ASYNCHRONOUS FILE TRANSFER IN CONDOR

Parag Mhashilkar (Fermi National Accelerator Laboratory)
Cathrin Weiss (Condor Project, University of Wisconsin at Madison)

Overview

- CEDPS Project
- Motivation: Asynchronous sandbox management
- Asynchronous sandbox management
- Sandbox Management & Sandbox Manager
- Sandbox Transfer in Condor
  - Current Protocol
  - Asynchronous sandbox transfer
- Prototype & Its Performance
- Ongoing & Future Work
- Acknowledgements

CEDPS Project

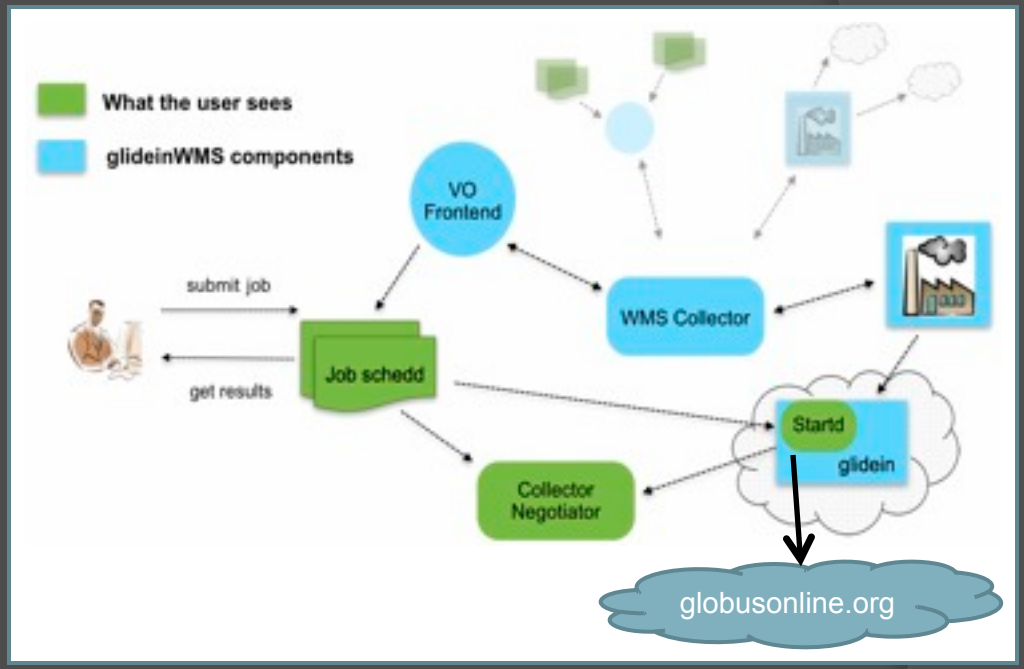
- CEDPS: The five year project started in 2006, funded by Department of Energy (DOE)
- Goals
  - Produce technical innovations for rapid and dependable **data placement** within a distributed high performance environment and for the construction of **scalable science services** for data and computing from many clients.
  - Address performance and functionality troubleshooting of these and other related distributed activities.
- Collaborative Research
  - Mathematics & Computer Science Division, Argonne National Laboratory
  - Computing Division, Fermi National Accelerator Laboratory
  - Lawrence Berkeley National Laboratory
  - Dept of Computer Science, University of Wisconsin Madison
  - Information Sciences Institute, University of Southern California

CEDPS Activities at FNAL

- Investigating data movement mechanisms for data stage-out on Grids
  - globusonline.org needs integration with SRM interface for OSG
- Supporting the integration of data movement mechanisms with scientific DH frameworks
  - Supporting the integration of globusonline.org with Dark Energy Survey (DES) data handling system
- Integration of asynchronous data stage-out mechanisms in overlay workload management systems (…this talk…)
  - Release resources at job termination. Delegate data stage-out to external agents.
  - Integrate support for globusonline.org into glideinWMS

glideinWMS

- Pilot-based WMS that creates on demand a dynamically-sized overlay condor batch system on Grid resources to address the complex needs of VOs in running application workflows
- Components
  - WMS Collector
  - Glidein Factory
  - User Pool Collector
  - User Scheduler
  - VO Frontend



- Factory knows about the sites and how to submit glideins to the sites
- VO frontend knows about the user job details
- WMS Collector acts as a dashboard for Factory - VO Frontend communication.
- Provides an end-to-end solution for several VOs in OSG

Motivation

- Typical life cycle for a job on a worker node
  - Transfer input sandbox
  - Run computational phase and produce output
  - Transfer output sandbox
- We will be focusing on output sandbox transfer during this talk.

Motivation [contd.]

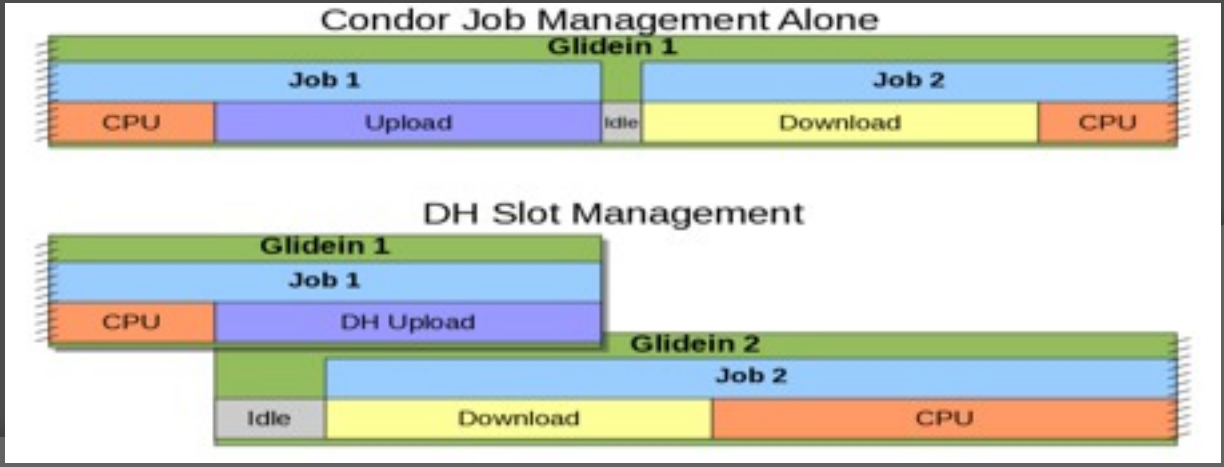
- Sandbox transfer failures & inefficiencies have less impact on throughput for -
  - Short running jobs
    - Less CPU wasted
  - Jobs with small output sandbox
- But …
  - They have different set of problems (not covered here) like -
    - Higher overhead before the jobs starts running
    - Black-hole effect

Motivation [contd.]

- Seen by glideinWMS users, failures & inefficiencies have significant impact on throughput for –
  - Jobs with large input sandbox
    - CPU is not utilized when a sandbox is transferred in
  - Long running jobs
    - Output sandbox transfer can fail wasting CPU consumed by the job
  - Jobs with large output sandbox
    - CPU is not utilized when a sandbox is transferred back
    - Wasted resources in case of intermittent failures during the transfer
- Can we do better?

Asynchronous Sandbox Management

- Enhance glideinWMS by
  - Increasing the CPU utilization of Condor-managed resources in a wide area environment through CPU and network I/O overlap enabled by asynchronous transfers of sandboxes – Miron Livny
- What does this mean?
  - Pipeline the transfer of asynchronous sandboxes in Condor using globusonline.org or other transfer protocols
  - Multiple transfers can take place via transfer slots
  - New job can start running if the previous job has entered stage-out state
  - Can reattempt failed transfers as needed
  - Support multiple transfer protocols using transfer Plug-ins



First Prototype: Condor Hooks

- Support at the Application Layer (glideinWMS).
- Use Condor Hooks
  - To identify end of CPU stage
  - Initiate the output sandbox transfer
- Start another condor_startd to accept new job

Asynchronous Sandbox Management in Condor

- Rather than the application, let condor transfer the output sandboxes asynchronously
  - Generic
  - Robust
  - Reliable
  - Scalable

Sandbox Manager

- Sandbox Manager module extends the sandbox management functionality of the Condor.
- Collaborative Effort
  - **Fermi team** is implementing the Sandbox Manager
    - A repository for sandboxes
  - **Condor team** working on output sandbox management in the Condor
    - Internal protocol changes
    - Details to follow …

Sandbox Manager

- Condor_startd interfaces with the sandbox manager to keep track of output sandboxes
- Sandbox Manager manages the sandbox objects
  - Register sandboxes
  - Update sandboxes
  - Transfer sandboxes
  - Unregister sandboxes
  - […]

What is necessary to integrate this into Condor?

