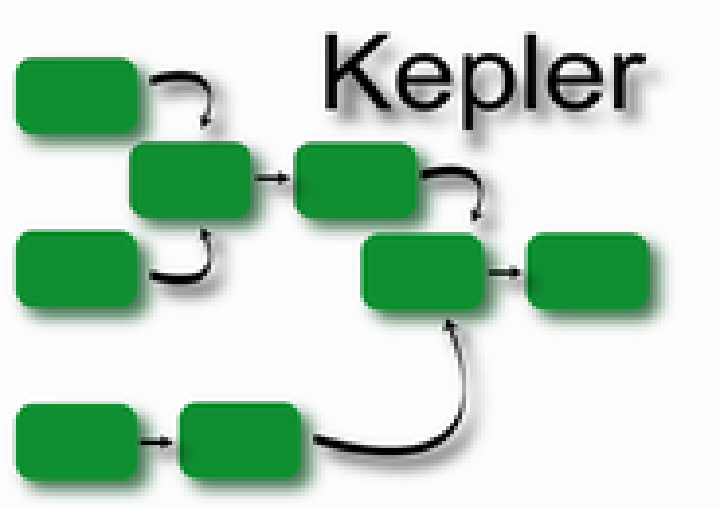
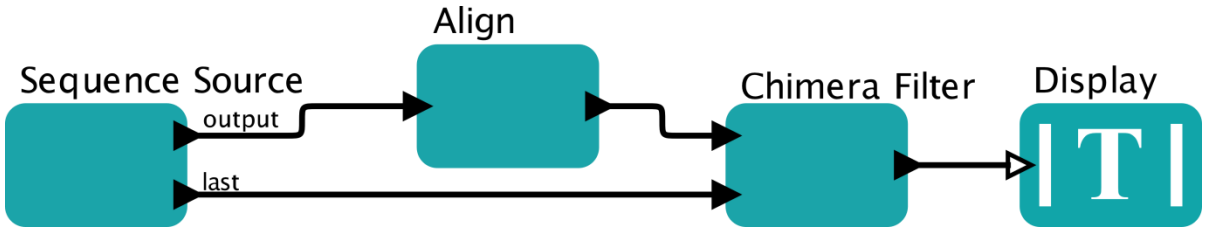


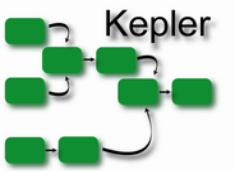
Workflow Fault Tolerance for Kepler

Sven Köhler, Thimothy McPhillips, Sean Riddle, Daniel Zinn, Bertram Ludäscher

Introduction

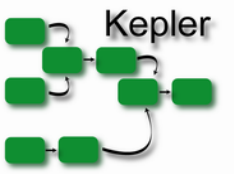
- Scientific Workflows
  - Automate scientific pipelines
  - Have long running computations
  - Often contain stateful actors
- Workflow execution can crash because of …
  - Hardware failures
  - Power outages
  - Buggy / malicious actors, …
- Current approach: Start workflow from the beginning





Current Fault Tolerance Solutions …

- 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.]
- Use caching strategies for faster re-execution
  - W.A.T.E.R.S. memoization [Hartman et al.]
  - "Skip over" strategy [Podhorszki et al.] (CPES)`



