to locations

Formalization: grammar

Workflow syntax

$$W ::= \langle l, D_l, t_d \rangle \parallel (W_1, W_2)$$

$$\mathbf{t ::= \mu \parallel t_1.t_2 \parallel (t_1 \mid t_2) \parallel (t_1 + t_2) \parallel {}^{\triangleright} t \parallel 0}$$

Formalization: grammar

Workflow syntax

$$W ::= \langle l,\ D_l,\ t_d \rangle \ \| \ (W_1,\ W_2)$$

$$\mathbf{t ::= \mu \ \| \ t_1.t_2 \ \| \ (t_1 \mid t_2) \ \| \ (t_1 + t_2) \ \| \ {}^{\triangleright}\, t \ \| \ 0}$$

- Actions: $\boldsymbol{\mu}$
- Operators: “**.**”, “|”, “**+**”
- Empty trace **0**
- Pointer denoting which is the current execution of the trace t: $^{\triangleright}$ **t**

Formalization: grammar

Workflow syntax

$$W ::= \langle l,\ D_l,\ t_d \rangle \ \| \ (W_1,\ W_2)$$

$$t ::= \mu \ \| \ t_1.t_2 \ \| \ (t_1 \mid t_2) \ \| \ (t_1 + t_2) \ \| \ \triangleright t \ \| \ 0$$

$$\boldsymbol{\mu ::= \mathrm{exec}(s, I,\ O) \ \| \ \mathrm{tran}(v,\ l_2) \ \| \ \underline{\mathrm{tran}}(v,\ l_1) \ \| \ \mathrm{rec}(x)}$$

Formalization: grammar

Workflow syntax

$W ::= \langle l,\ D_l,\ t_d \rangle \parallel (W_1,\ W_2)$

$t ::= \mu \parallel t_1 . t_2 \parallel (t_1 \mid t_2) \parallel (t_1 + t_2) \parallel {}^{\triangleright} t \parallel 0$

$\mu ::= \mathtt{exec}(s,\ I,\ O) \parallel \mathtt{tran}(v,\ l_2) \parallel \underline{\mathtt{tran}}(v,\ l_1) \parallel \mathtt{rec}(x)$

$\mathbf{v ::= d \parallel m_s \parallel m_{d,l} \parallel ok_{ms} \parallel ok_d \parallel err\ (x)}$

Formalization: grammar

Workflow syntax

$$W ::= \langle l,\ D_l,\ t_d \rangle \parallel (W_1,\ W_2)$$

$$t ::= \mu \parallel t_1.t_2 \parallel (t_1 \mid t_2) \parallel (t_1 + t_2) \parallel {}^{\triangleright}t \parallel 0$$

$$\mu ::= \mathrm{exec}(s, I,\ O) \parallel \mathrm{tran}(v,\ l_2) \parallel \underline{\mathrm{tran}}(v,\ l_1) \parallel \mathrm{rec}(x)$$

$$v ::= d \parallel m_s \parallel m_{d,l} \parallel ok_{ms} \parallel ok_d \parallel err\ (x)$$

$$\mathbf{x ::= s \parallel D,\ l \parallel d,\ l \parallel m_s,\ l}$$

Formalization: rule actions

$$(\textsc{Init})\quad \frac{l = \mathcal{M}(s)}{\langle l_d, D_{l_d}, {}^{\triangleright}\texttt{init}(t(s), l).t_d\rangle \mid \langle l, D_l, t\rangle \longrightarrow \langle l_d, D_{l_d}, \texttt{init}(t(s), l).{}^{\triangleright}t_d\rangle \mid \langle l, D_l, t \mid {}^{\triangleright}t(s)\rangle}$$

$$(\textsc{Exec})\quad \frac{\begin{cases} \text{if } I(s) \not\subset D_l \ \wedge\ m_s \in D_l,\ X = \{err(D,l)\} & \text{where } D = \{d \mid d \in I(s) \ \wedge\ d \notin D_l\} \\ \text{if } I(s) \subset D_l\ X = \{O(s), M_d\} & \text{where } M_d = \{m_{d,l} \mid d \in O(s)\} \ \vee\ X = \{err(s)\} \end{cases}}{\langle l, D_l, {}^{\triangleright}\texttt{exec}(s, I, O).t\rangle \longrightarrow \langle l, (D_l \setminus \{m_s\}) \cup X, \texttt{exec}(s, I, O).{}^{\triangleright}t\rangle}$$