Current sandbox transfer

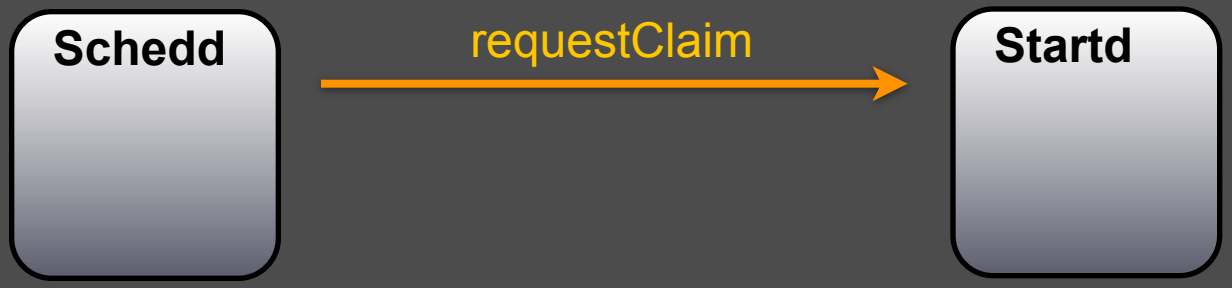
New Claim

Current sandbox transfer

New Claim

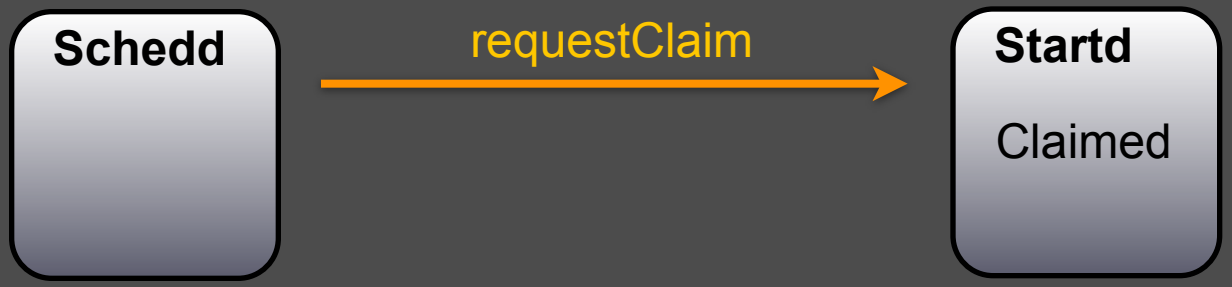
Current sandbox transfer

New Claim



Current sandbox transfer

New Claim

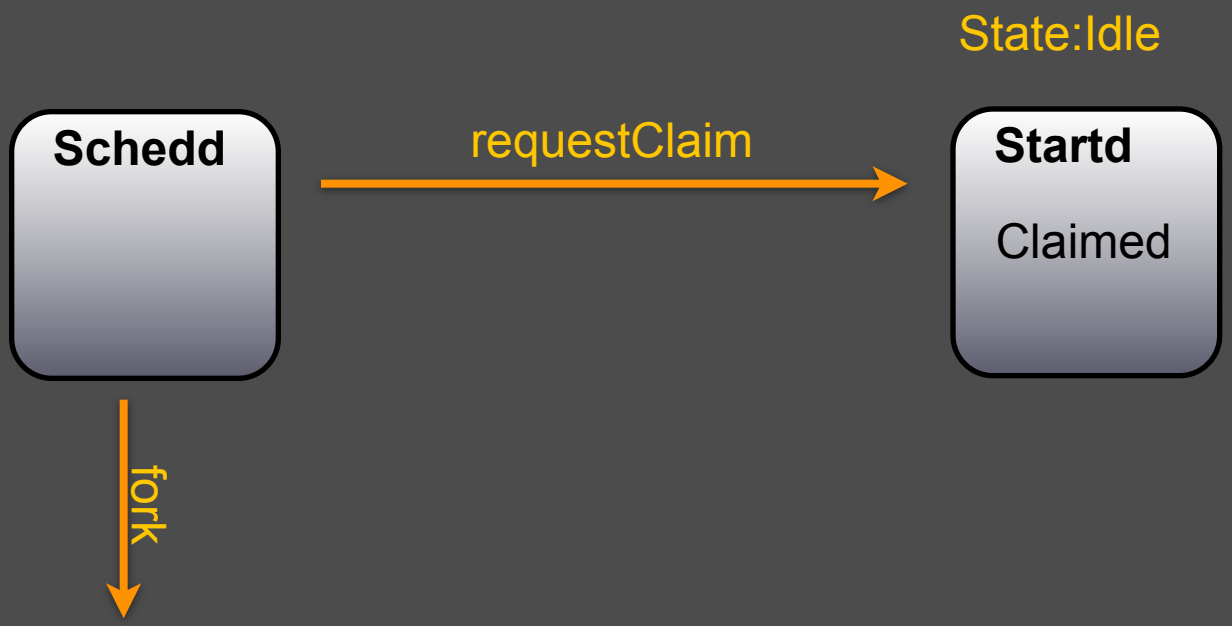


Current sandbox transfer

New Claim

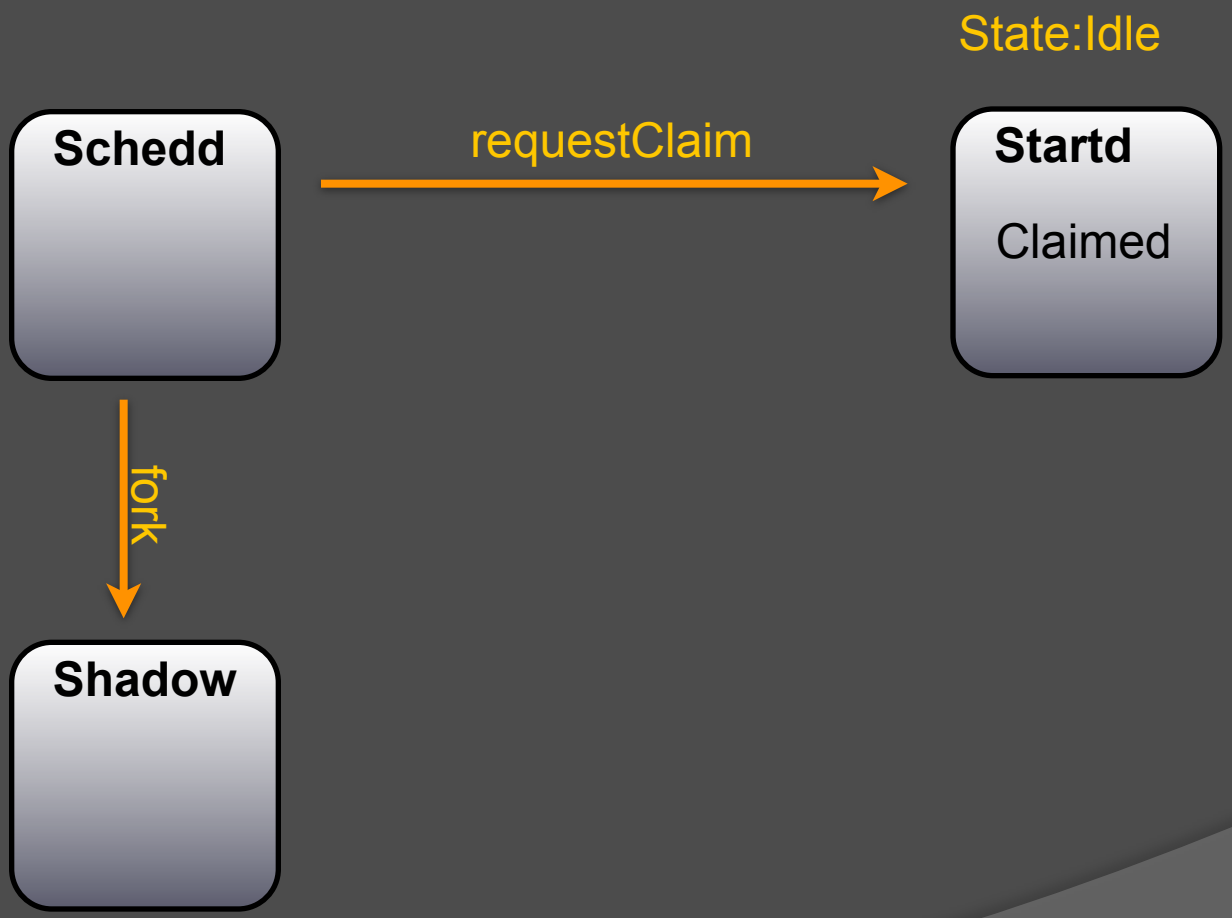
Current sandbox transfer

New Claim



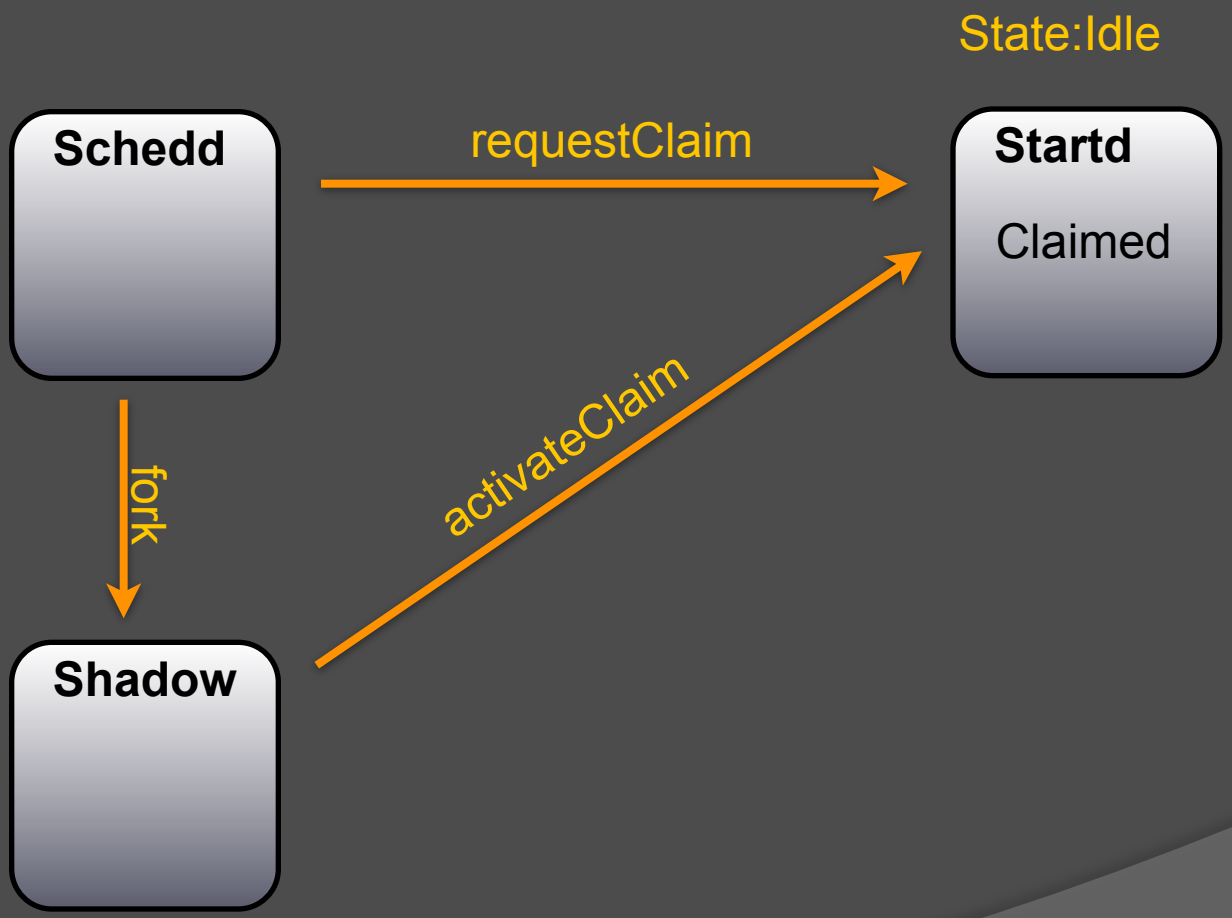
Current sandbox transfer

New Claim



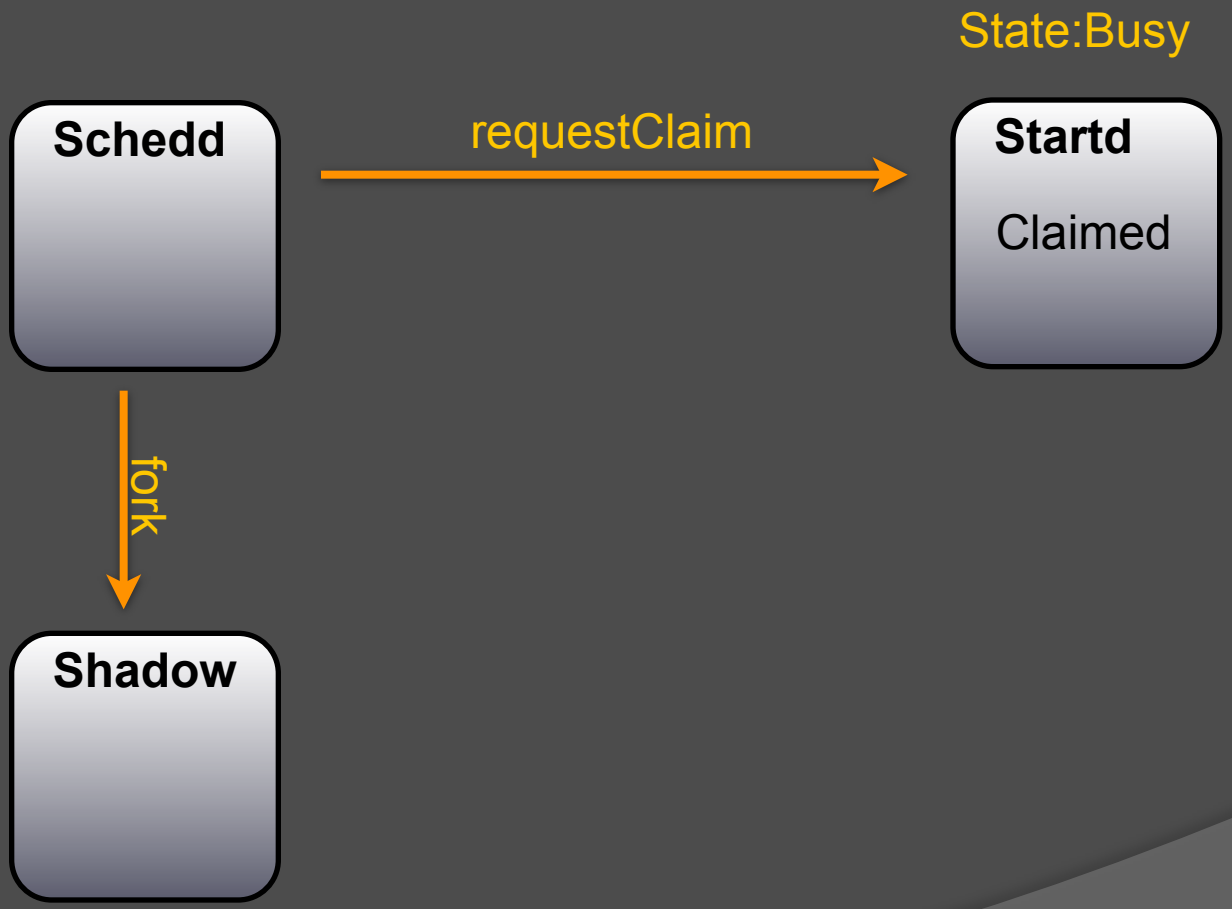