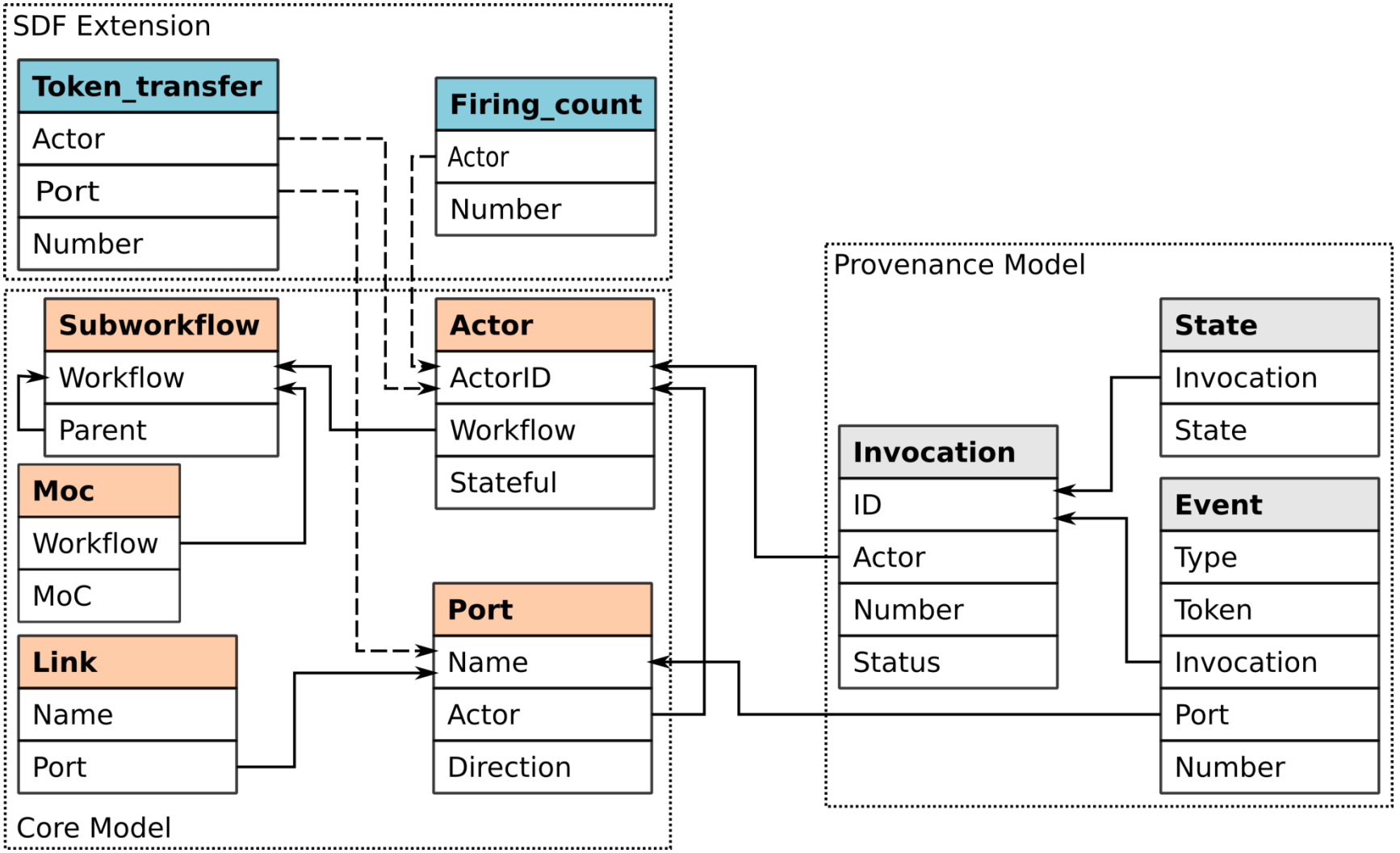
Our Fault Tolerance Approach

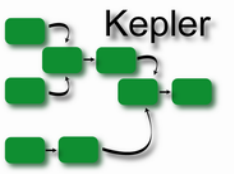
- Recovery based on readily available Provenance