$$(\textsc{Trans})\quad \frac{v \in D_{l_1} \quad \wedge \quad \begin{cases} \text{if } v \in \{d, m_s\},\ \text{then } X = \emptyset \text{ and } V = \{v, ok_v\} \vee V = \{err(v, l_2)\} \\ \text{if } v = \{m_{d,l_1}, ok_d, ok_{m_s}\},\ \text{then } X = \{v\} \text{ and } V = \{v\} \end{cases}}{\begin{array}{c} \langle l_1, D_{l_1}, {}^{\triangleright}\texttt{tran}(v, l_2).t_1\rangle \mid \langle l_2, D_{l_2}, {}^{\triangleright}\underline{\texttt{tran}}(v, l_1).t_2\rangle \longrightarrow \\ \langle l_1, D_{l_1} \setminus X, \texttt{tran}(v, l_2).{}^{\triangleright}t_1\rangle \mid \langle l_2, D_{l_2} \cup V, \underline{\texttt{tran}}(v, l_1).{}^{\triangleright}t_2\rangle \end{array}}$$

$$(\textsc{TransErr})\quad \frac{v = err(x) \quad \wedge \quad err(x) \in D_l \quad \text{where } x = \{s, (D,l), (d,l), (m_s, l)\}}{\begin{array}{c} \langle l, D_l, {}^{\triangleright}\texttt{tran}(v, l_d).t\rangle \mid \langle l_d, D_{l_d}, {}^{\triangleright}\underline{\texttt{tran}}(v, l).t_d\rangle \longrightarrow \\ \langle l, D_l \setminus \{v\}, \texttt{tran}(v, l_d).{}^{\triangleright}t\rangle \mid \langle l_d, D_{l_d} \cup \{v\}, \underline{\texttt{tran}}(v, l).{}^{\triangleright}t_d \mid {}^{\triangleright}\texttt{rec}(x)\rangle \end{array}}$$

$$(\textsc{LostData})\quad \frac{err(d, l) \in D_{l_d} \quad \wedge \quad m_{d_i,l} \in D_{l_d}}{D_{l_d} = D_{l_d} \setminus \{m_{d_i,l}\}}$$

Formalization: recovery rule actions

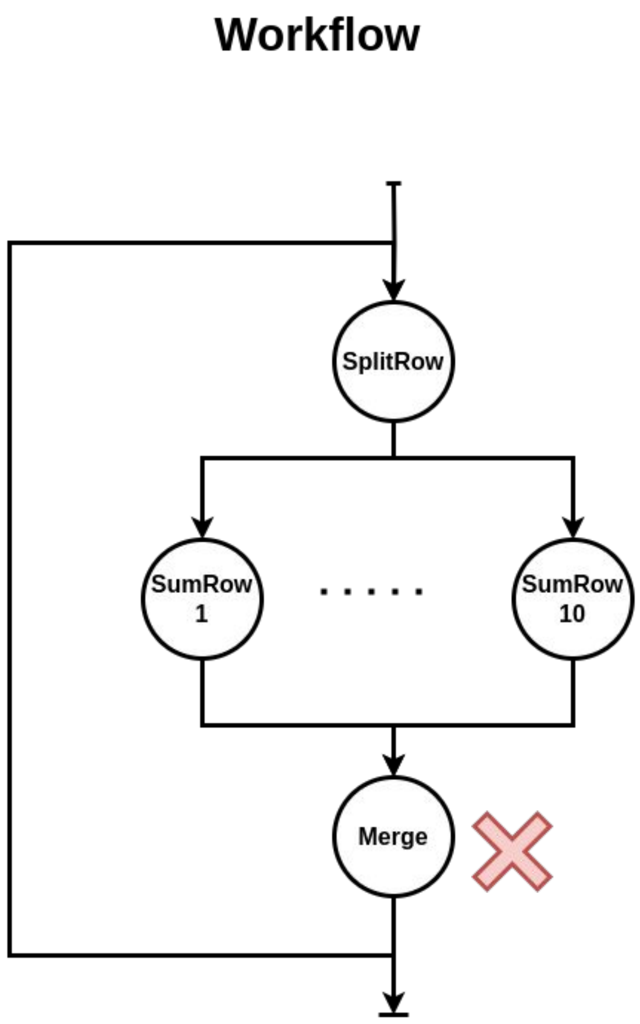
$$(\textsc{Rec}(s)) \quad \frac{err(s) \in D_{l_d}}{\begin{array}{c}\langle l_d, D_{l_d}, t_d \mid {}^{\triangleright}\mathtt{rec}(s)\rangle \mid \langle l, D_l, t(s)\rangle \rightarrow \\ \left\langle l_d, D_{l_d} \setminus \{err(s)\}, t_d \mid \mathtt{rec}(s)^{\triangleright} \mid {}^{\triangleright}t_d(s \setminus t_{d_{I(s)}})\right\rangle \mid \langle l, D_l, \mathbf{0} \mid t_s\rangle\end{array}}$$