Current sandbox transfer

New Claim



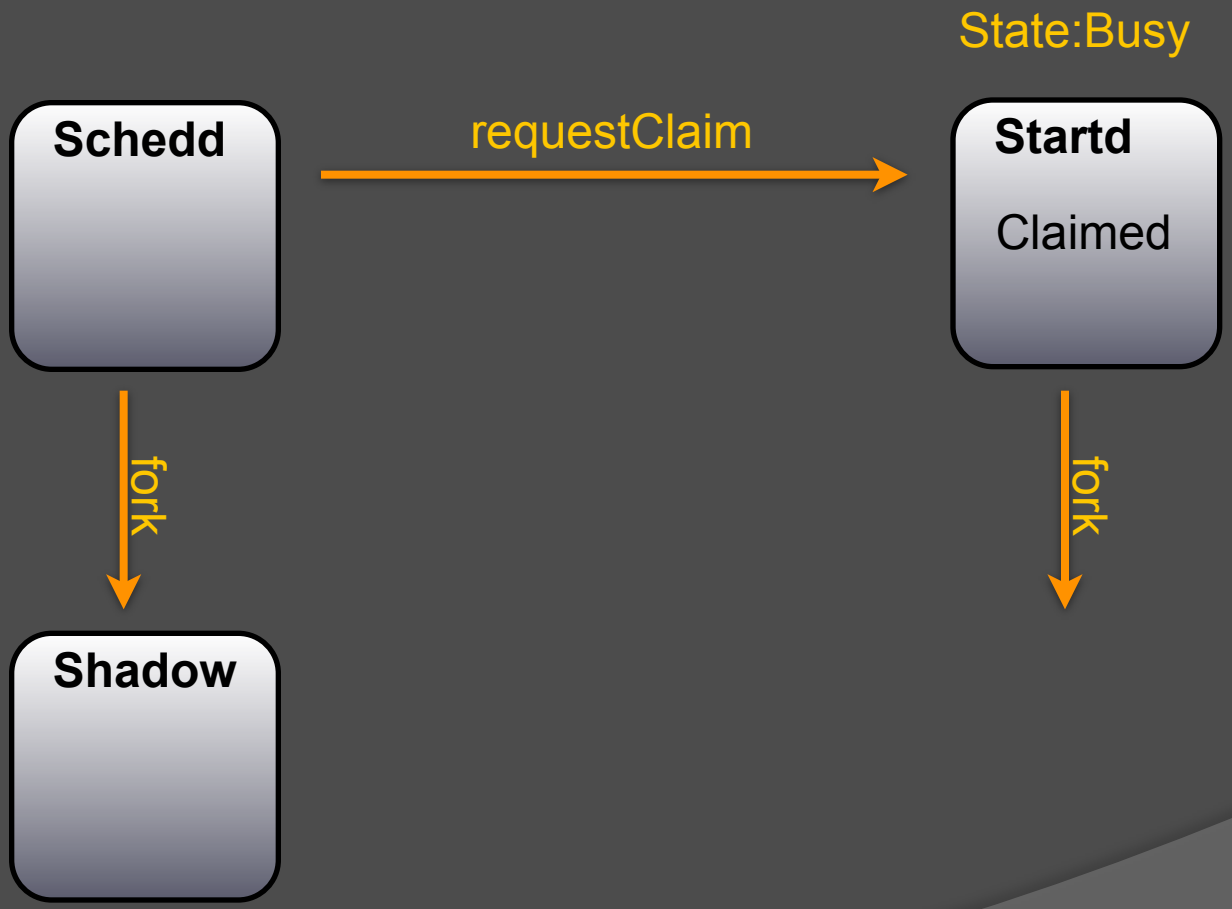
Current sandbox transfer

New Claim



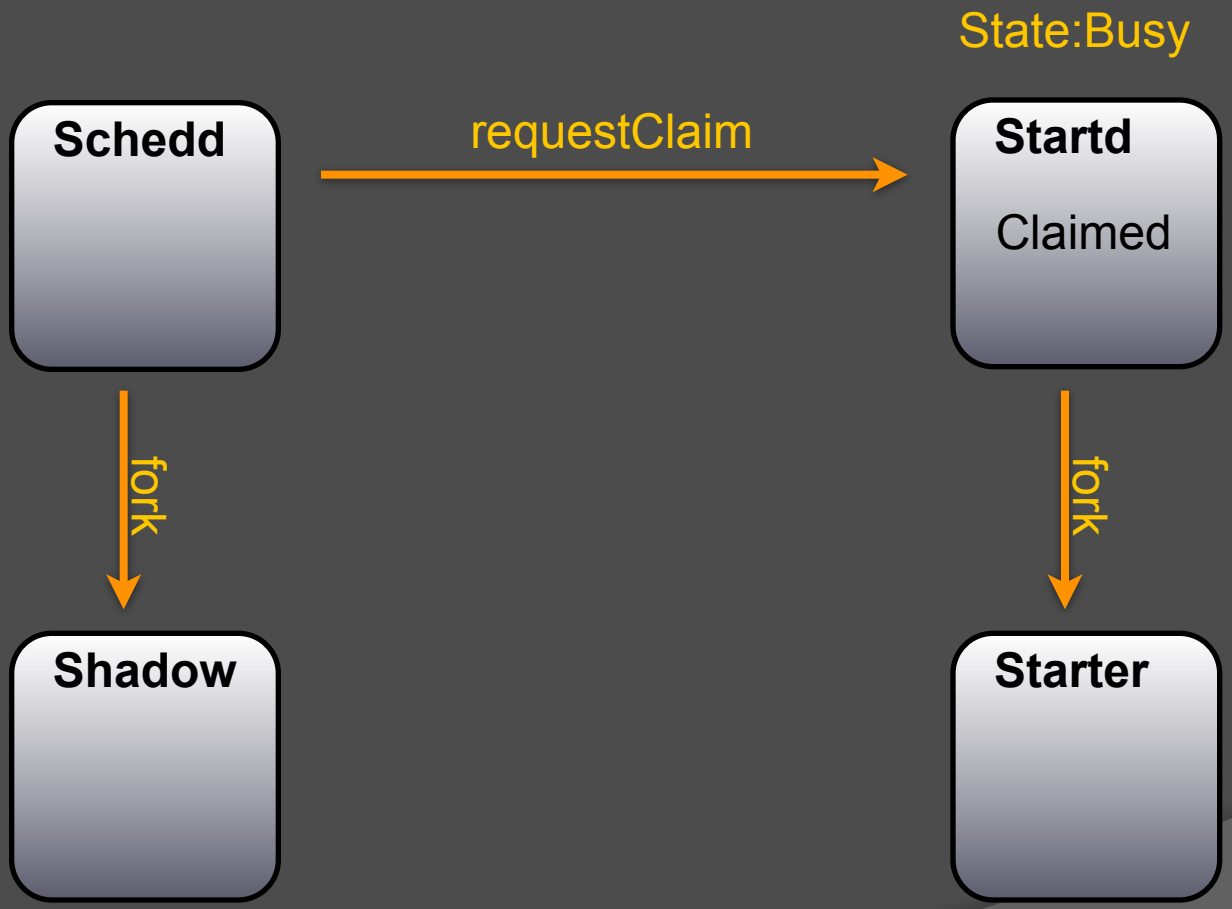
Current sandbox transfer

New Claim



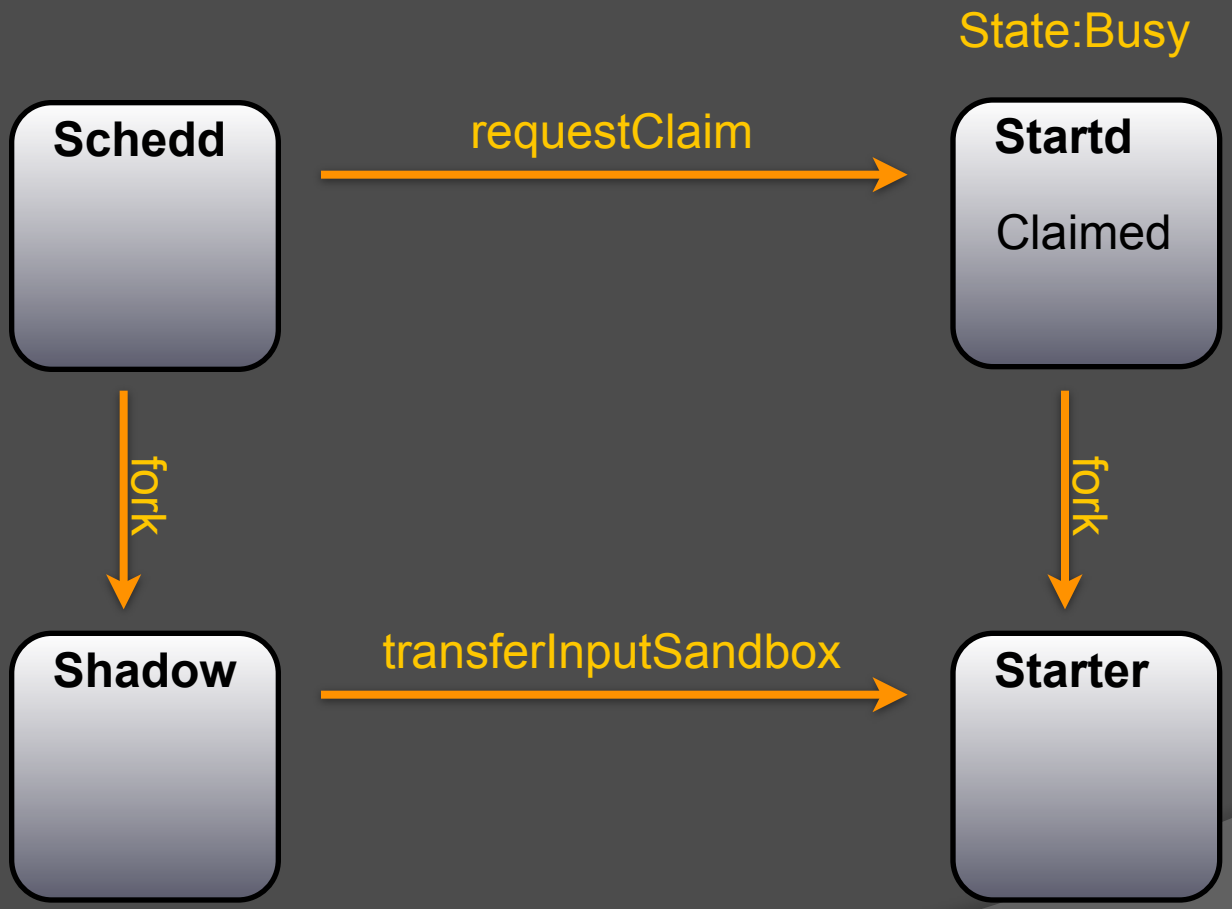
Current sandbox transfer

New Claim



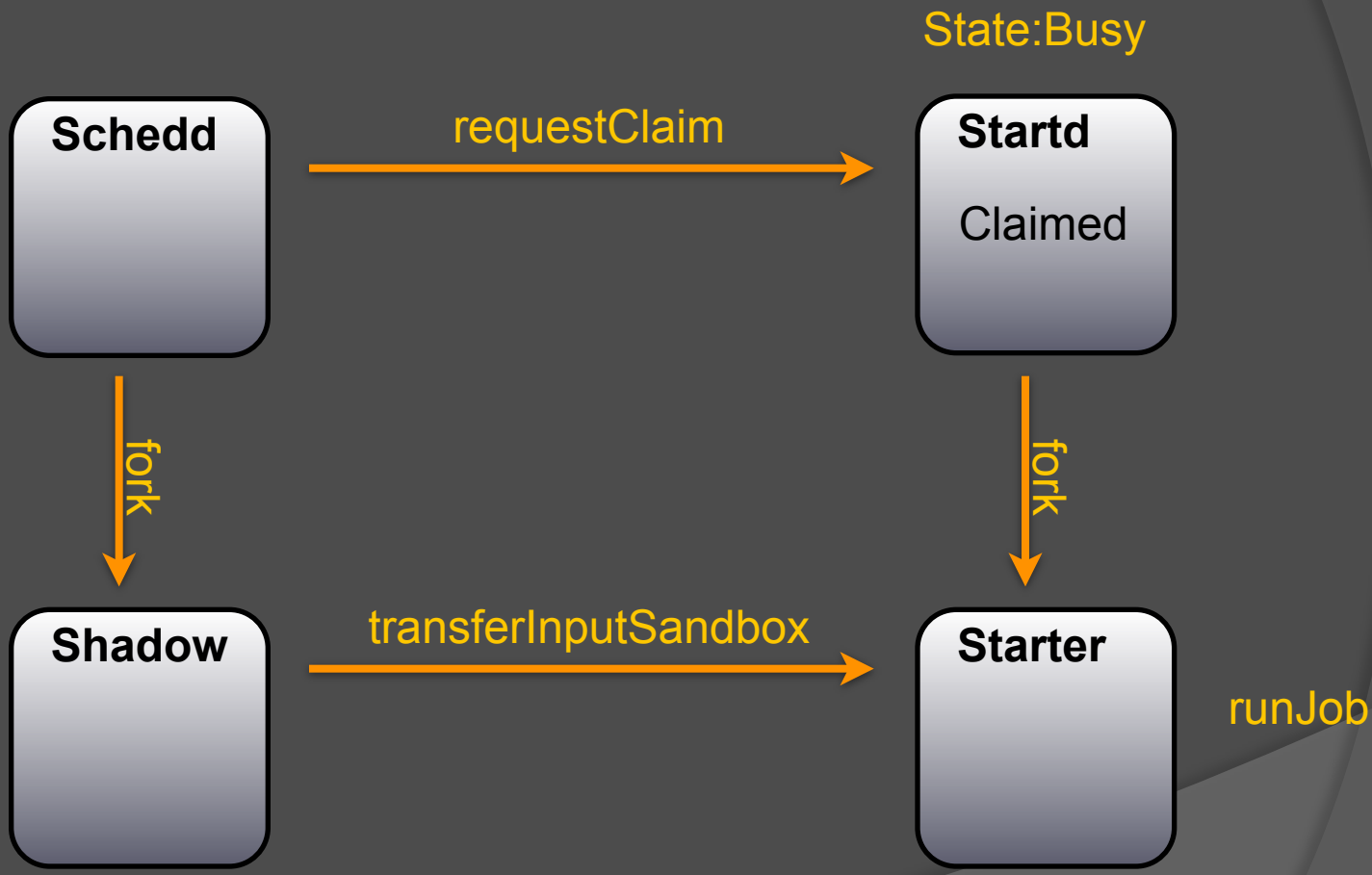
Current sandbox transfer

New Claim



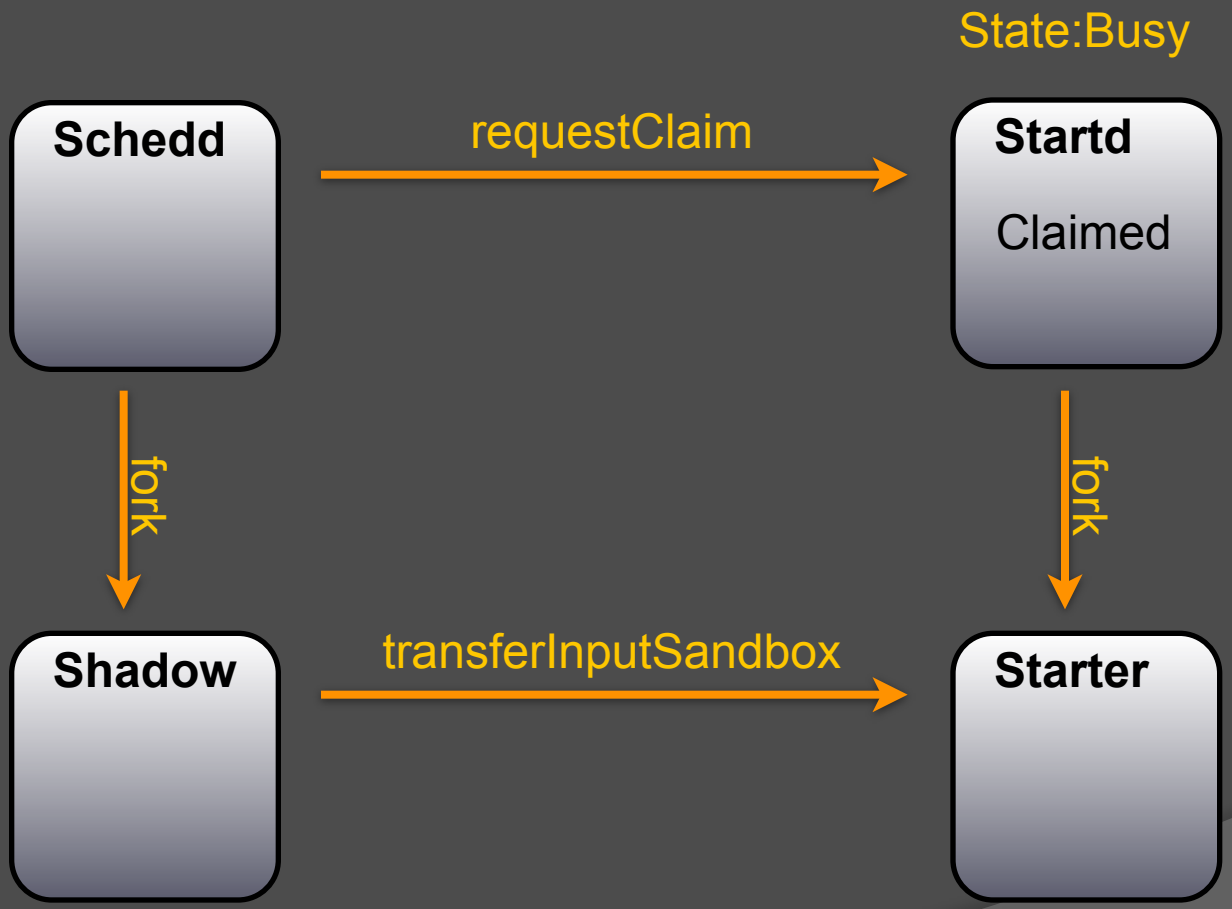
Current sandbox transfer

New Claim



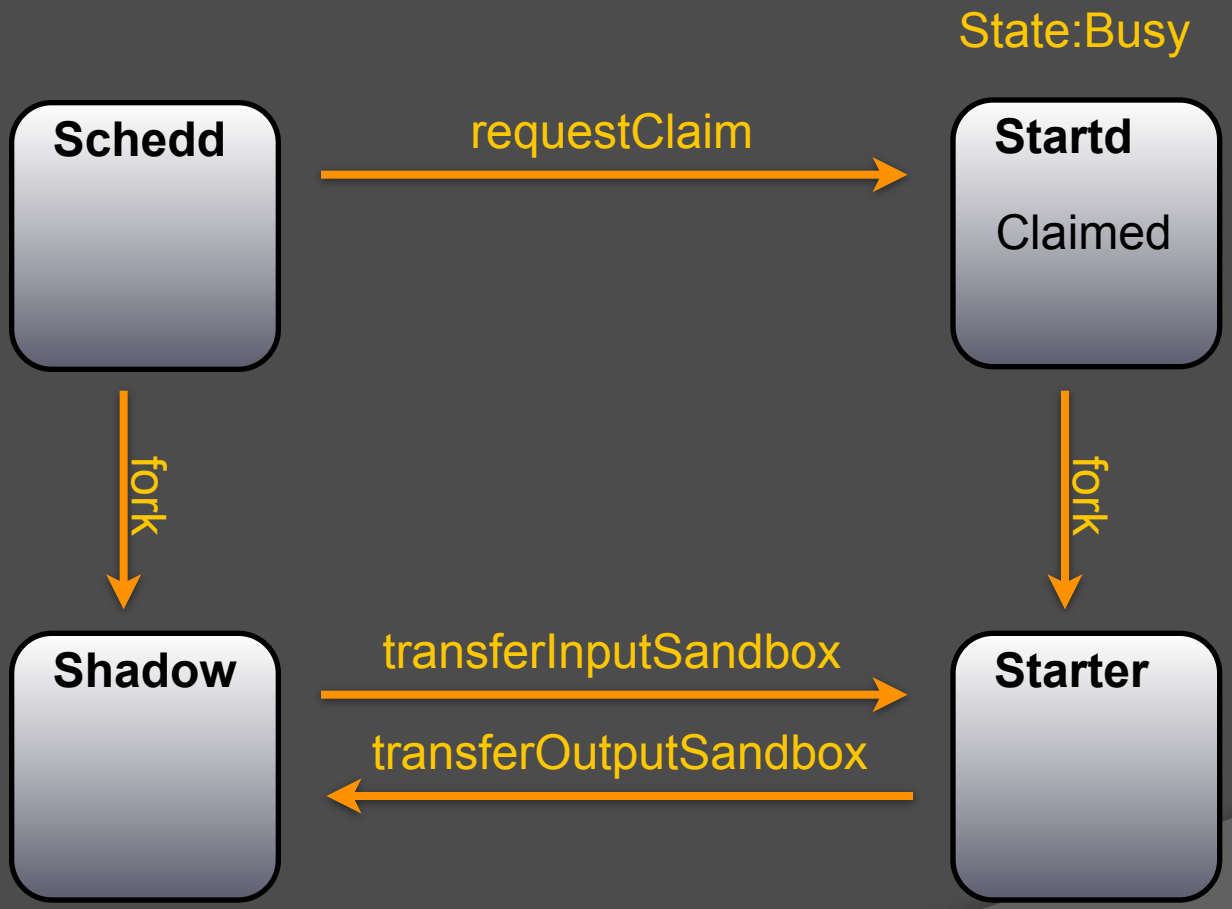
