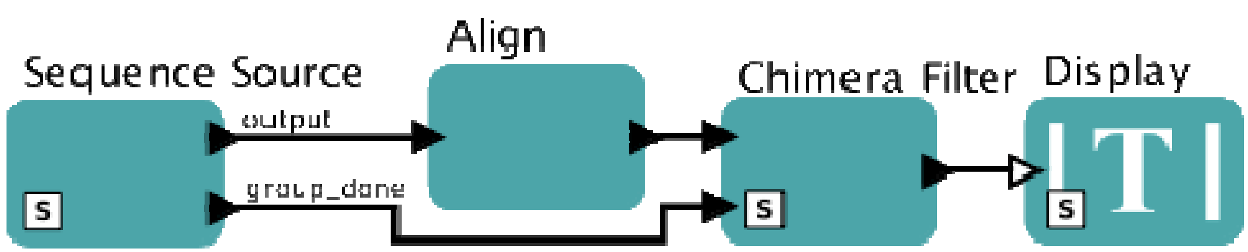
Improving Workflow Fault Tolerance through Provenance-based Recovery

Sven Köhler, Timothy McPhillips, **Sean Riddle**, Daniel Zinn, Bertram Ludäscher

Introduction

- Scientific Workflows
  - Automate scientific pipelines
  - Have long running computations
  - Often contain stateful actors
- Workflow execution can crash because of …
  - Hardware failures
  - Power outages
  - Buggy / malicious actors, …
- Current approach: Start workflow from the beginning



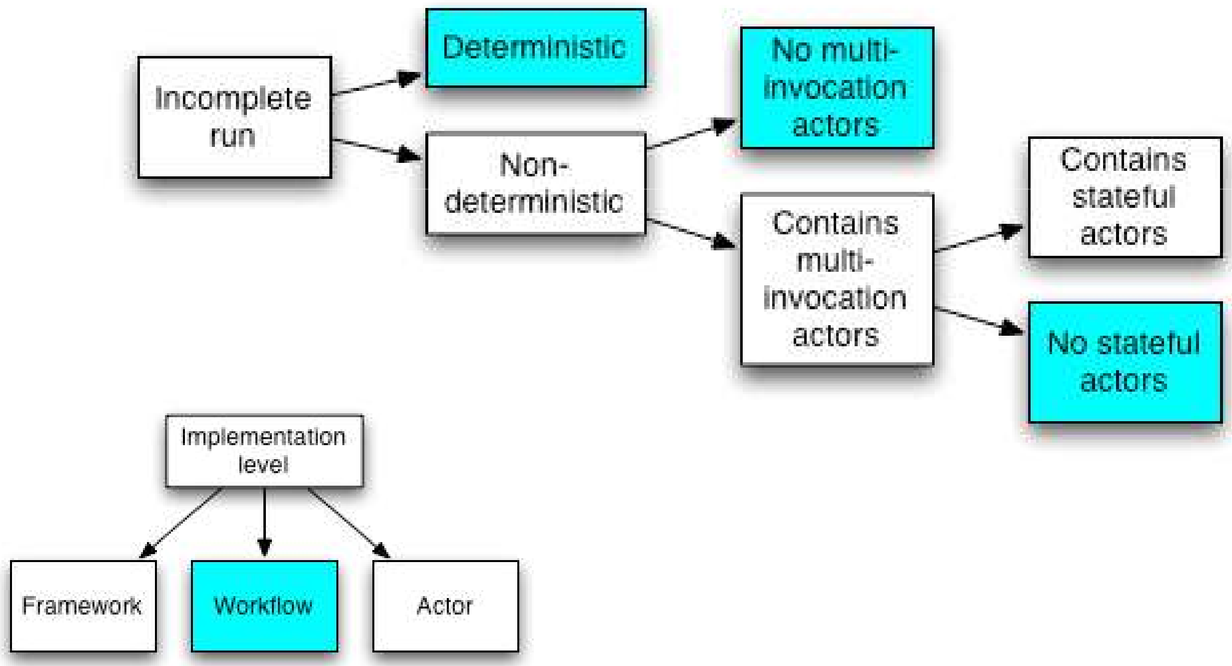
Current Fault Tolerance Solutions …

- Use caching strategies for faster re-execution
  - WATERS memoization [Hartman et al.]
  - “Skip over” strategy [Podhorszki et al.] (CPES)

- Manage actor failures or sub-workflow failures AND their effects
  - Atomicity and provenance support for pipelined scientific workflows [Wang et al.]
  - Ptolemy’s “Backtrack” [Feng et al.]