$$(\textsc{Rec}(D)) \quad \frac{\begin{array}{c}\forall d_i \in D \text{ such that } err(d_i, l), m_{d_i, l_j} \in D_{l_d} \\ \forall d_h \in D \text{ such that } err(d_h, l) \in D_{l_d} \text{ and } m_{d_h, l'}, d_h \notin D_{l_d} \qquad d_h \in I(s_r)\end{array}}{\begin{array}{c}\langle l_d, D_{l_d}, t_d \mid {}^{\triangleright}\mathtt{rec}(D, l)\rangle \mid \prod_j \left\langle l_j, D_{l_j}, t_j\right\rangle \mid \prod_k \langle l_k, D_{l_k}, t_k\rangle \mid \langle l, D_l, t\rangle \rightarrow \\ \langle l_d, D_{l_d} \setminus \{err(D, l)\}, t_d \mid \mathtt{rec}(D, l)^{\triangleright} \mid {}^{\triangleright}t_d(s) \mid \mathtt{rec}(s_r)\rangle \mid \\ \prod_j \left\langle l_j, D_{l_j}, t_j \mid {}^{\triangleright}\mathtt{tran}(d_i, l)\right\rangle \mid \langle l, \emptyset, \mathbf{0} \mid t(s)\rangle\end{array}}$$

$$(\textsc{Rec}(d)) \quad \frac{err(d, l), m_{d, l'} \in D_{l_d} \quad \wedge \quad l' \in \mathcal{M}(s_1) \quad d \in O(s_1)}{\begin{array}{c}\langle l_d, D_{l_d}, t_d \mid {}^{\triangleright}\mathtt{rec}(d, l)\rangle \mid \langle l', D_{l'}, t'\rangle \mid \langle l, D_l, t\rangle \rightarrow \\ \langle l_d, D_{l_d} \setminus \{err(d, l)\}, t_d \mid \mathtt{rec}(d, l)^{\triangleright}\rangle \mid \langle l', D_{l'}, t' \mid {}^{\triangleright}\mathtt{tran}(d, l)\rangle \mid \\ \langle l, D_l, t \mid \underline{\mathtt{tran}}(d, l').\mathbf{conf}(ok_d, l)\rangle\end{array}}$$