Current sandbox transfer

New Claim



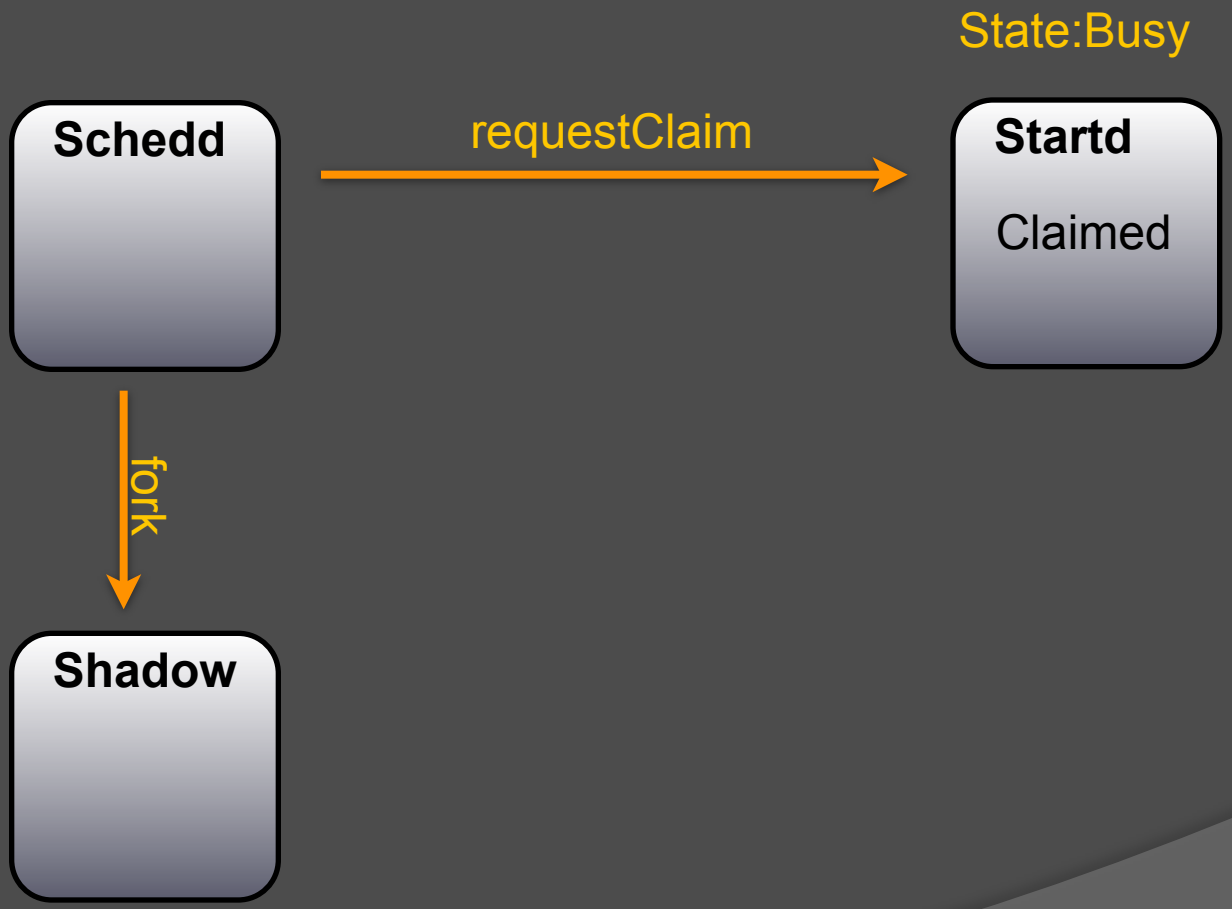
Current sandbox transfer

New Claim



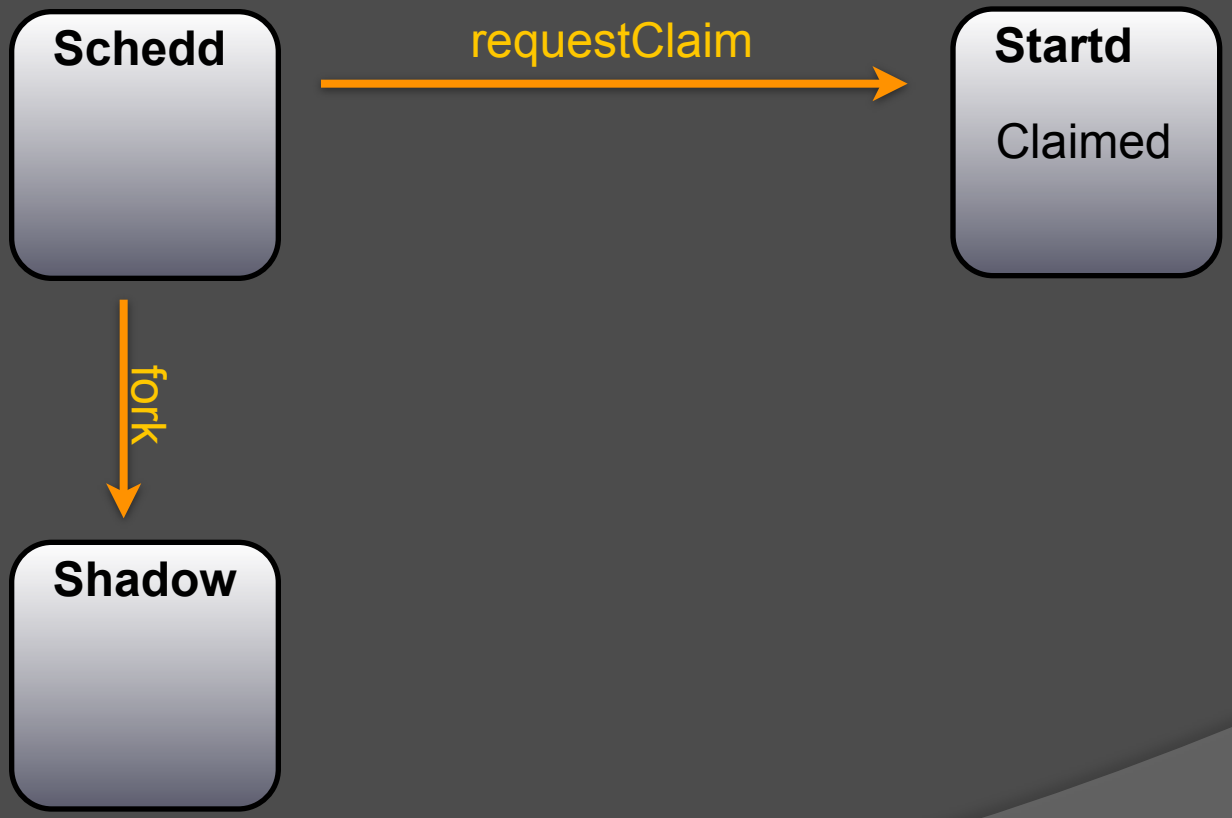
Current sandbox transfer

New Claim



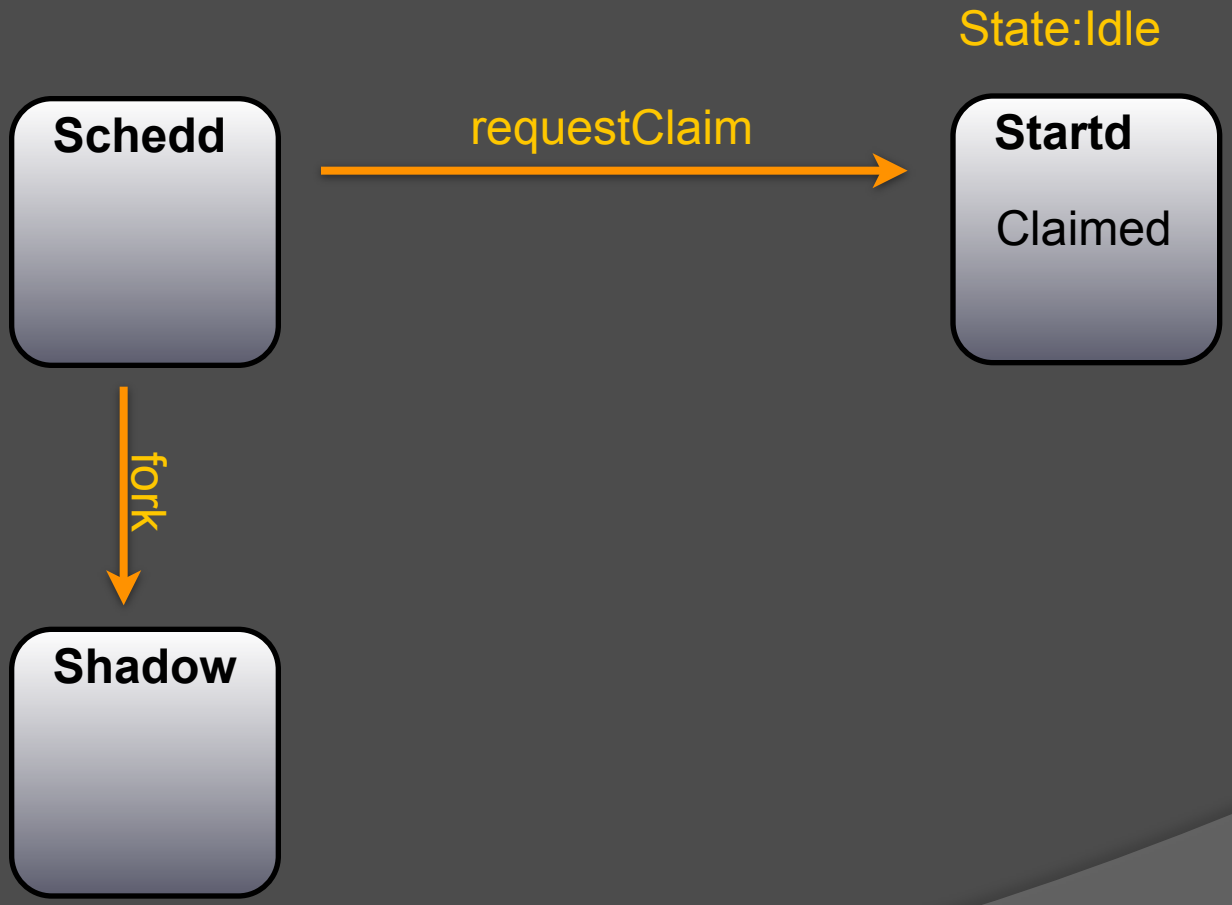
Current sandbox transfer

New Claim



Current sandbox transfer

New Claim



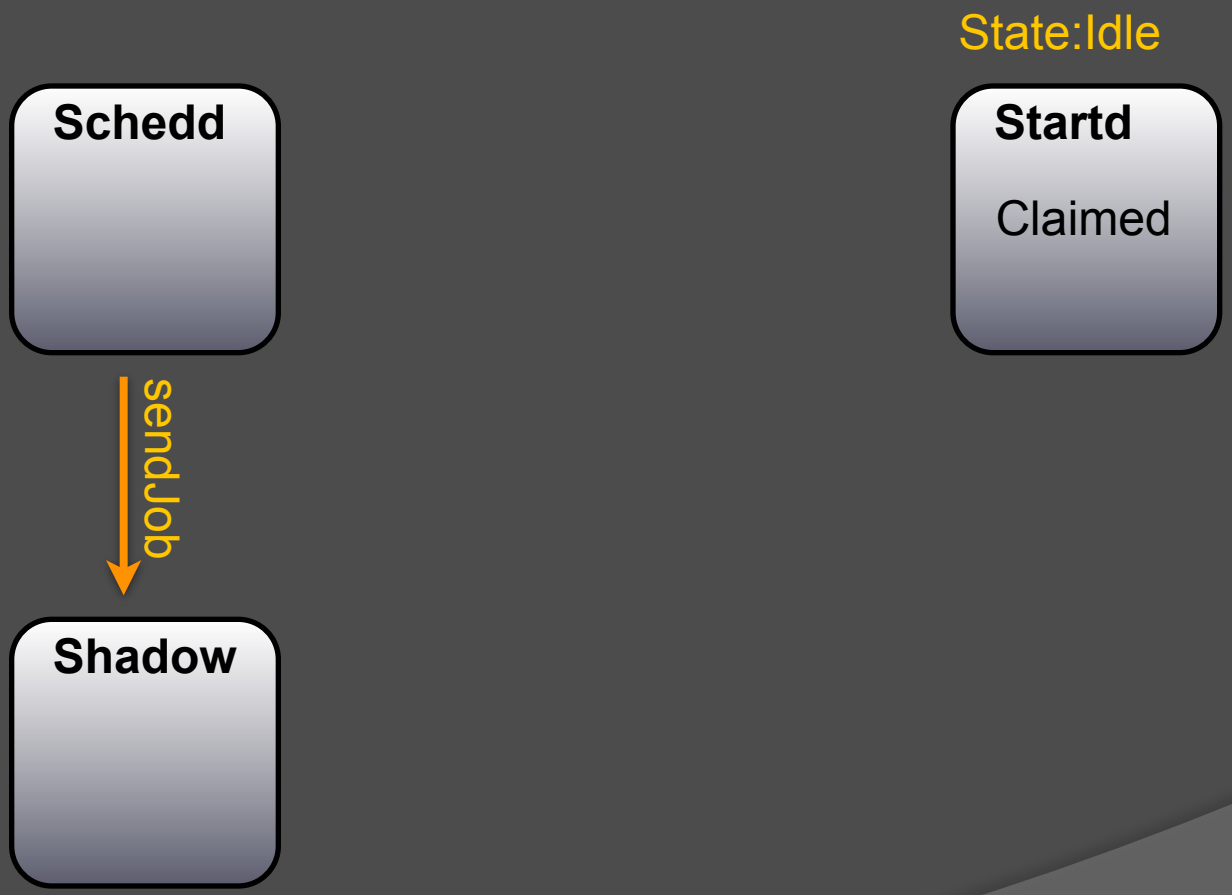
Current sandbox transfer

Existing Claim

State:Idle

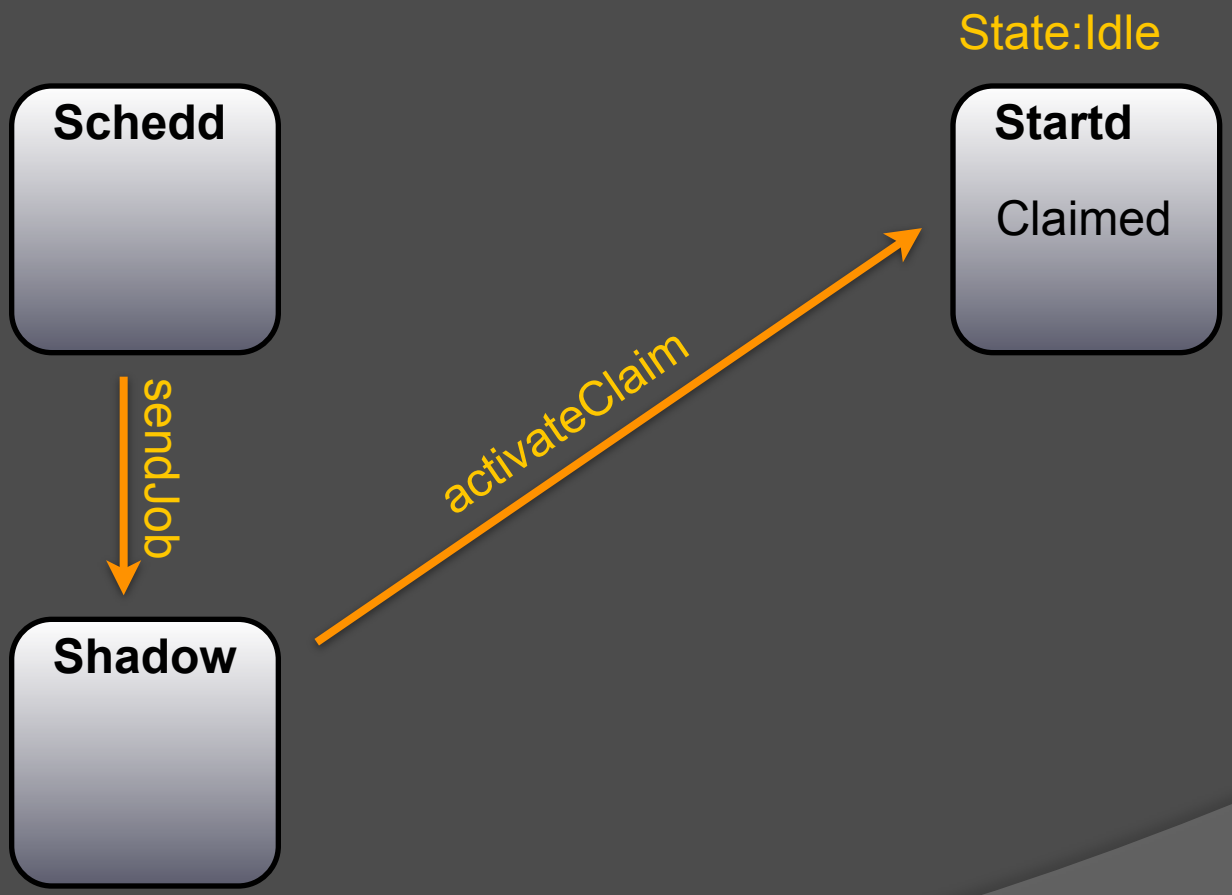
Current sandbox transfer

Existing Claim