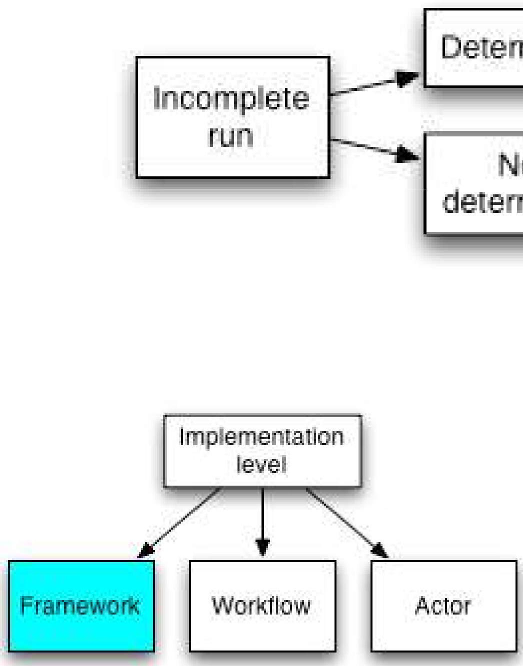
Fault Tolerance Solutions Compared

Contingency actors – Use: Exception handling



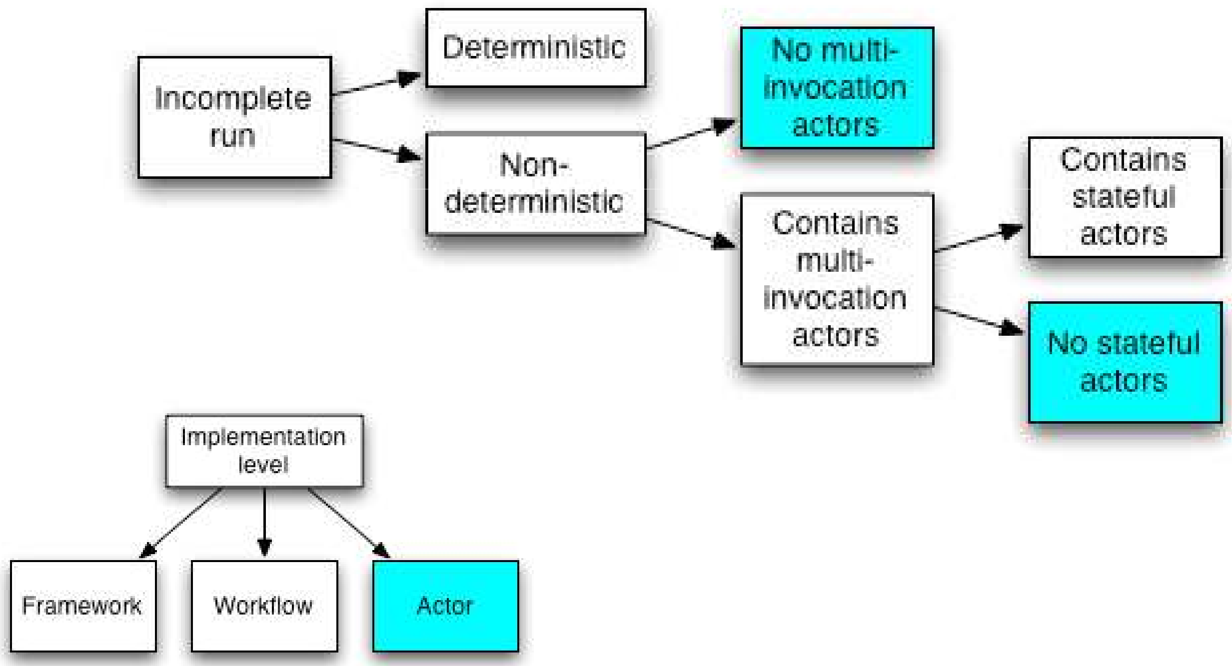
Fault Tolerance Solutions Compared

Rescue DAG – Use: Workflow recovery



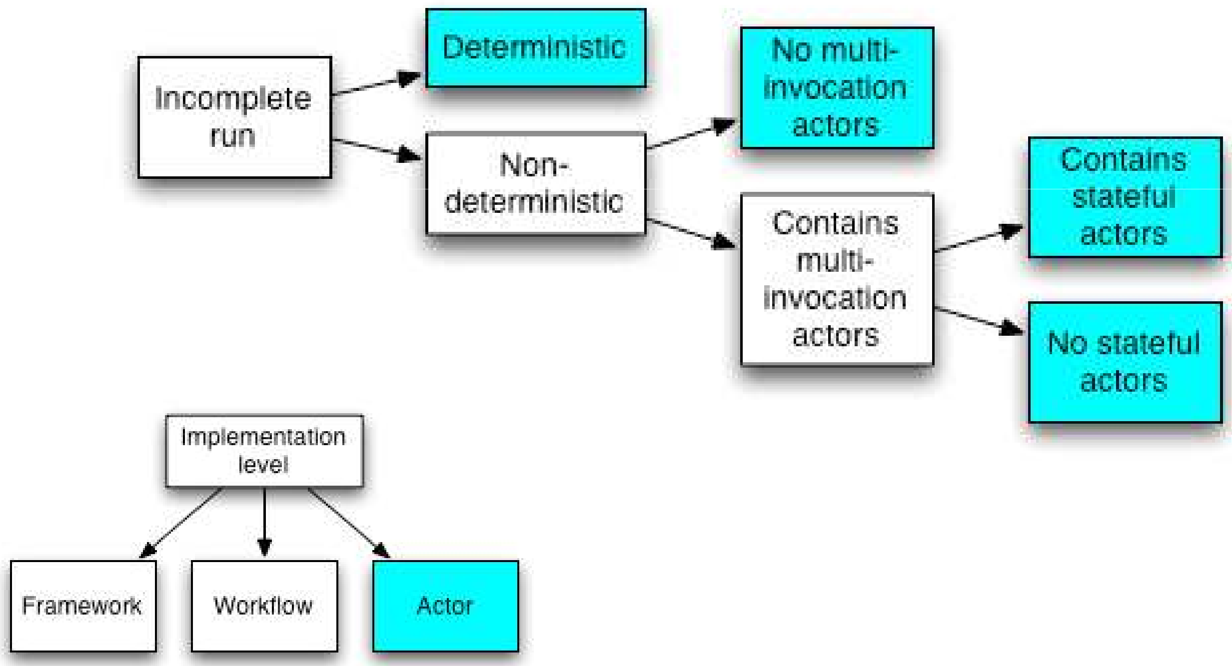
Fault Tolerance Solutions Compared

WATERS/CPES – Use: Workflow recovery



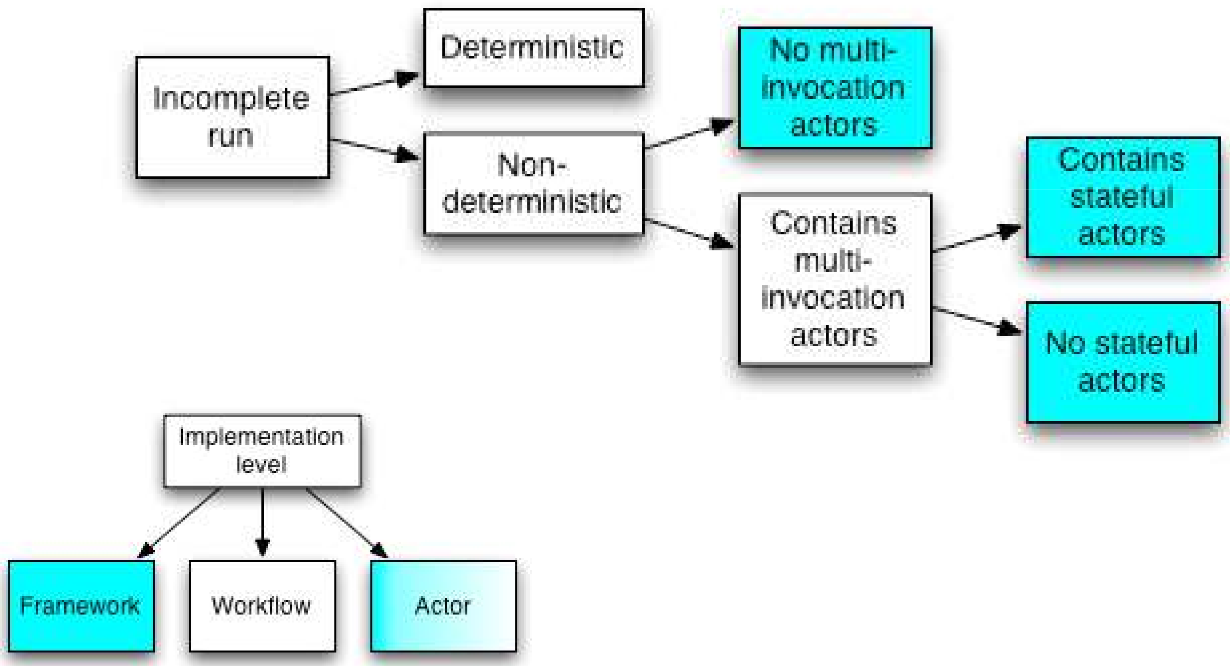
Fault Tolerance Solutions Compared

Ptolemy Backtrack – Use: Exception handling



Fault Tolerance Solutions Compared

Replay/Checkpointing – Use: Workflow recovery



Our Fault Tolerance Approach

- Handles complex MoCs that include streaming, statefulness, etc.
- Uses pre-existing provenance data
- Does not assume that data dependencies within actors are transparent