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

Model for Workflows and Provenance



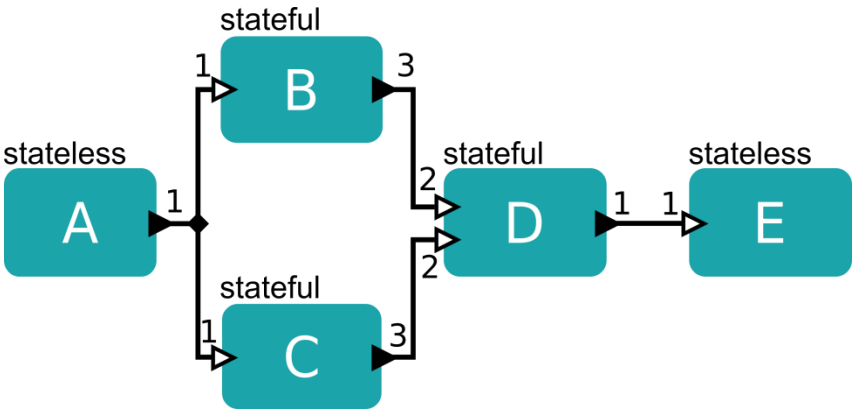


Our Recovery Strategies

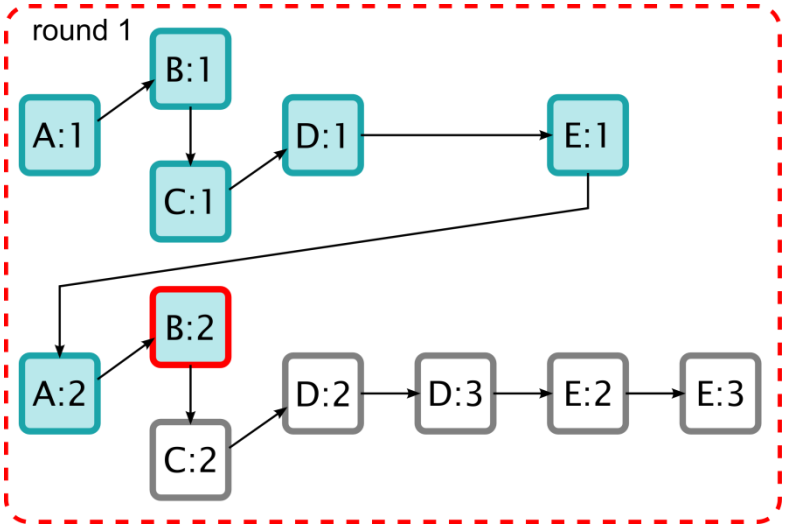
Strategy	Description
Naive	- Restart the workflow without using provenance - Re-executes everything
Replay	- Use **basic provenance** to speed up recovery - Re-execute stateful actor with input from provenance (*replay*) - Restore all queues - Resume the workflow according to the model of computation
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

Example: Checkpoint in SDF

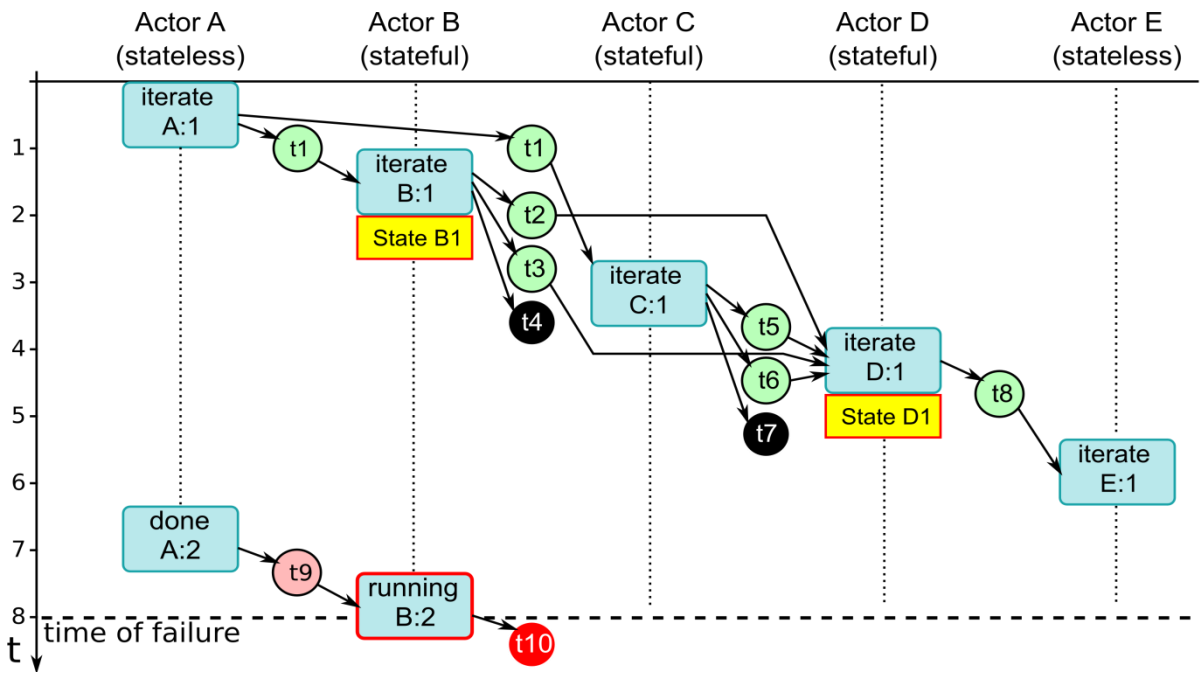
Workflow with a mix of stateful and stateless actors