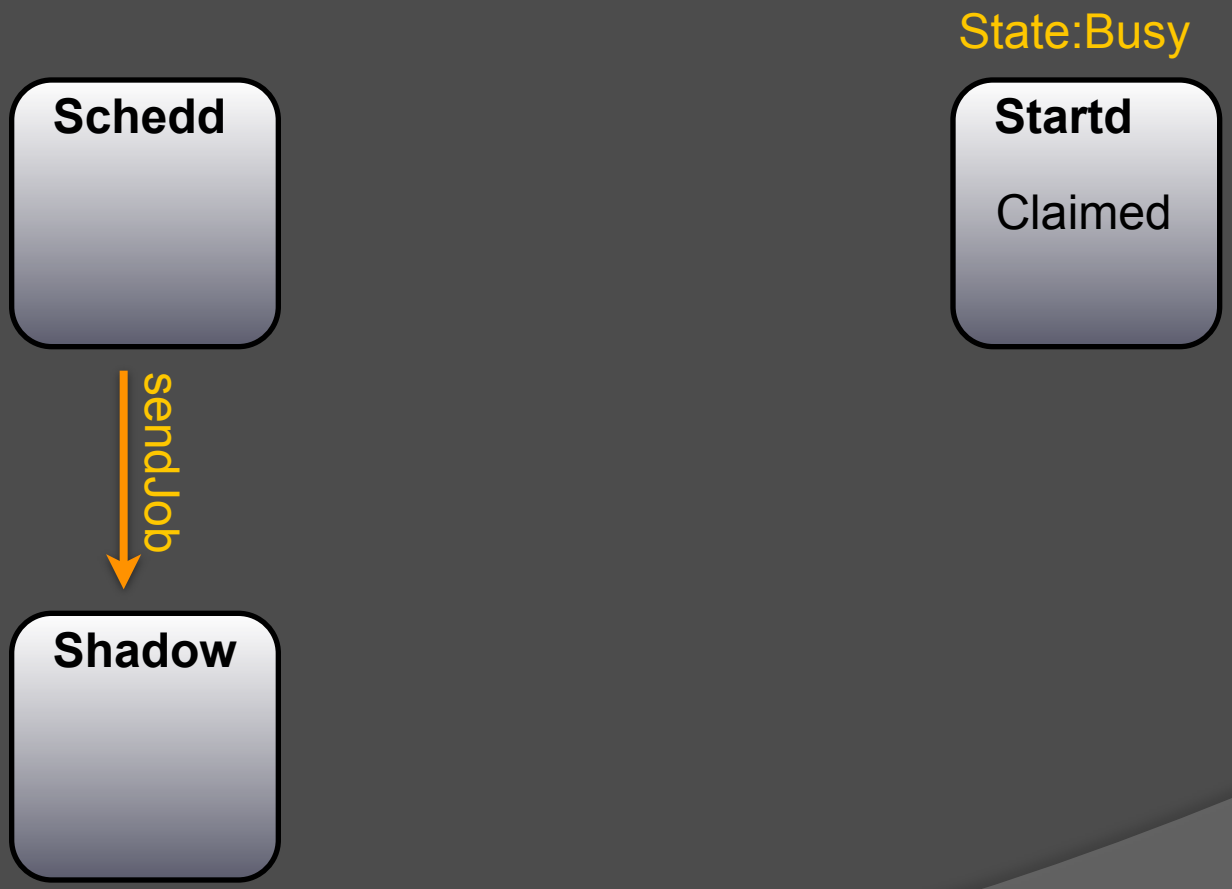
Current sandbox transfer

Existing Claim



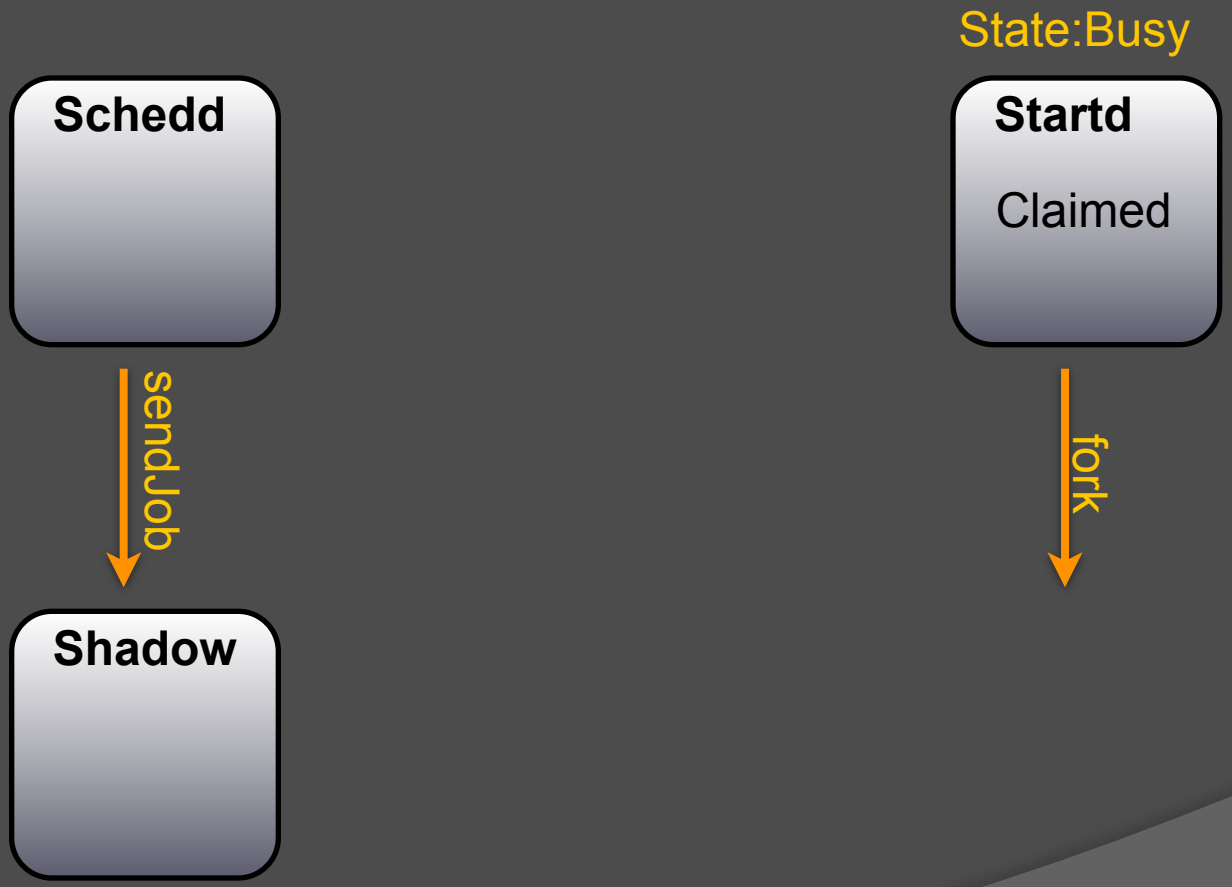
Current sandbox transfer

Existing Claim



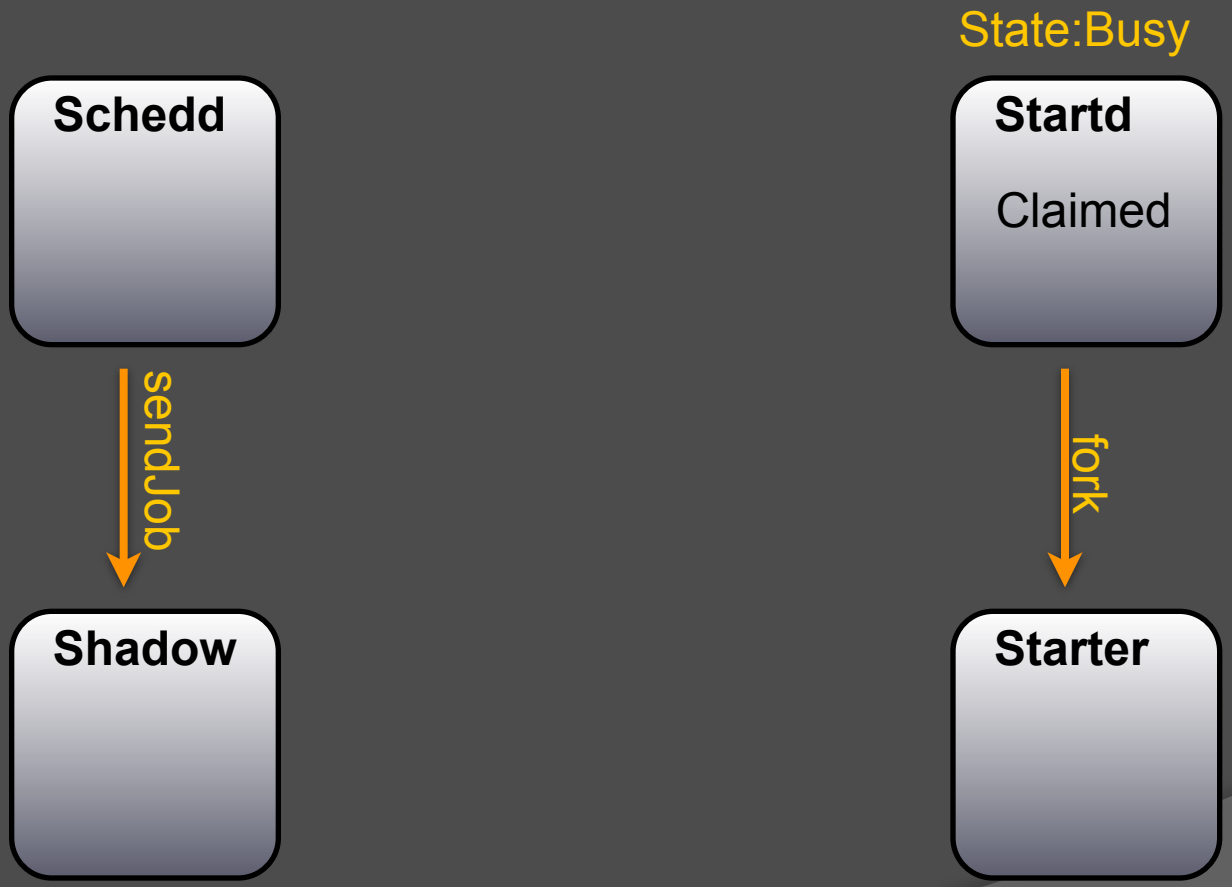
Current sandbox transfer

Existing Claim



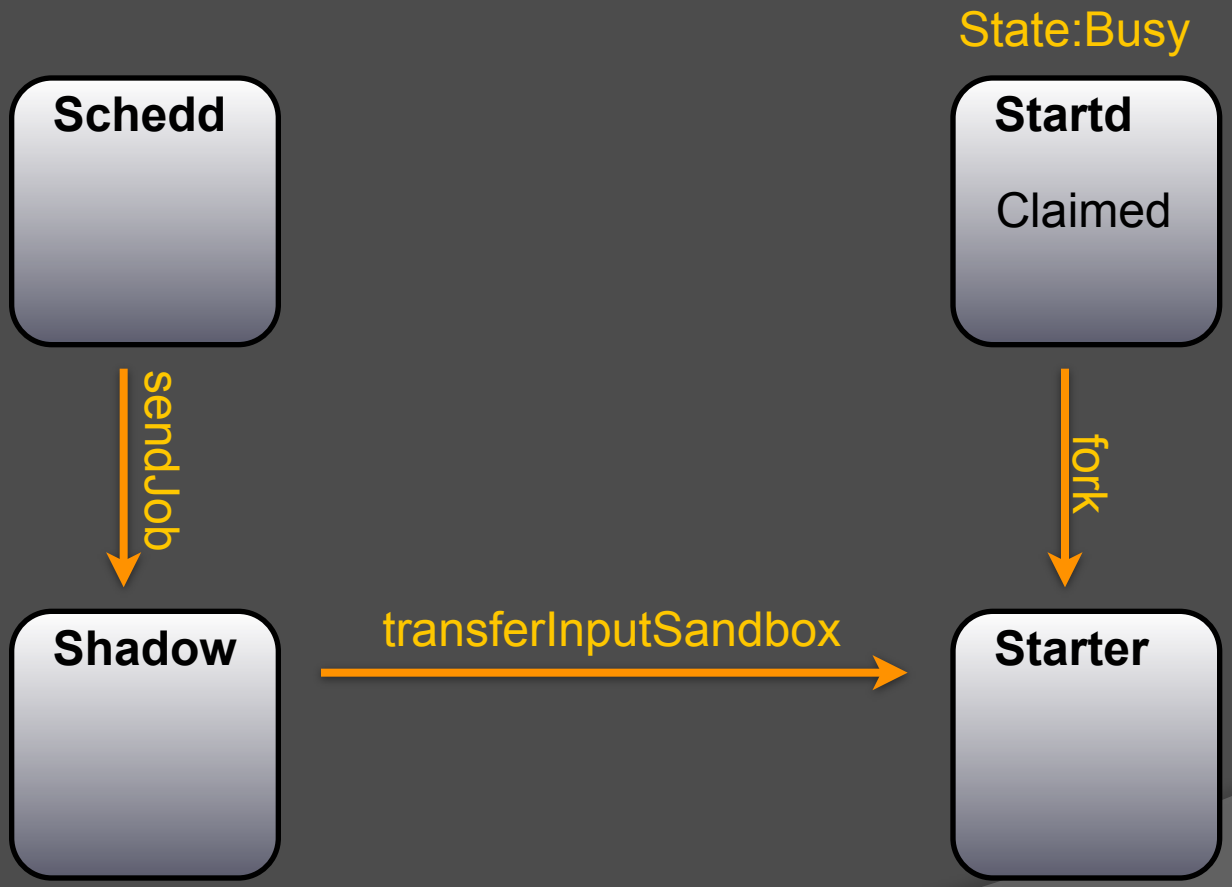
Current sandbox transfer

Existing Claim



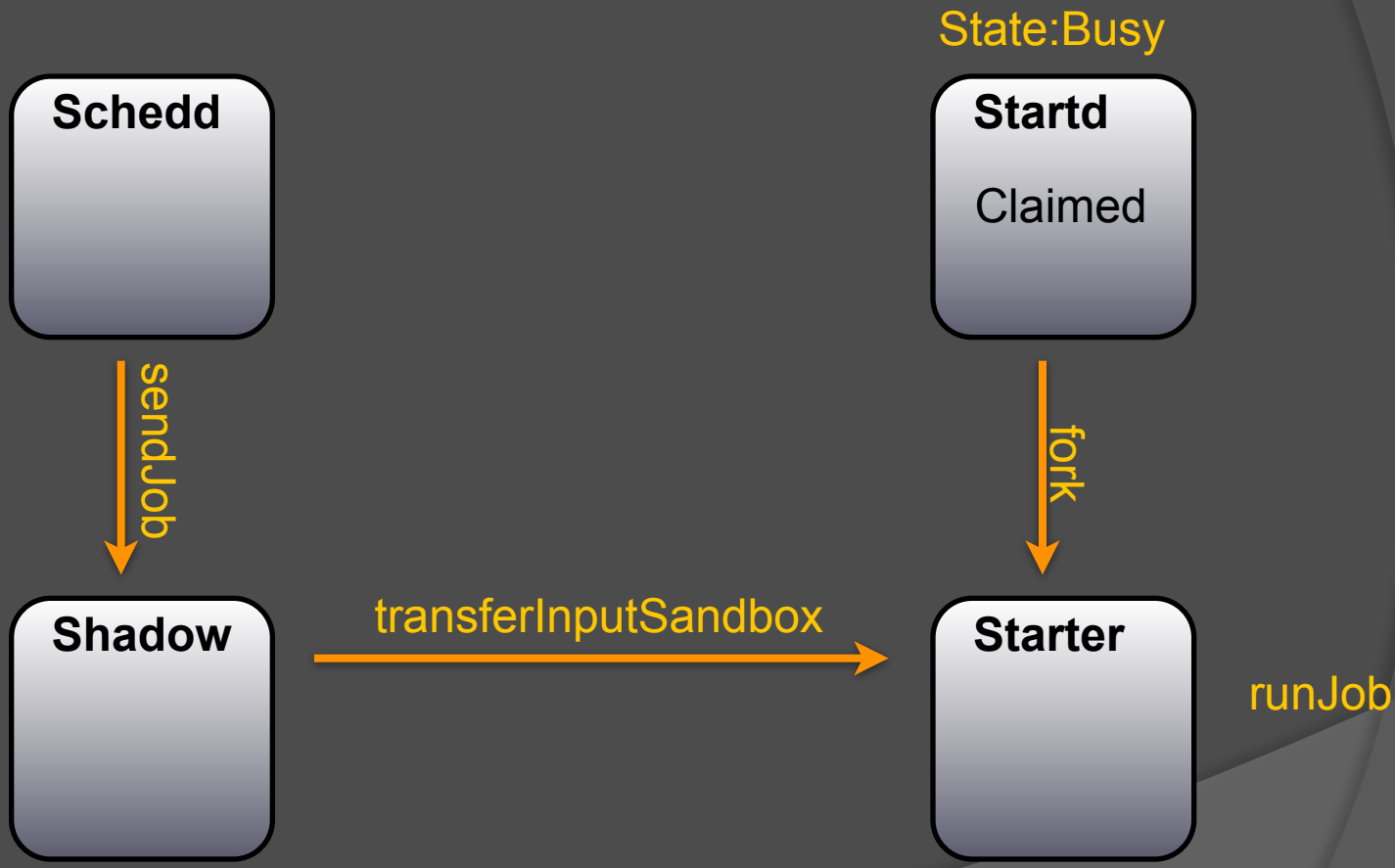
Current sandbox transfer

Existing Claim



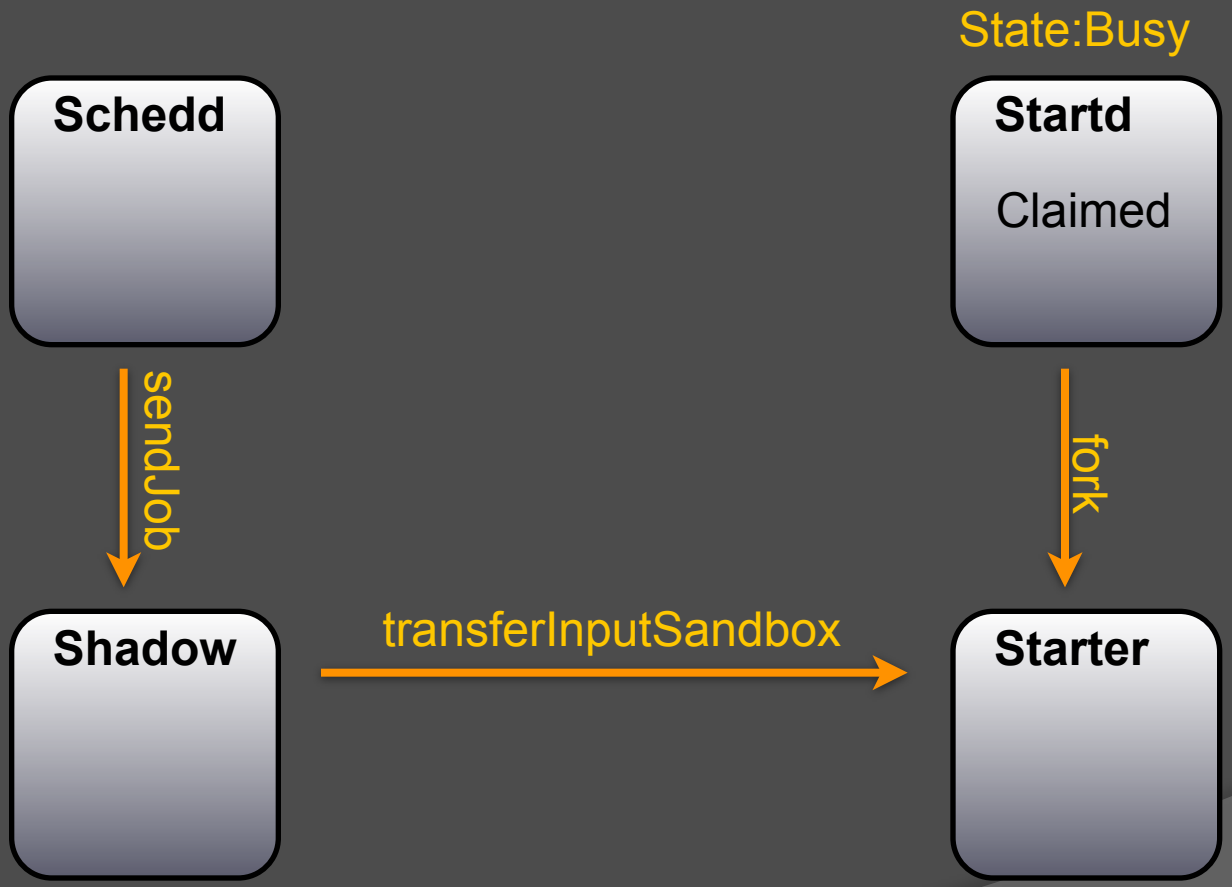
Current sandbox transfer

Existing Claim