Our Fault Tolerance Approach

- Recovery based on readily available Provenance

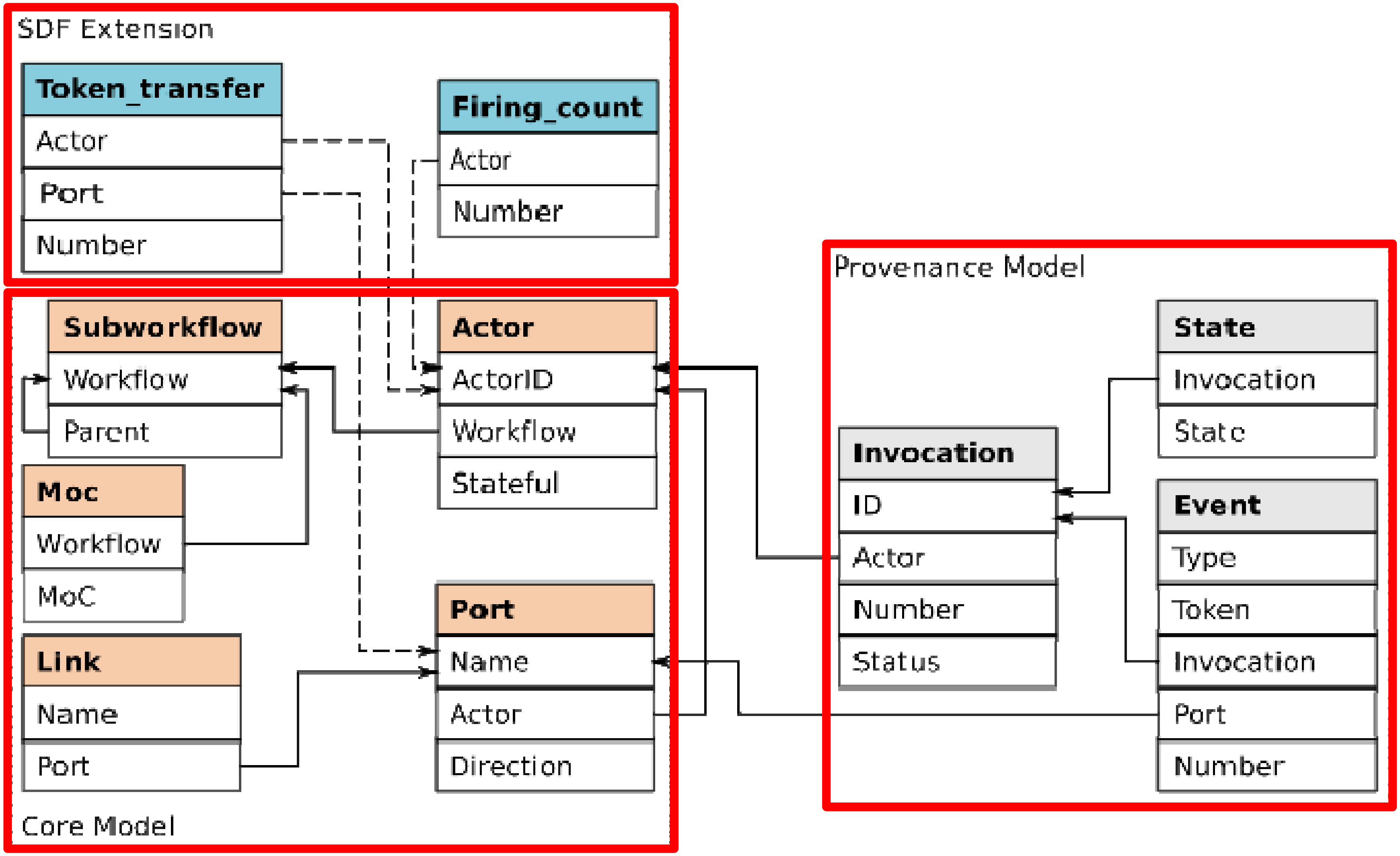
1. Create a uniform model for workflow descriptions and provenance
2. Record actor state in provenance in relation to invocations
3. After a workflow crash: Use provenance data in our uniform model and start recovery

- Different strategies for recovery that balance information captured with recovery efficiency