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



Execution with a Failure



Execution of the previous workflow

Checkpoints for actor B and D but not for C

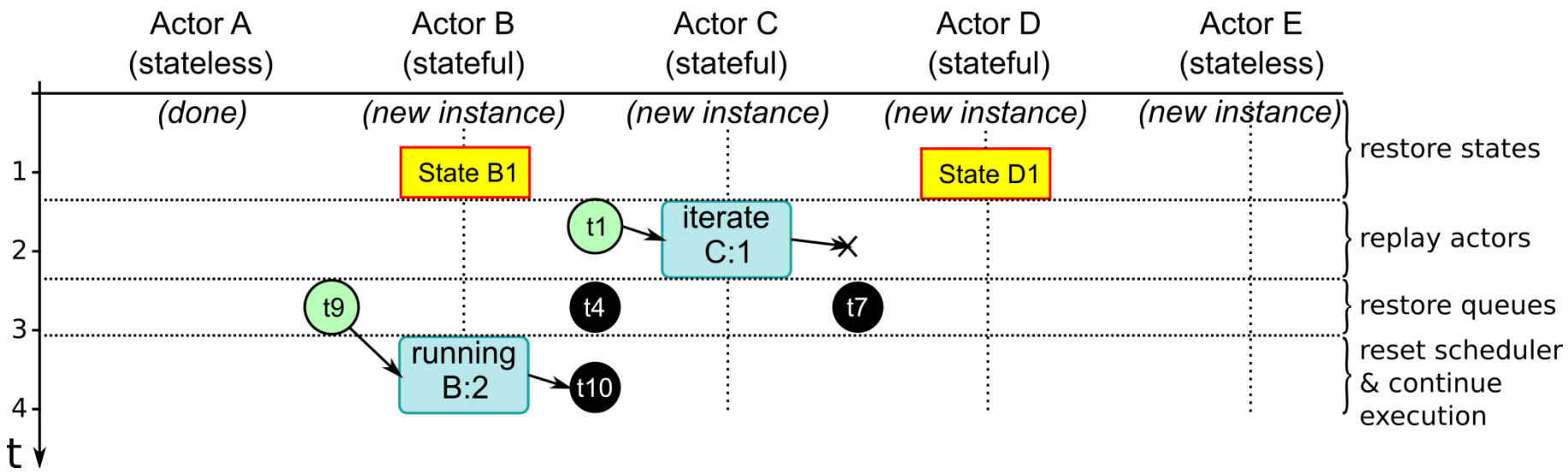
At invocation B:2 - Crash

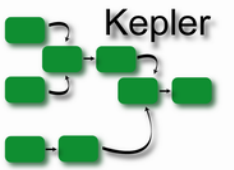
Tokens **t4** and **t7** - in queue

Token **t9** - to be restored

Token **t10** - to be deleted

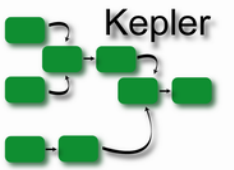
Stages of Checkpoint Recovery





Prototype Implementation in Kepler

- Using Kepler with the Provenance Recorder
- Extensions to the Provenance Recorder:
  - Record serialized tokens
  - Extend the provenance schema
  - 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