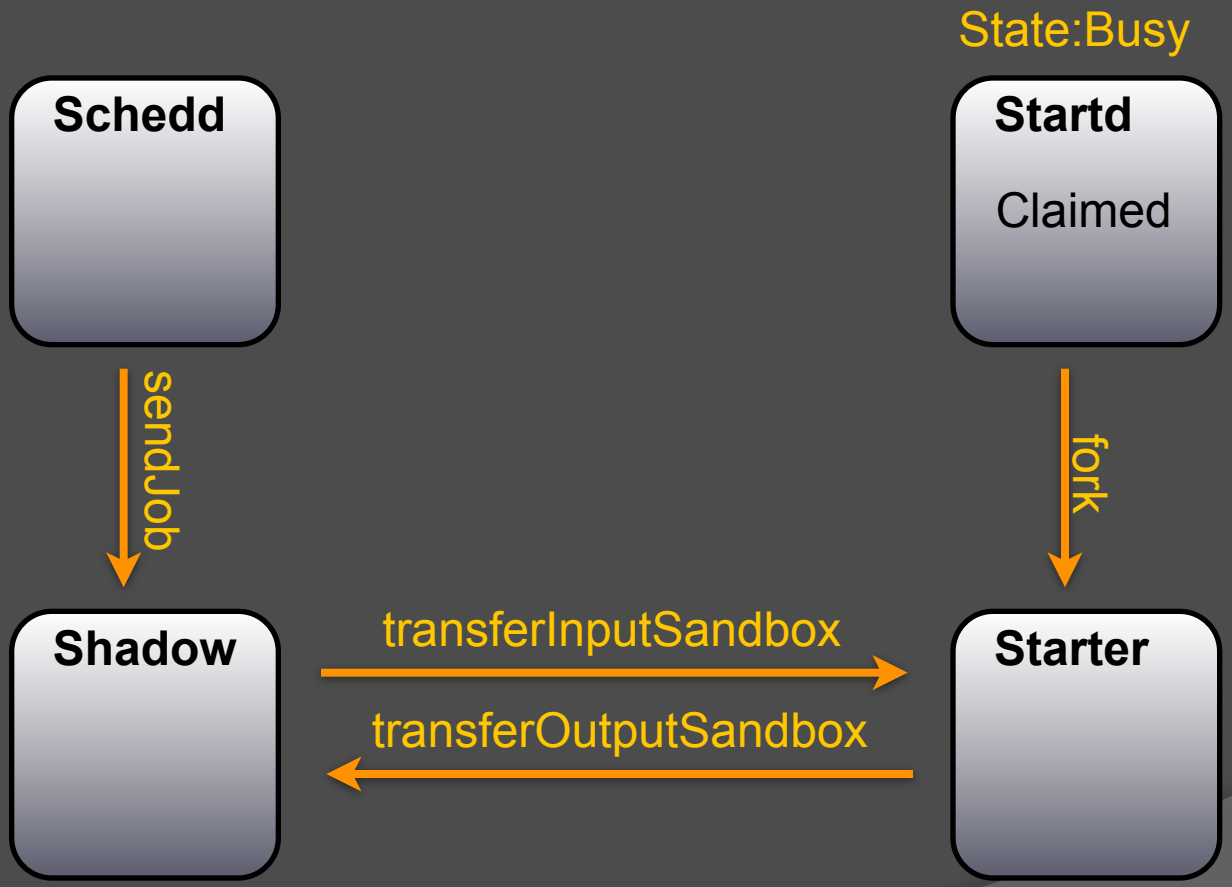
Current sandbox transfer

Existing Claim



Current sandbox transfer

Existing Claim

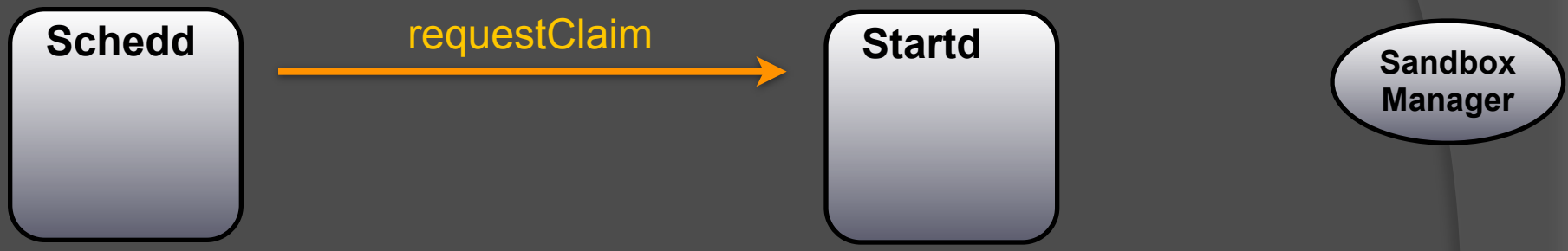


Asynchronous sandbox transfer

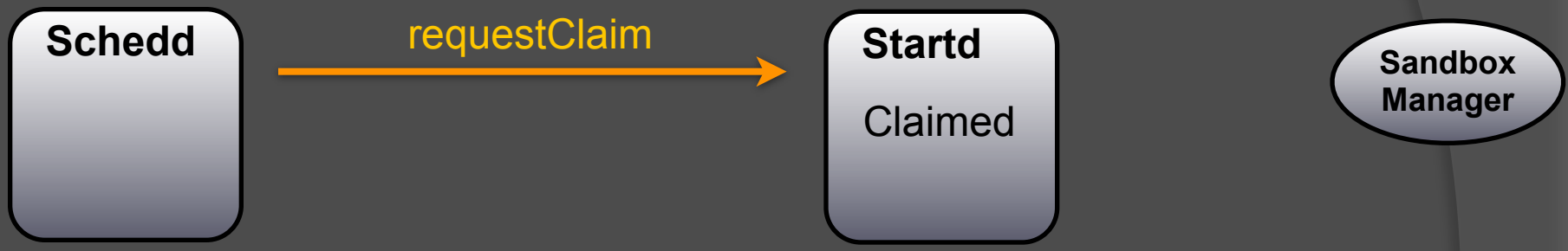
Asynchronous sandbox transfer

Asynchronous sandbox transfer

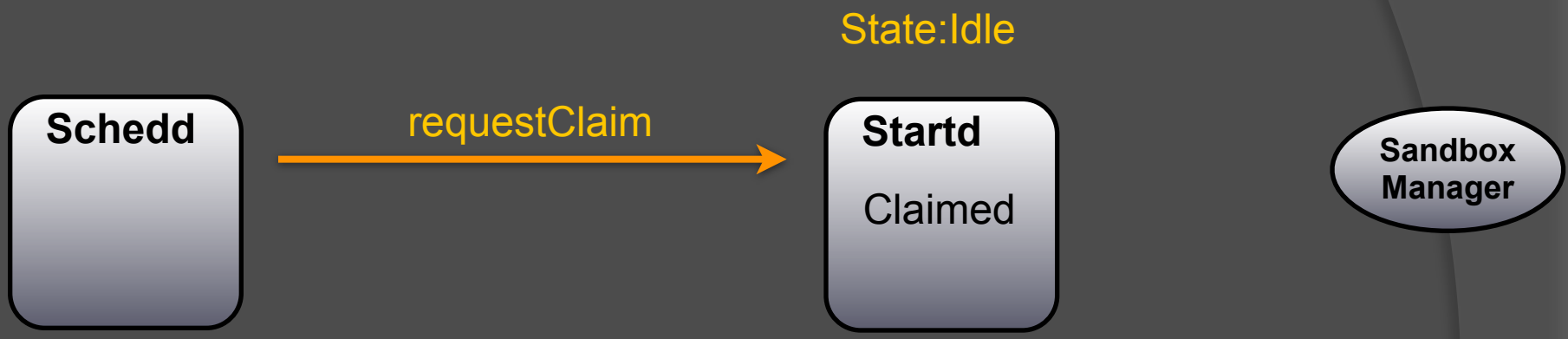
Asynchronous sandbox transfer



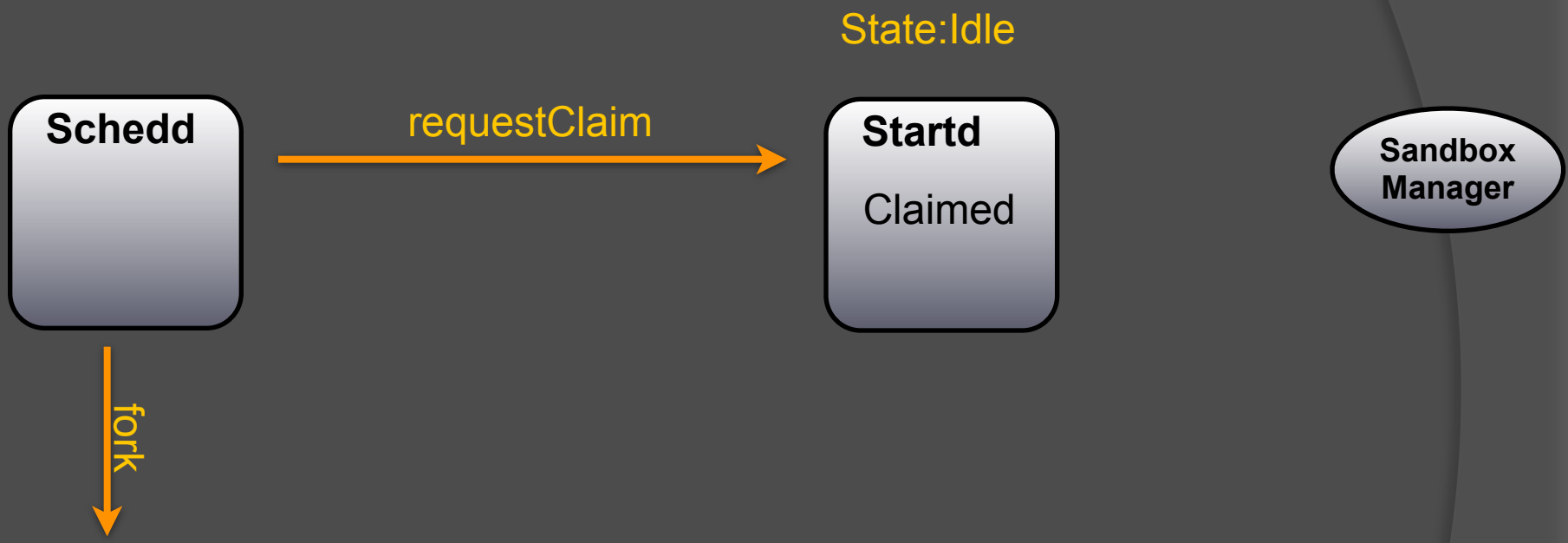
Asynchronous sandbox transfer



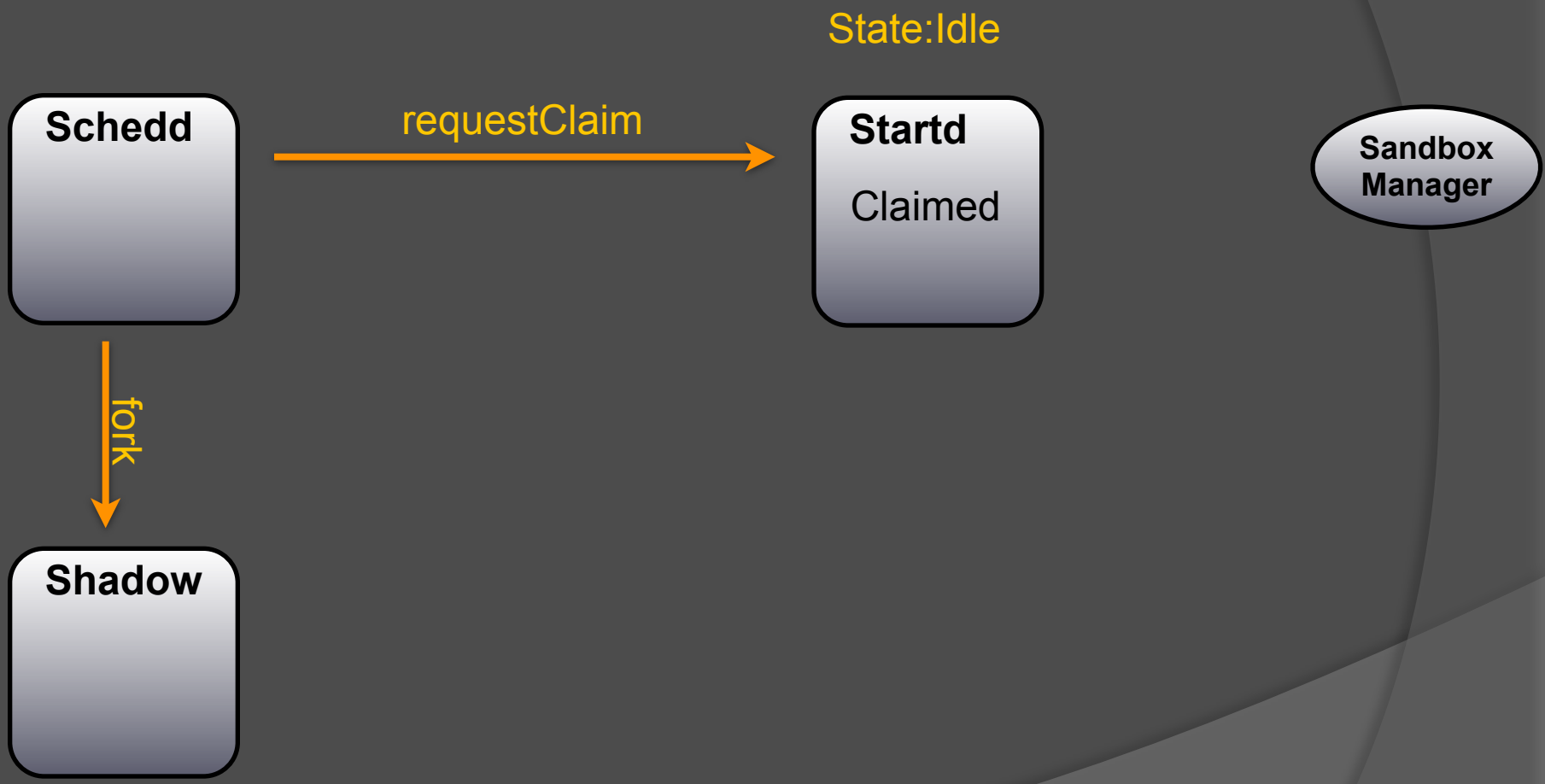