Our Recovery Strategies

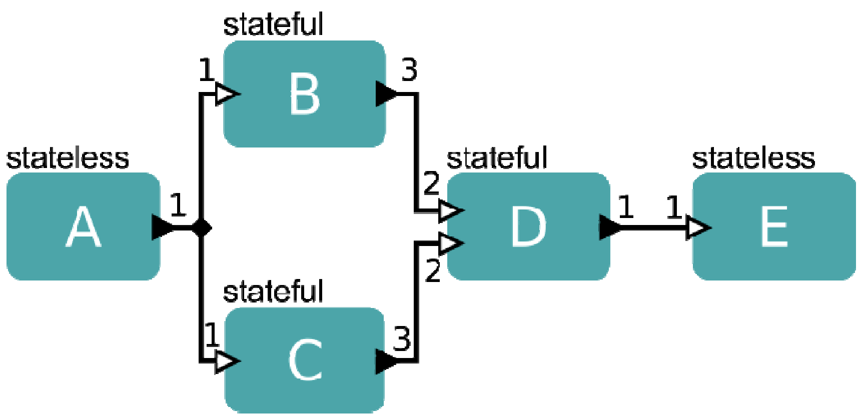
Strategy	Description
Naïve	- Restart the workflow without using provenance - Re-executes everything
Replay	- Use **basic provenance** to speed up recovery - Re-execute stateful actors with input from provenance (*replay*) - Restore all queues - Resume the workflow according to the model of computation
Checkpoint	- **extension** of **replay** strategy - Use **checkpoints** (state of actors stored in provenance) - Reset stateful actors to recorded state - Replay successful invocations after the checkpoint - Restore queue content - Resume the workflow

Model for Workflows and Provenance

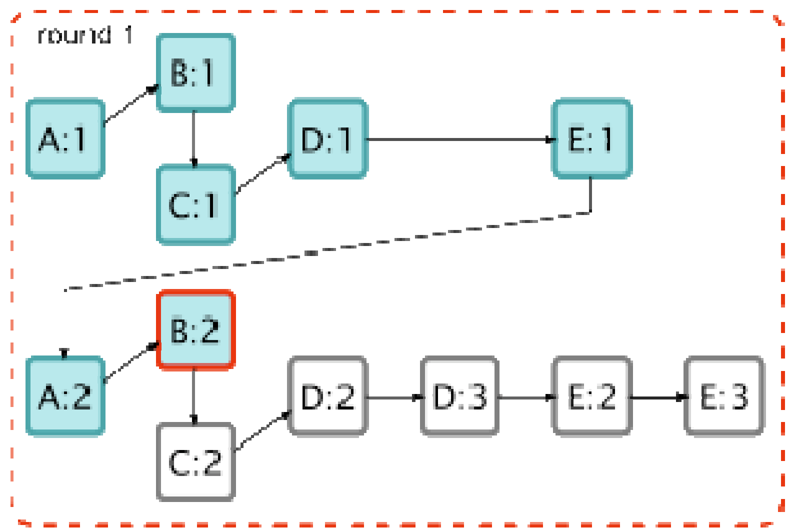


Example: Checkpoint in SDF

Workflow with a mix of stateful and stateless actors



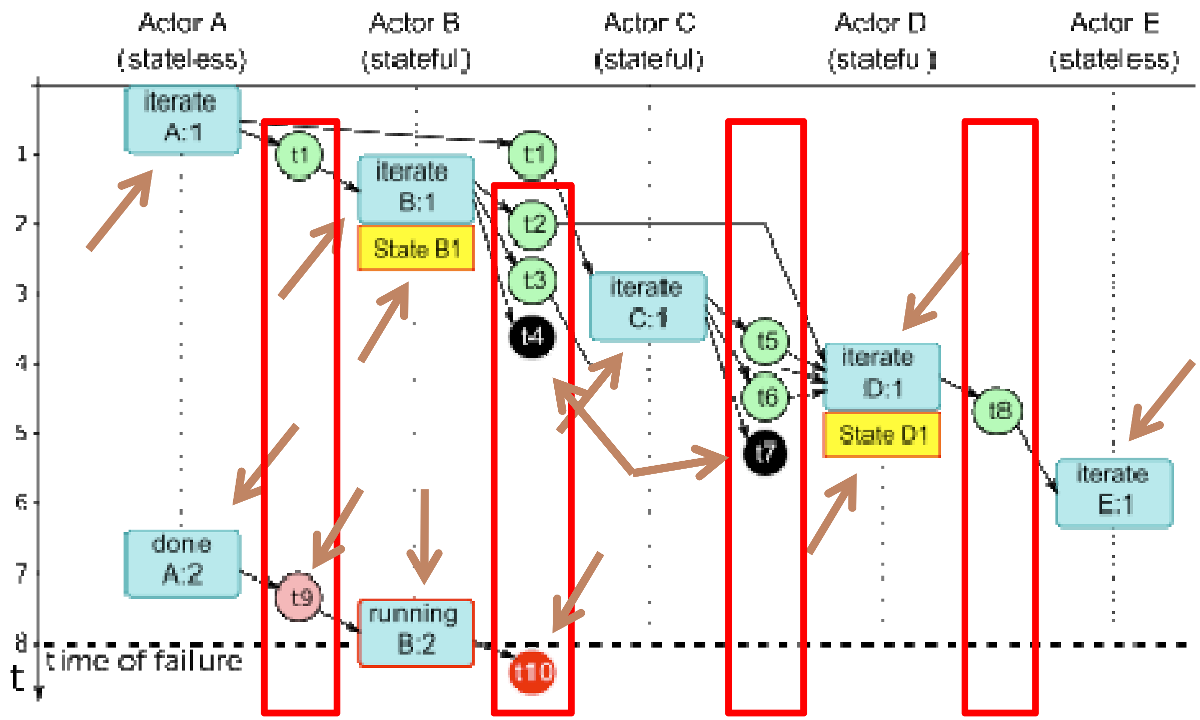
Corresponding schedule of the workflow with a fault during invocation B:2