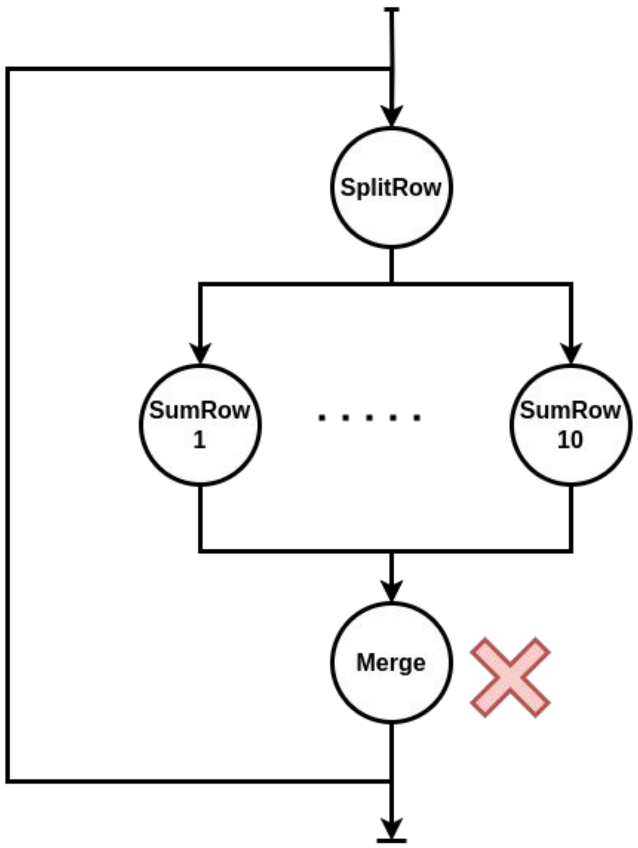
$$(\textsc{Rec}(m_s)) \quad \frac{err(m_s, l) \in D_{l_d}}{\begin{array}{c}\langle l_d, D_{l_d}, t_d \mid {}^{\triangleright}\mathtt{rec}(m_s, l)\rangle \mid \langle l, D_l, t\rangle \rightarrow \\ \langle l_d, D_{l_d} \setminus \{err(m_s, l)\}, t_d \mid \mathtt{rec}(d, l)^{\triangleright}\mathtt{tran}(m_s, l).\underline{\mathbf{conf}}(ok_{m_s}, l)\rangle \mid \\ \langle l, D_l, t \mid \underline{\mathtt{tran}}(m_s, l_d).\mathbf{conf}(ok_{m_s}, l)\rangle\end{array}}$$