Asynchronous sandbox transfer



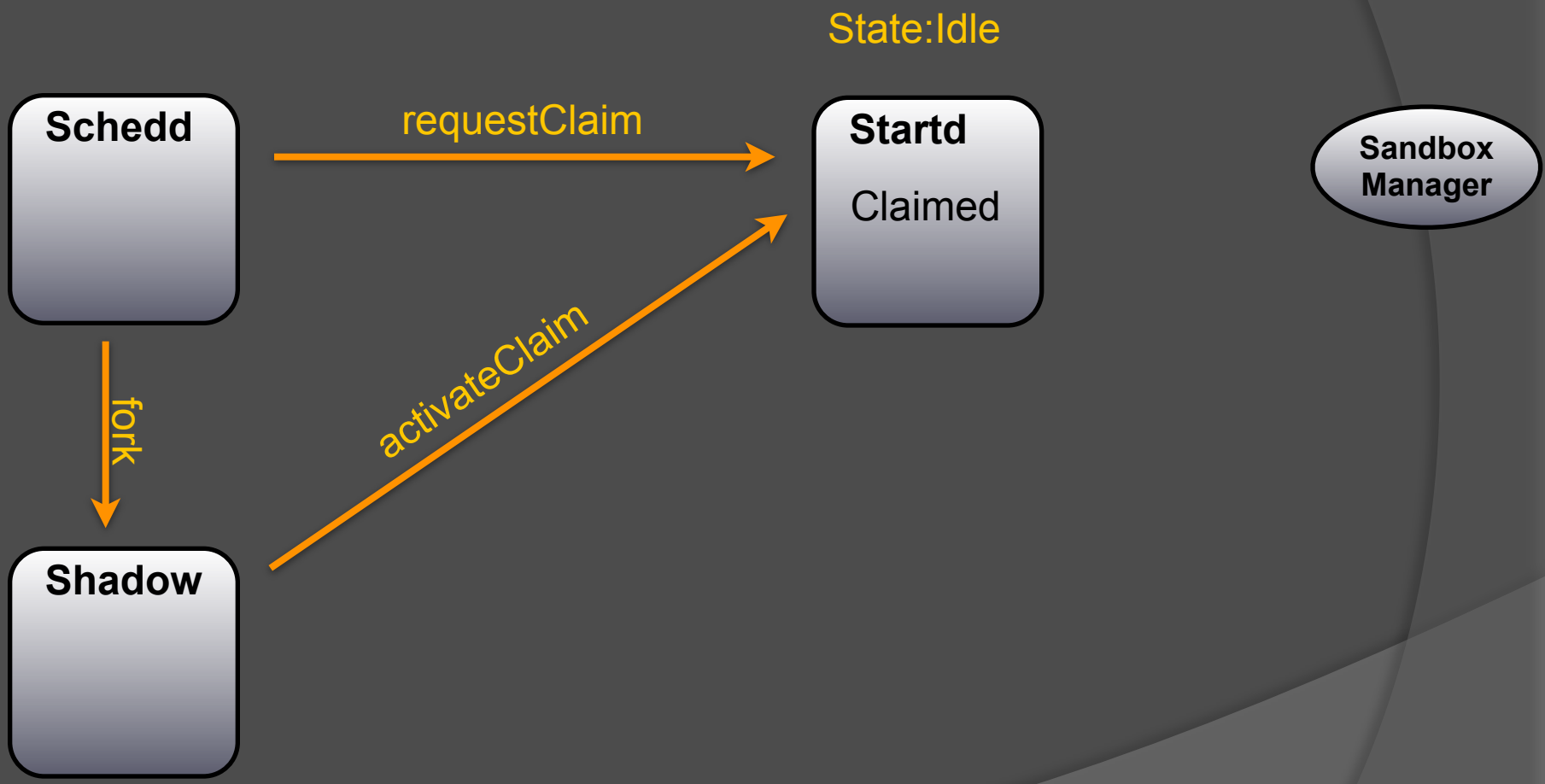
Asynchronous sandbox transfer



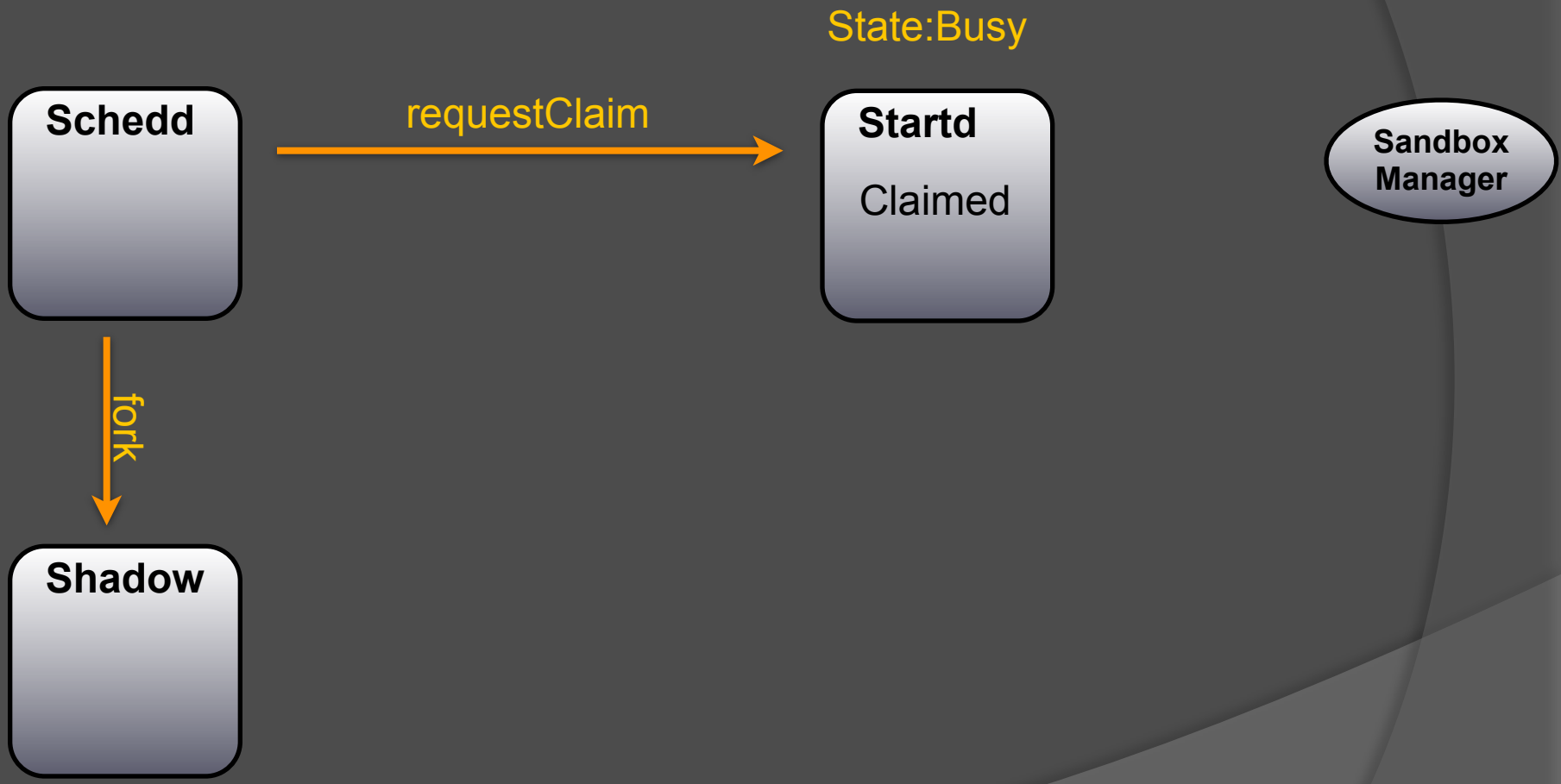
Asynchronous sandbox transfer



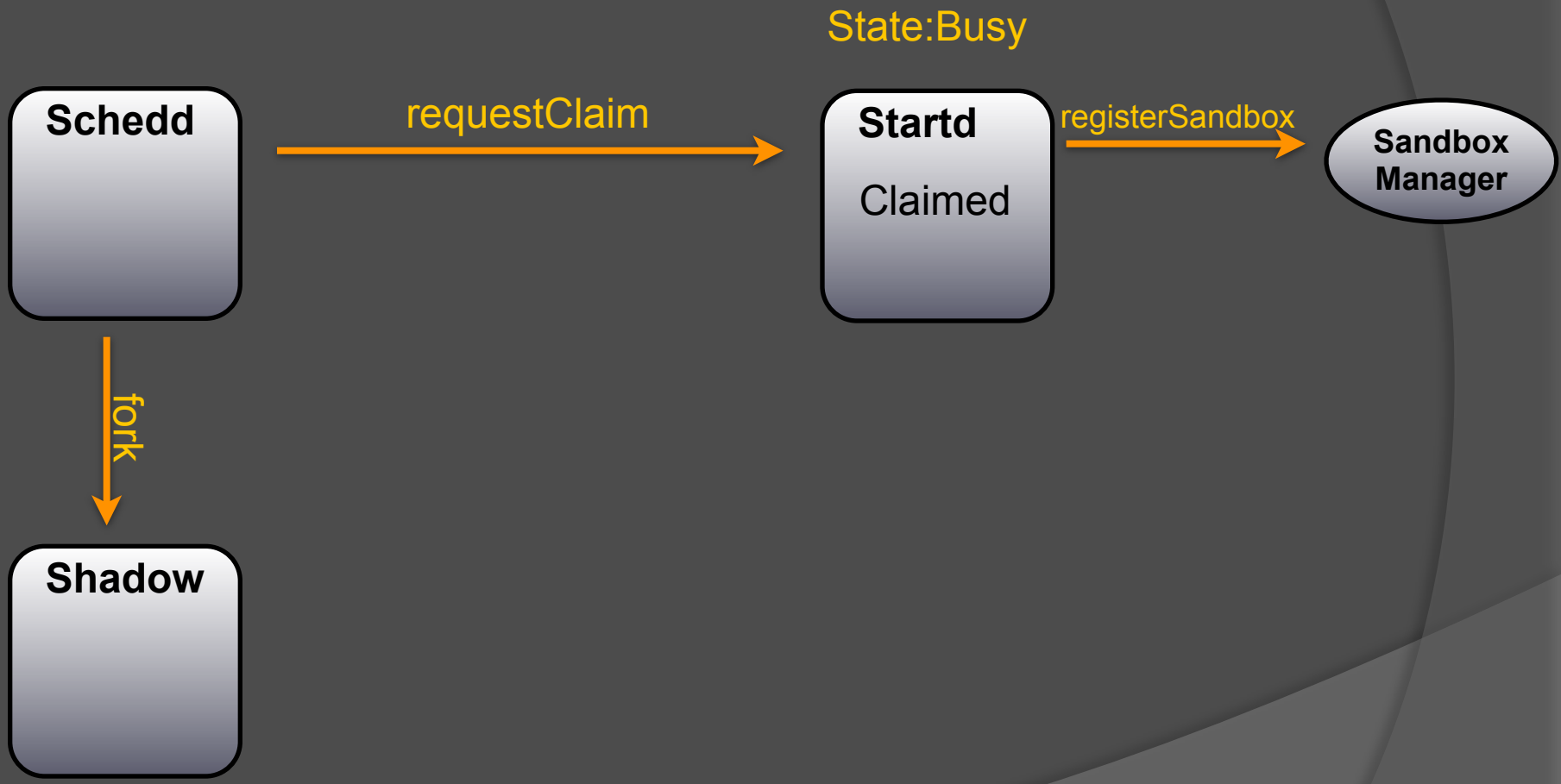
Asynchronous sandbox transfer



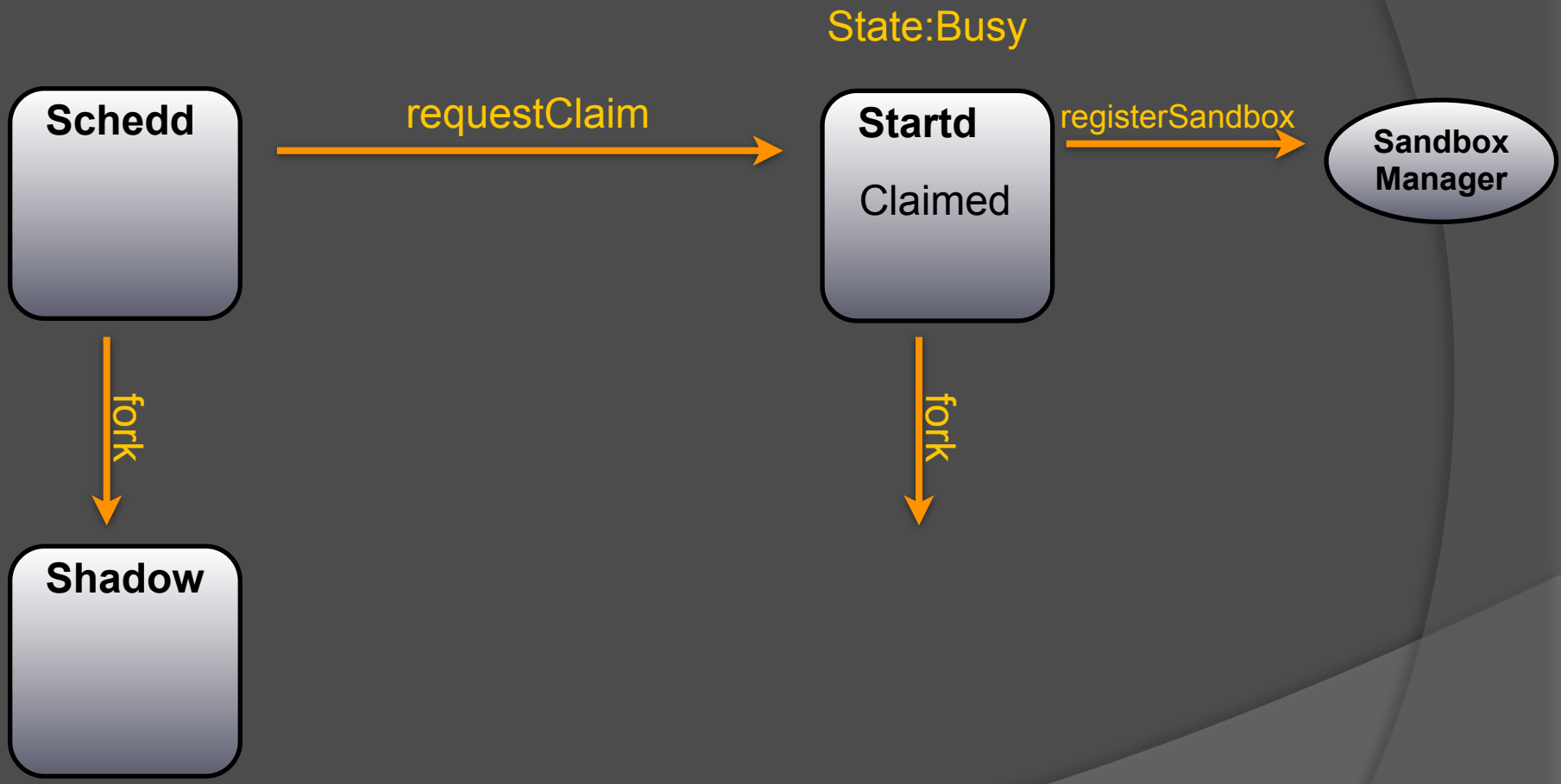
Asynchronous sandbox transfer



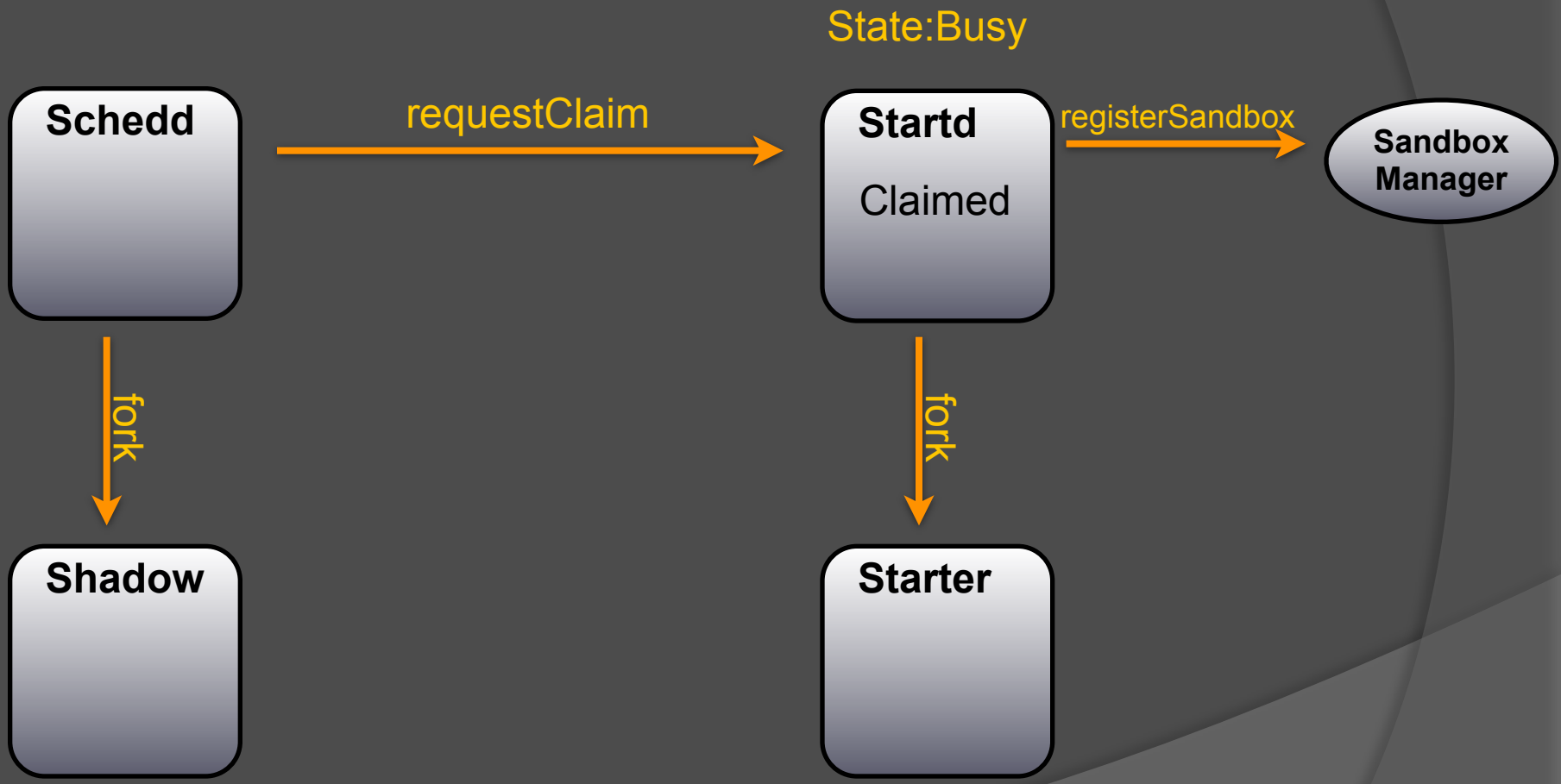
Asynchronous sandbox transfer



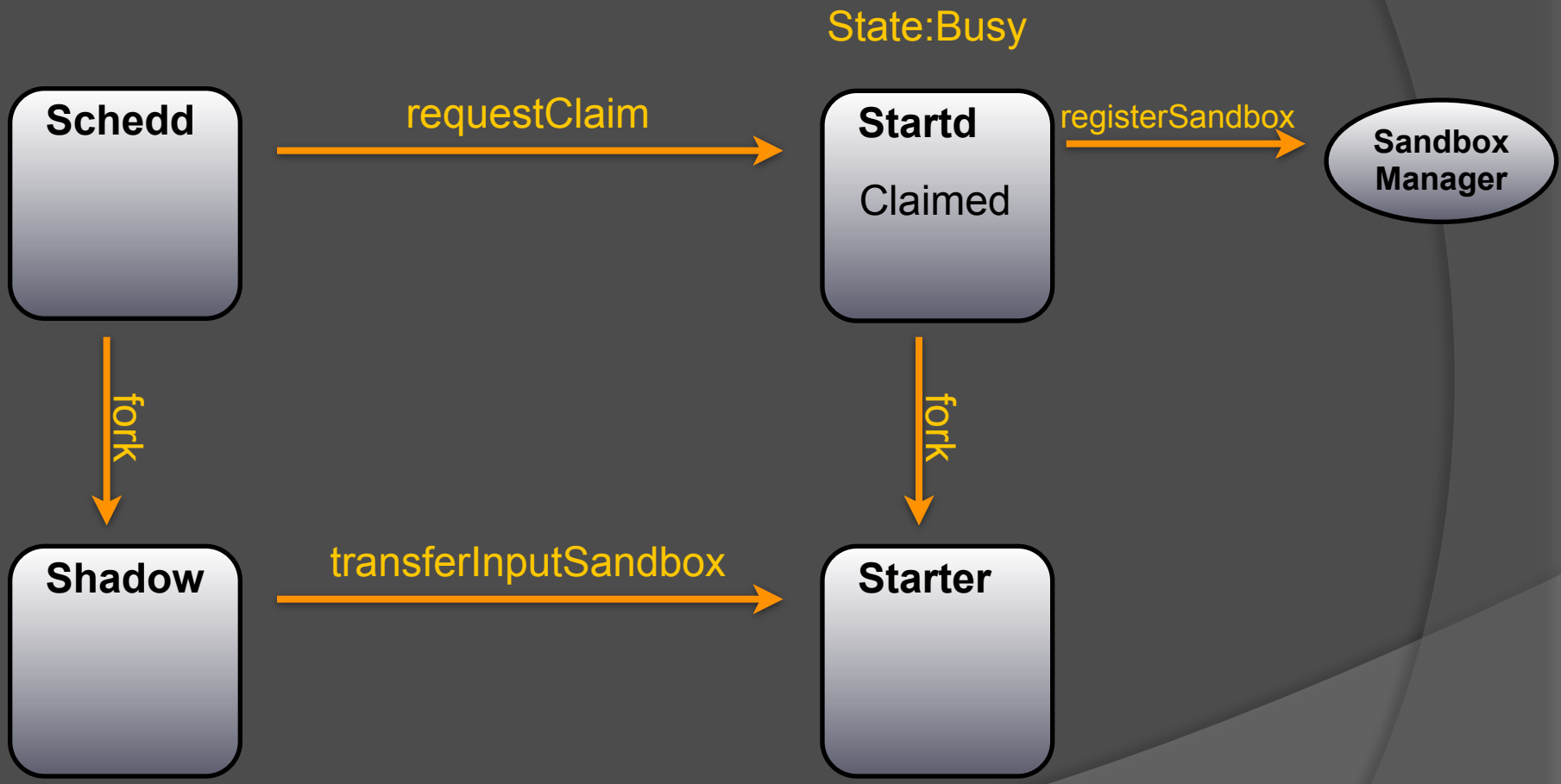
Asynchronous sandbox transfer