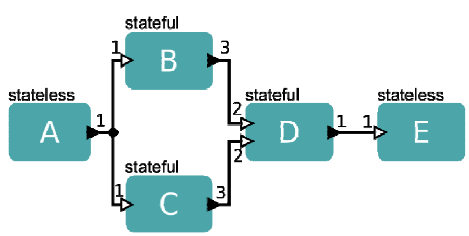


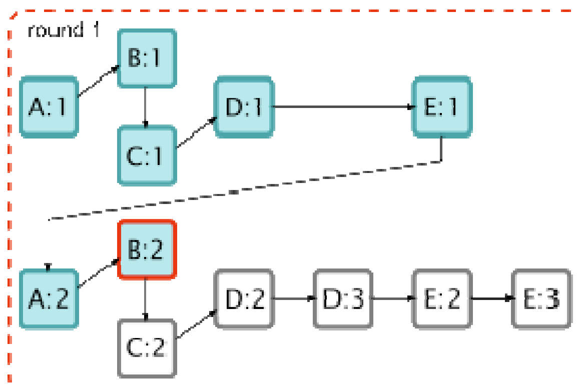
Recovery process overview

- Upon recovery request:
  - SDF director calls the recovery engine

- Recovery:
  - Restore the internal state of actors
  - Replay successful invocations using input tokens from provenance
  - Restore content of all queues
  - Repeat faulty invocations
  - Return to SDF director with information about where to resume