Example

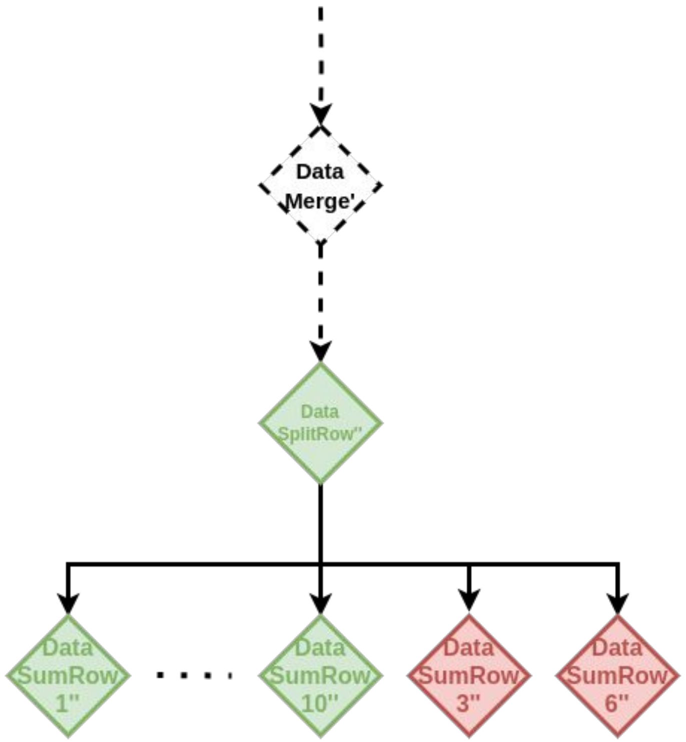


Example

Workflow

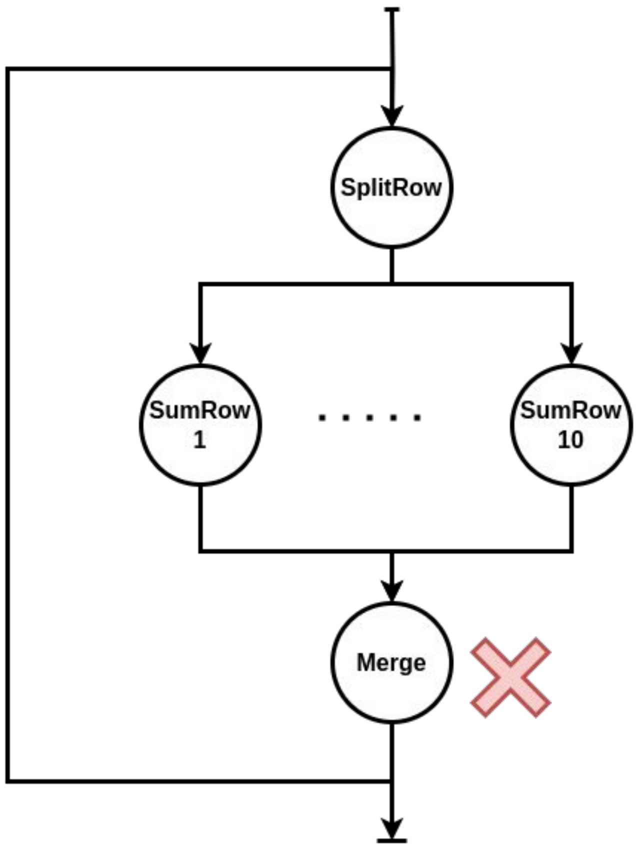


Provenance graph

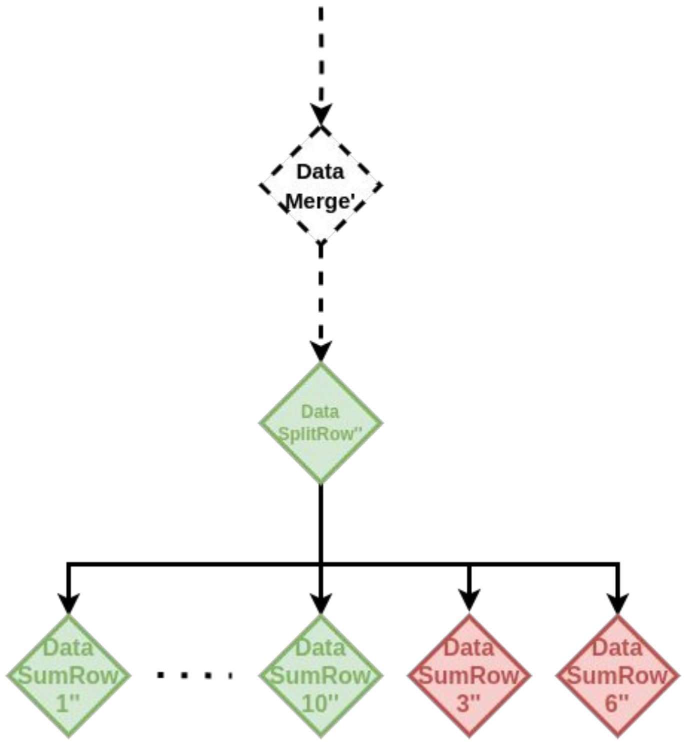


Example

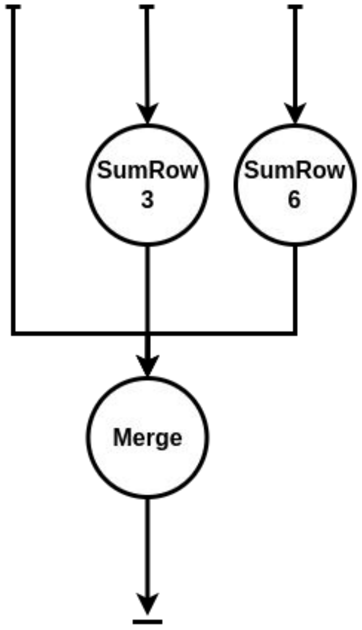
Workflow



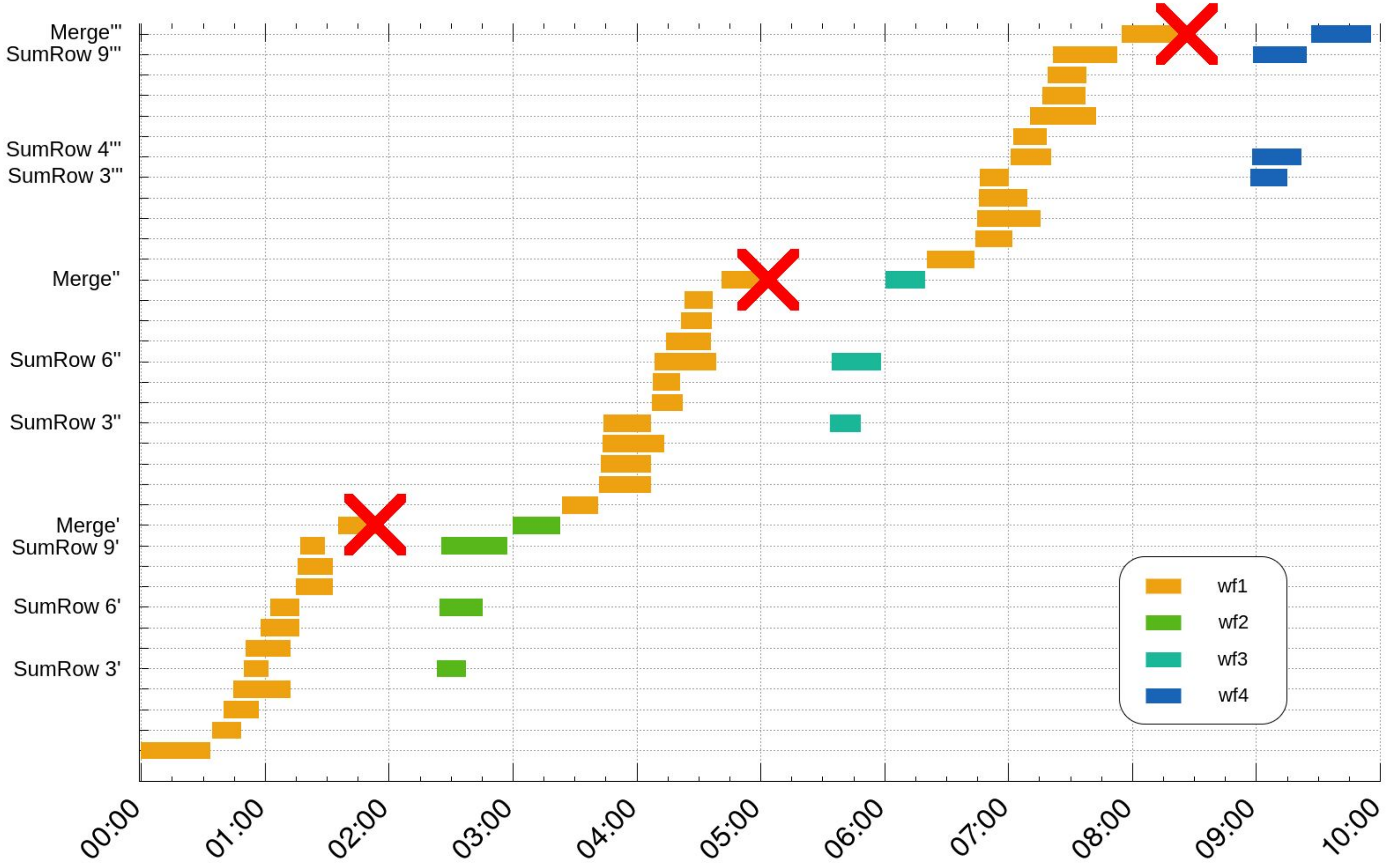
Provenance graph



Recovery-workflow



Experiment



StreamFlow
Towards Cloud-HPC Continuum

- Fault tolerance mechanism implemented in StreamFlow (a hybrid WMS)
- Kubernetes with 4 workers
- Injected fail-stop errors on Merge step

Conclusions

- Mixed different fault tolerance mechanisms
- Introduced a semantics for fault tolerant hybrid workflows
- Implemented the mechanism in StreamFlow, a hybrid workflow system

Future works

- Deep performance study about the implemented mechanism in StreamFlow
- Extend the semantics with workflow loop
- Support non-deterministic workflows