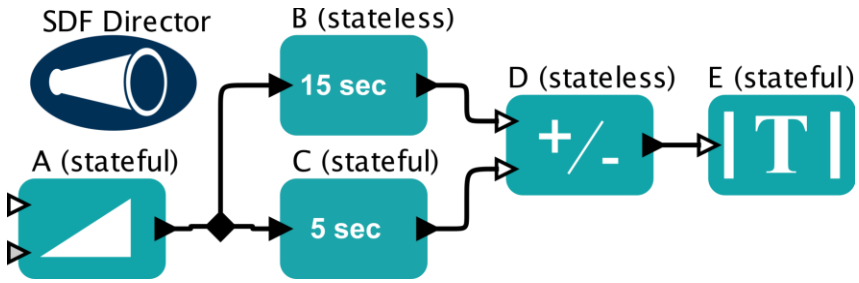
Prototype Implementation in Kepler

- Upon restart:
  - SDF director checks provenance information
  - SDF director calls the recovery engine

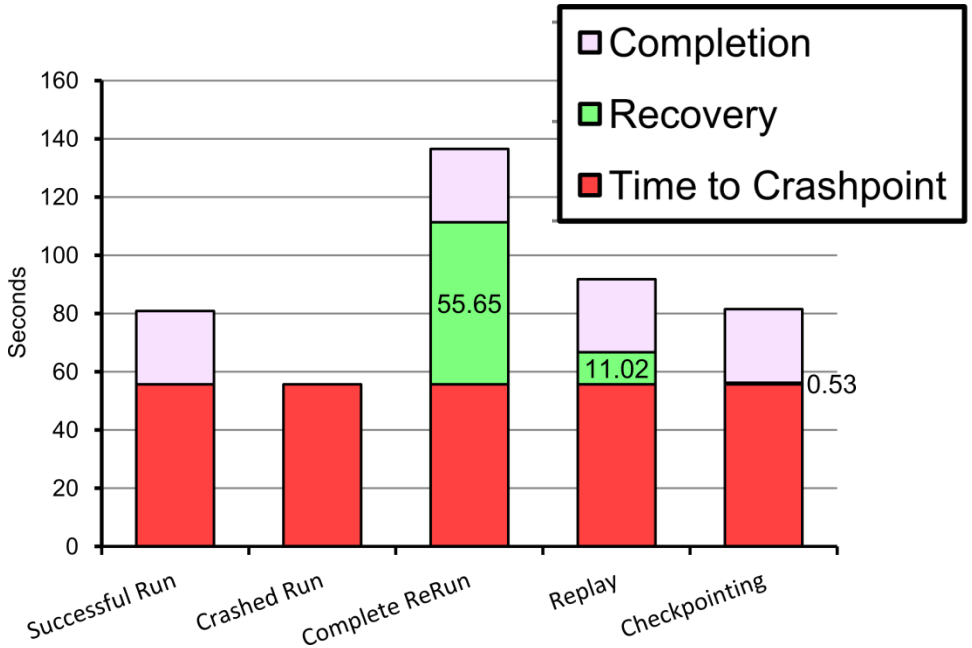
- Recovery:
  - Restore the internal state of actors
  - Replay successful invocations using input tokens from provenance
  - Restore content of all queues
  - Return to SDF director with information about where to resume

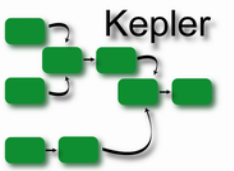
Evaluation

Synthetic Workflow



Results





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