Execution with a Failure

Execution of the previous workflow

Checkpoints for actor B and D but not for C

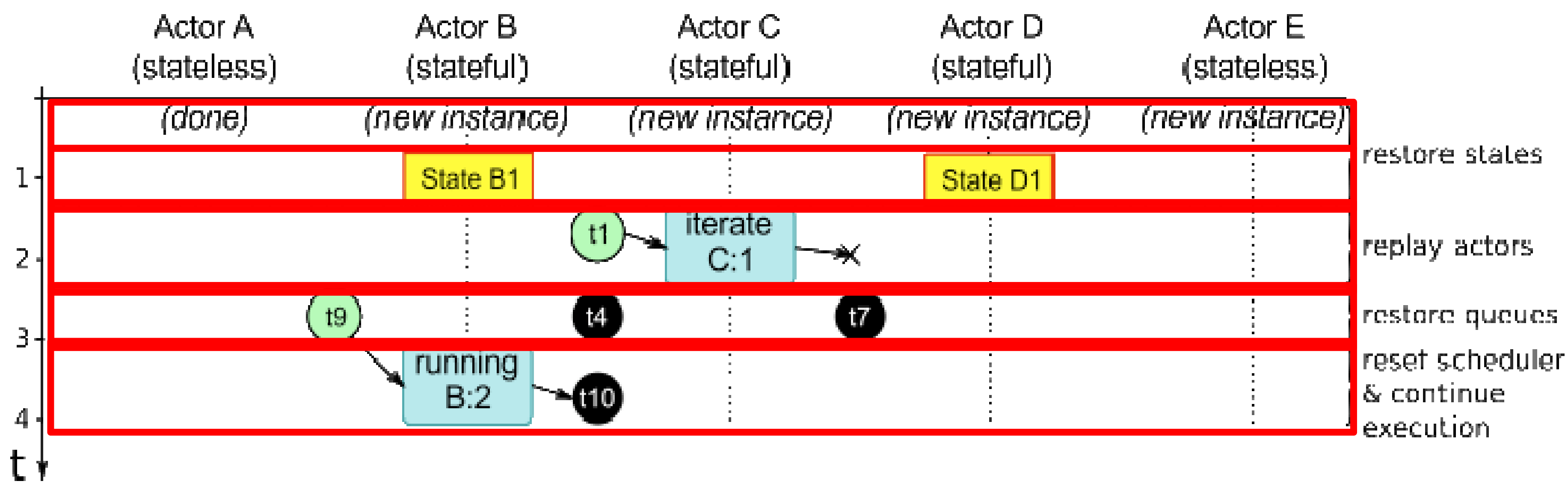
At invocation B:2 - Crash

Tokens **t4** and **t7** - in queue

Token **t9** - to be restored

Token **t10** - to be deleted

Stages of Checkpoint Recovery

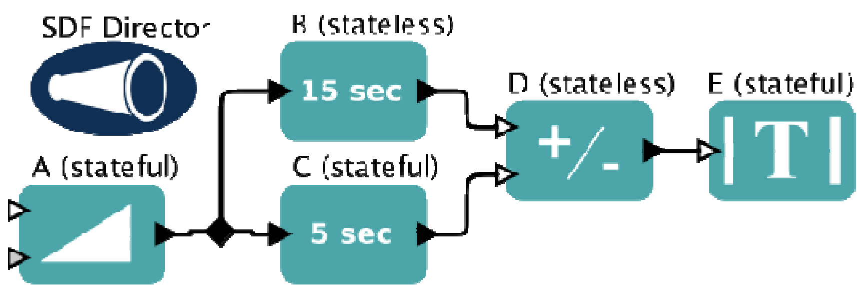


Prototype Implementation in Kepler

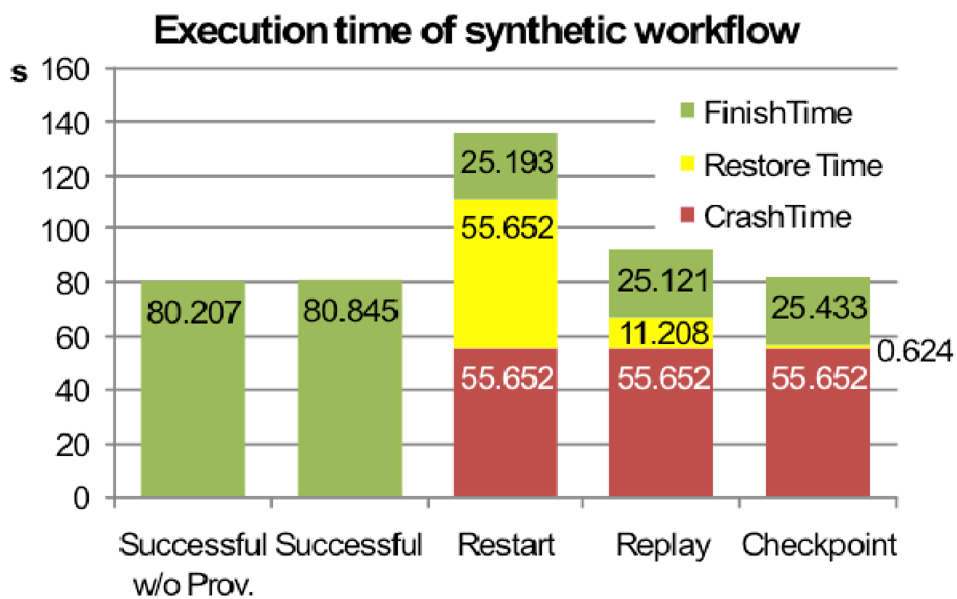
- Using Kepler with the Provenance Recorder
- Extensions to the Provenance Recorder:
  - Extend the provenance schema
  - Record serialized tokens
  - Add queries
- Recovery Extension in the SDF Director:
  - Serialize states after one iteration of the SDF schedule
  - Black-list to prevent capturing transient actor information
  - White-list if actors are annotated with state-information

Evaluation

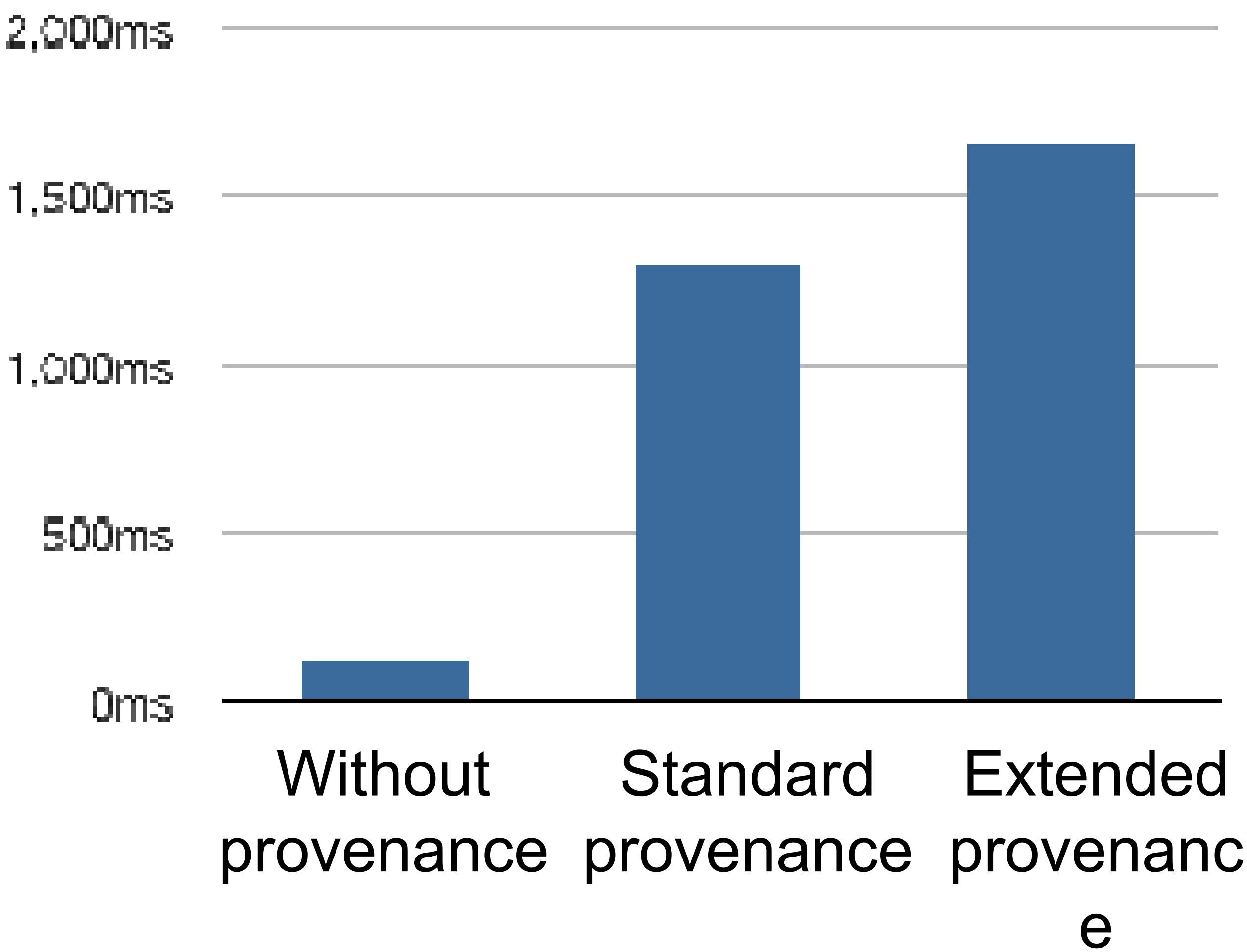
Synthetic Workflow



Results



Provenance Recording Overhead



Worst-case scenario

PN implementation

- PN Domain - Kahn process networks, blocking reads
- Can be multiple failed invocations
- Tokens output by failed invocations can already be used

Conclusion

- Advantages of our strategy:
  - Efficient workflow recovery using readily available information
  - Quick constant time recovery (checkpoint strategy)
  - Generalized approach, saving labor
  - Robustness

- Disadvantages of previous strategies:
  - Required labor-intensive customized systems
  - Failure required restarting long-running workflows from the beginning
  - Caching only works for stateless actors
  - Caching only provides a partial recovery

Related Works

- Hartman, A., Riddle, S., McPhillips, T., Ludäscher, B., Eisen, J.: Introducing W.A.T.E.R.S.: a Workflow for the Alignment, Taxonomy, and Ecology of Ribosomal Sequences. BMC Bioinformatics 11(1) (2010) 317.
- Podhorszki, N., Ludäscher, B., Klasky, S.A.: Workflow automation for processing plasma fusion simulation data. In: Proceedings of the 2nd workshop on Workflows in support of large-scale science. WORKS '07, New York, NY, USA (2007) 35–44.
- Crawl, D., Altintas, I.: A Provenance-Based Fault Tolerance Mechanism for Scientific Workflows. In: Provenance and Annotation of Data and Processes. Volume 5272 of LNCS. Springer Berlin / Heidelberg (2008) 152–159.
- Wang, L., Lu, S., Fei, X., Chebotko, A., Bryant, H.V., Ram, J.L.: Atomicity and provenance support for pipelined scientific workflows. Future Generation Computer Systems 25(5) (2009) 568 – 576.