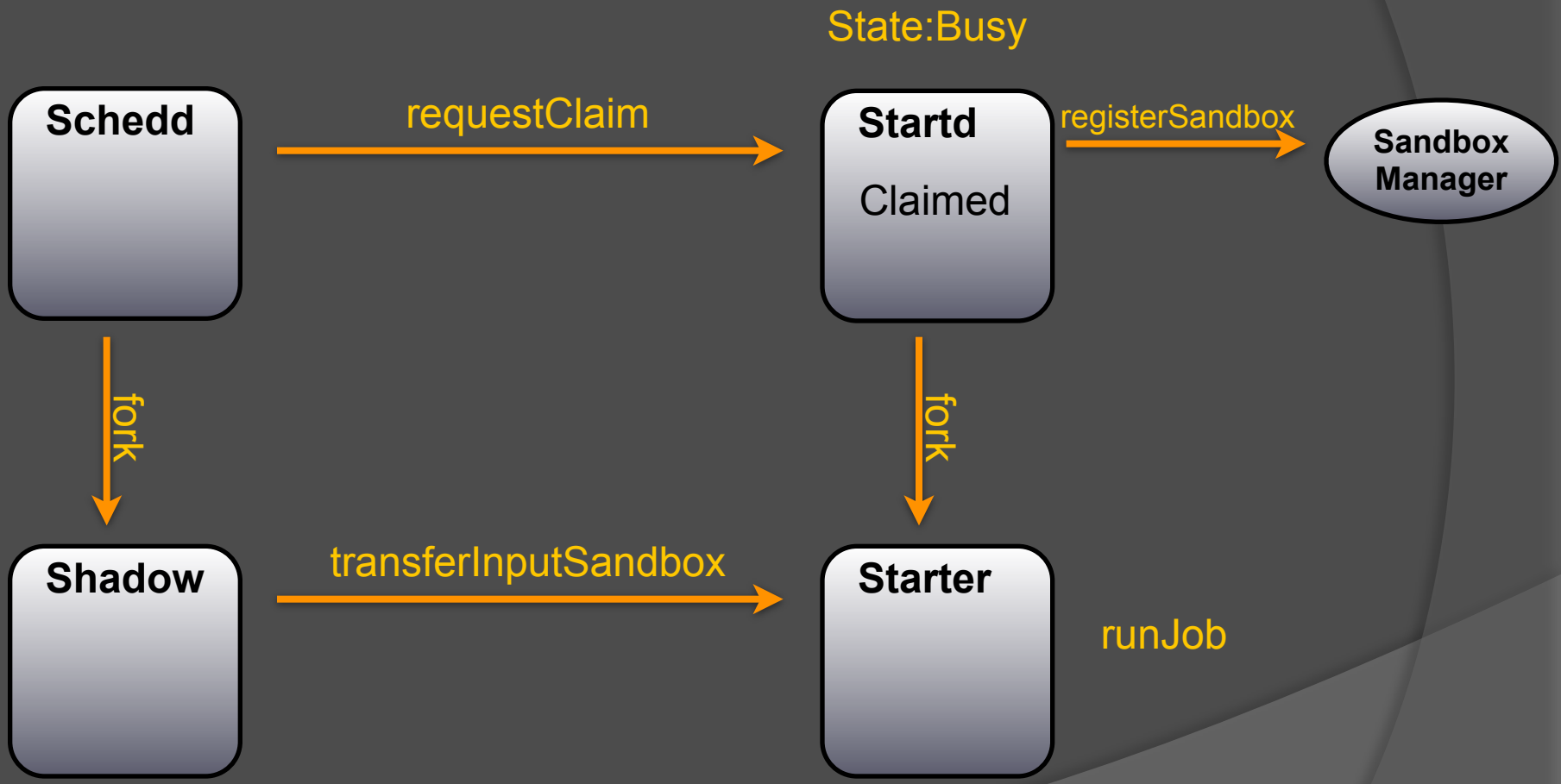
Asynchronous sandbox transfer



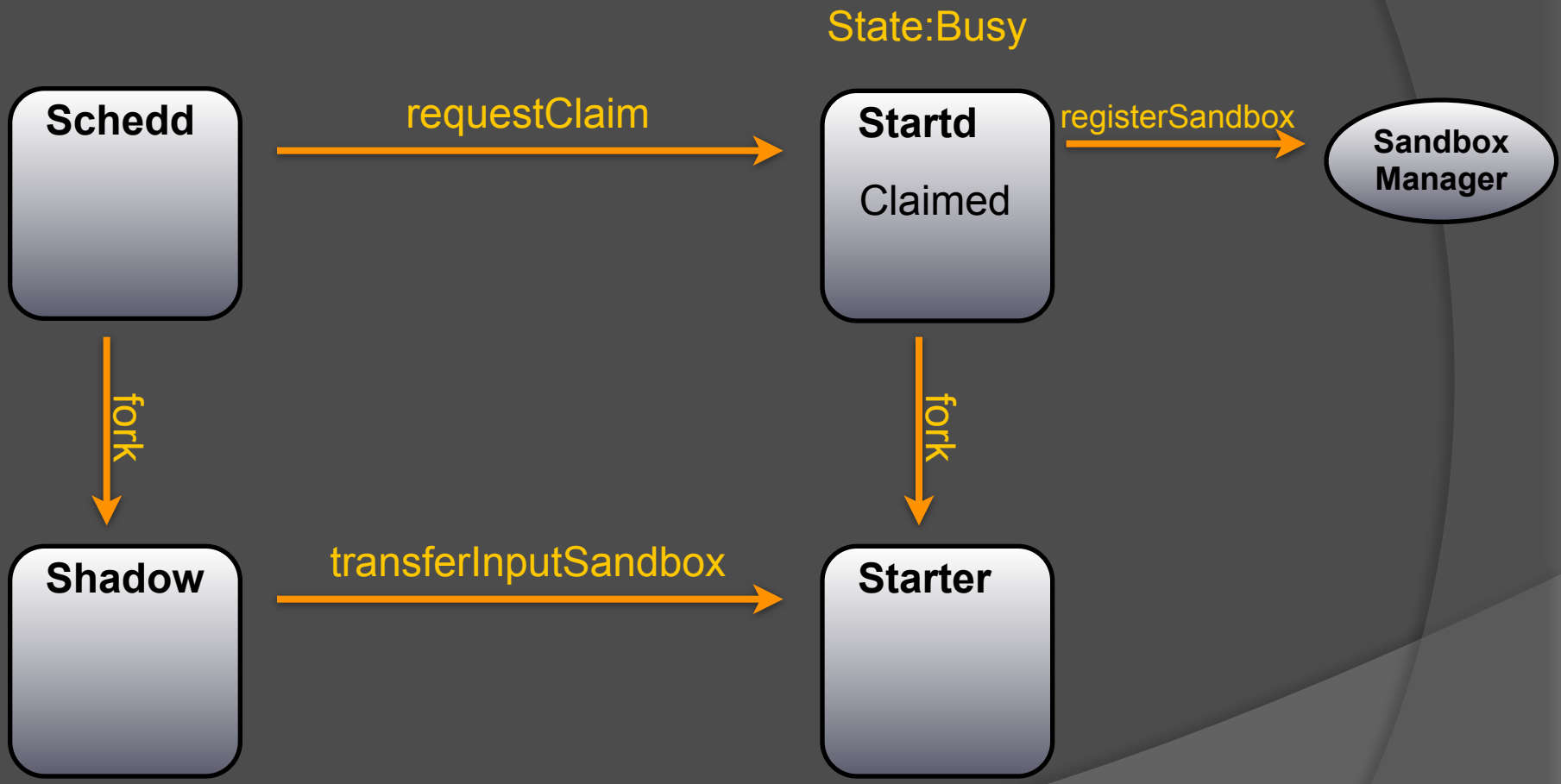
Asynchronous sandbox transfer



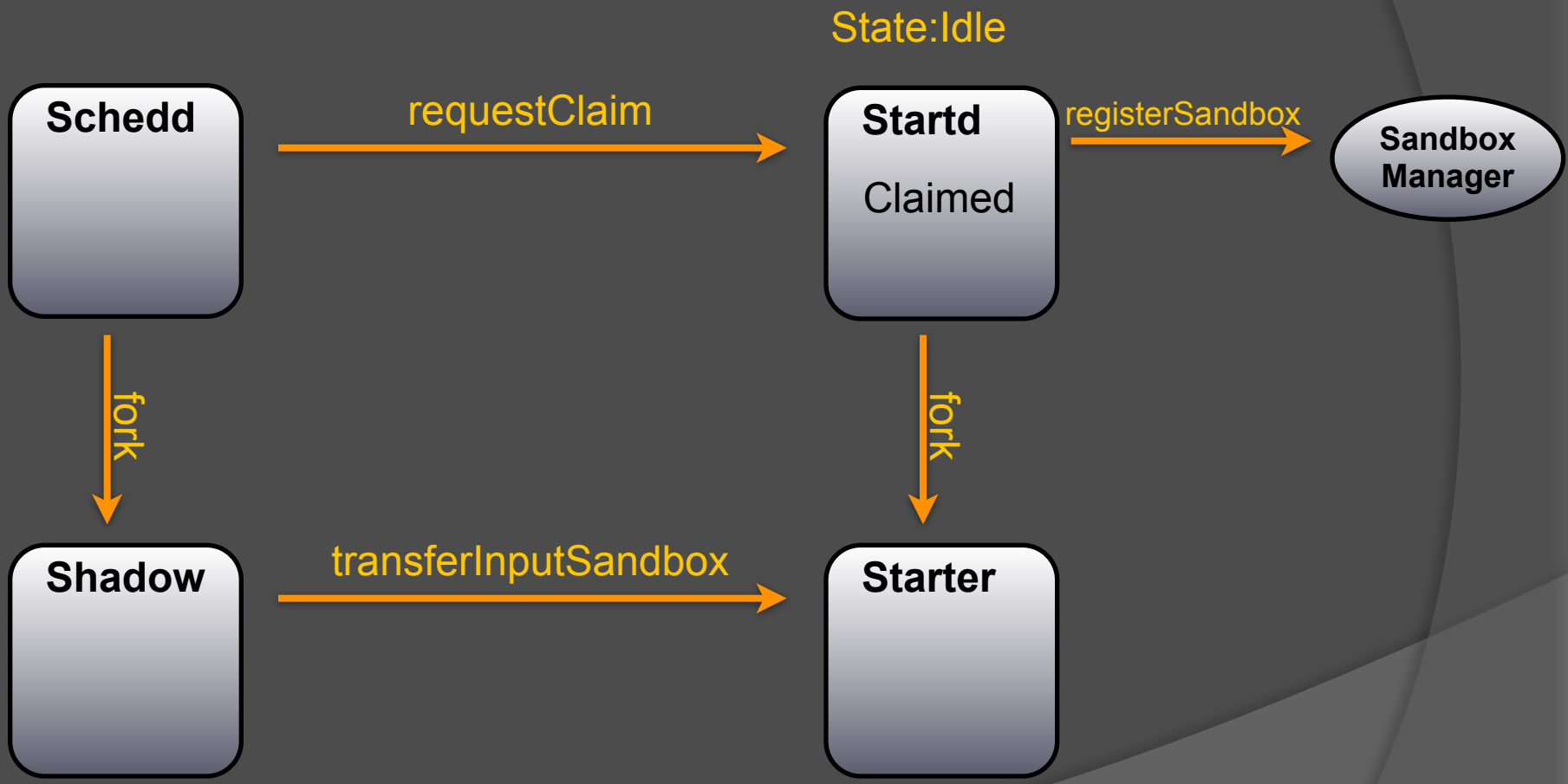
Asynchronous sandbox transfer



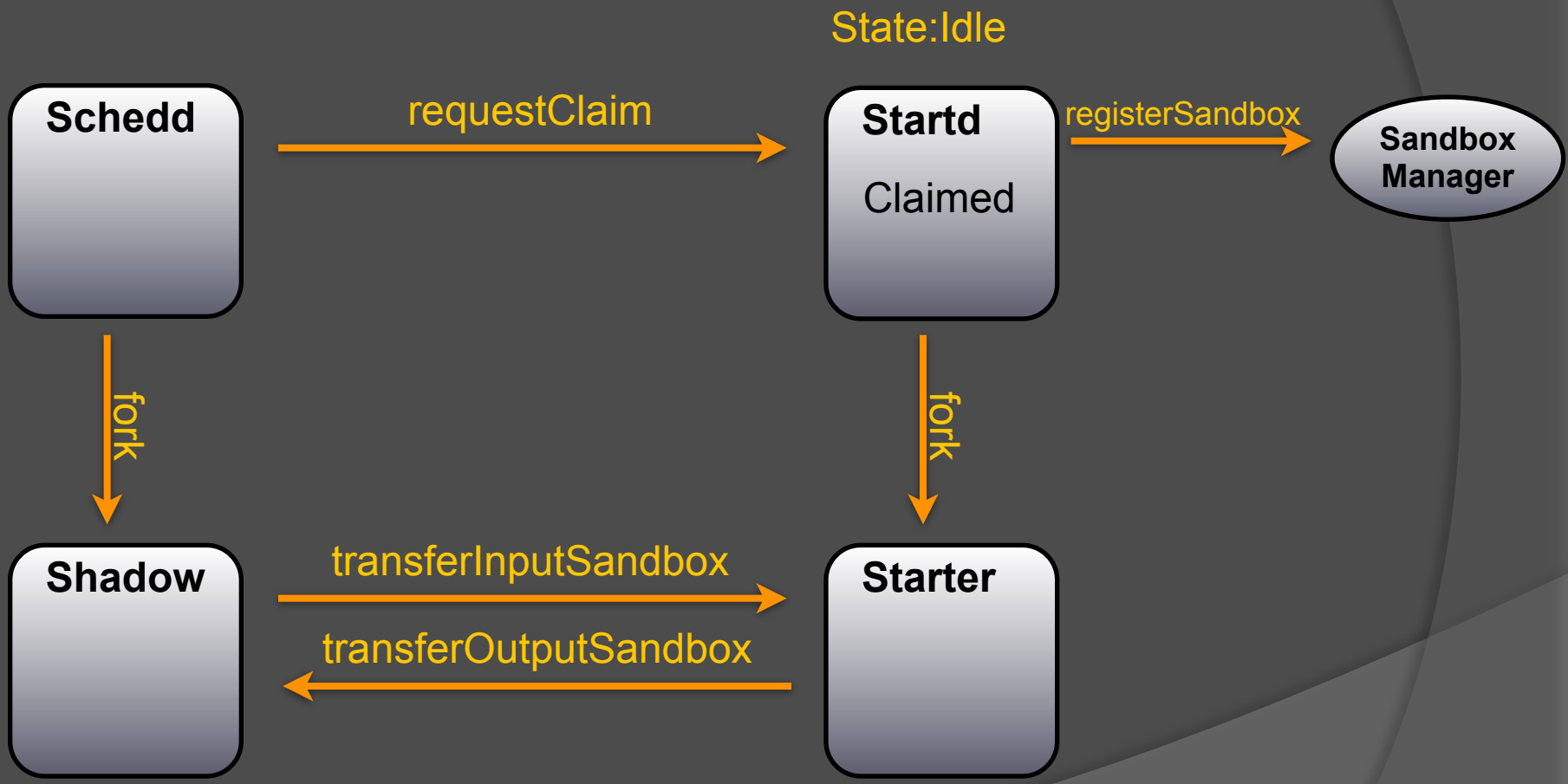
Asynchronous sandbox transfer



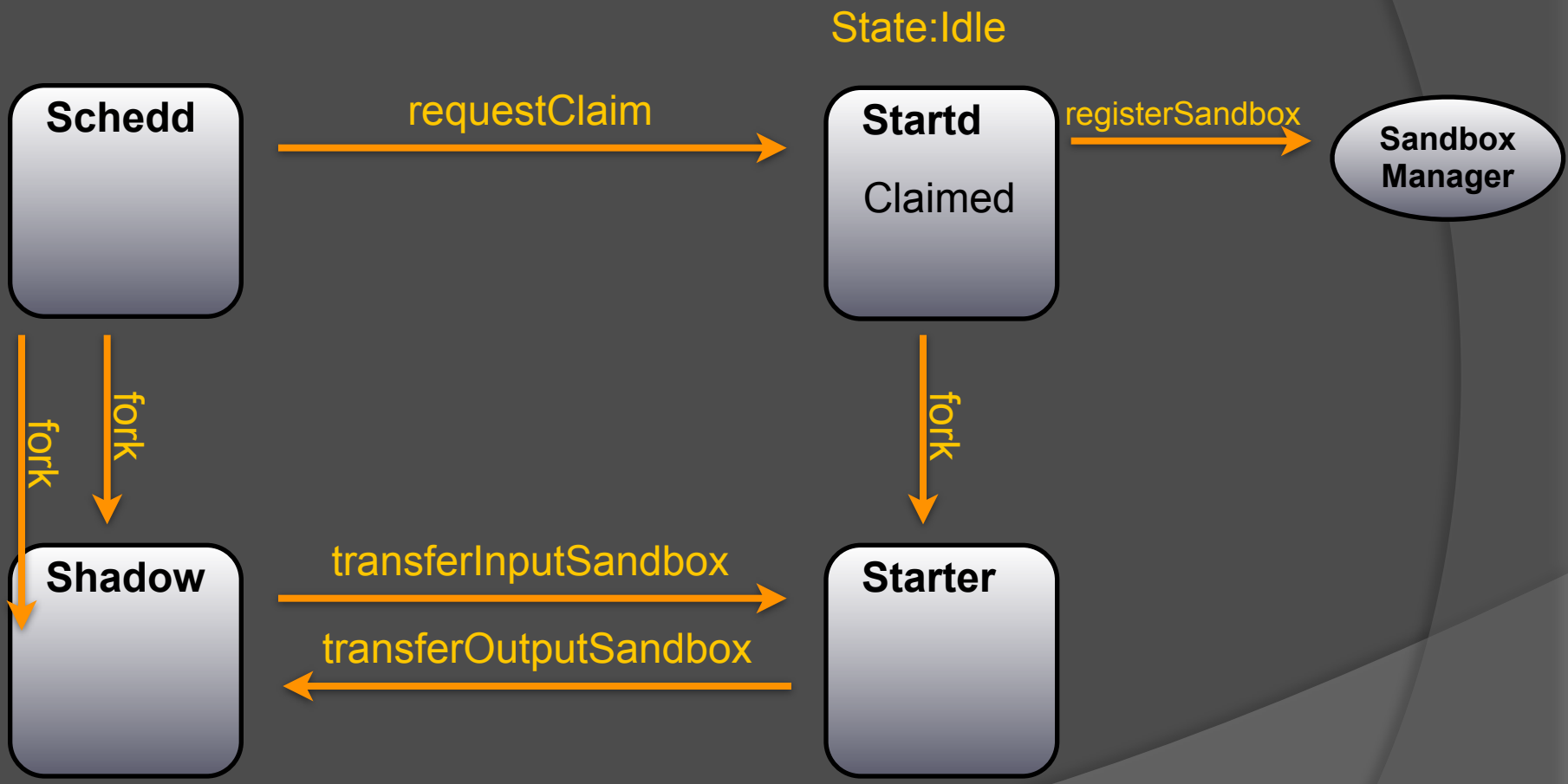
Asynchronous sandbox transfer



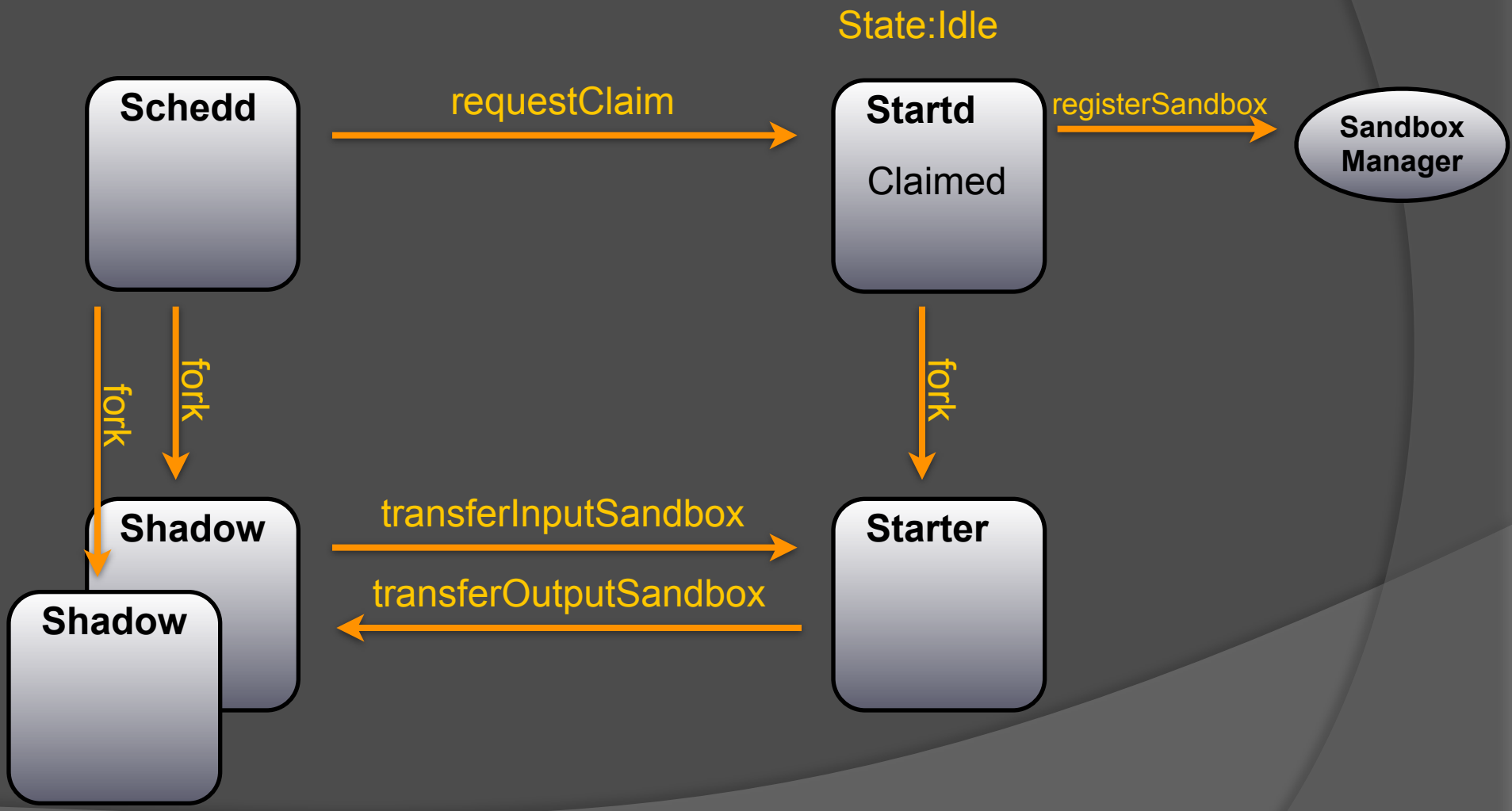
Asynchronous sandbox transfer



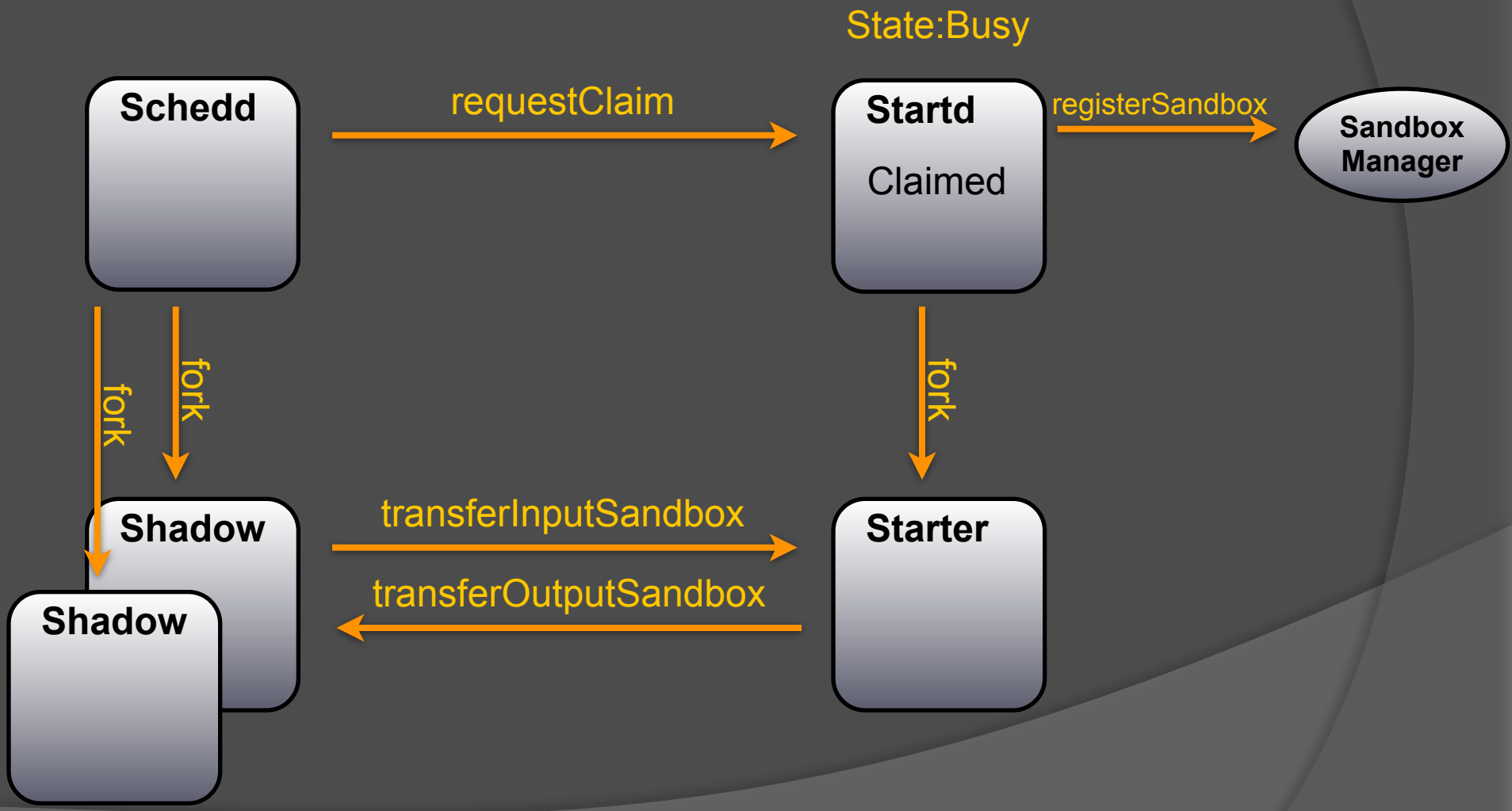
Asynchronous sandbox transfer



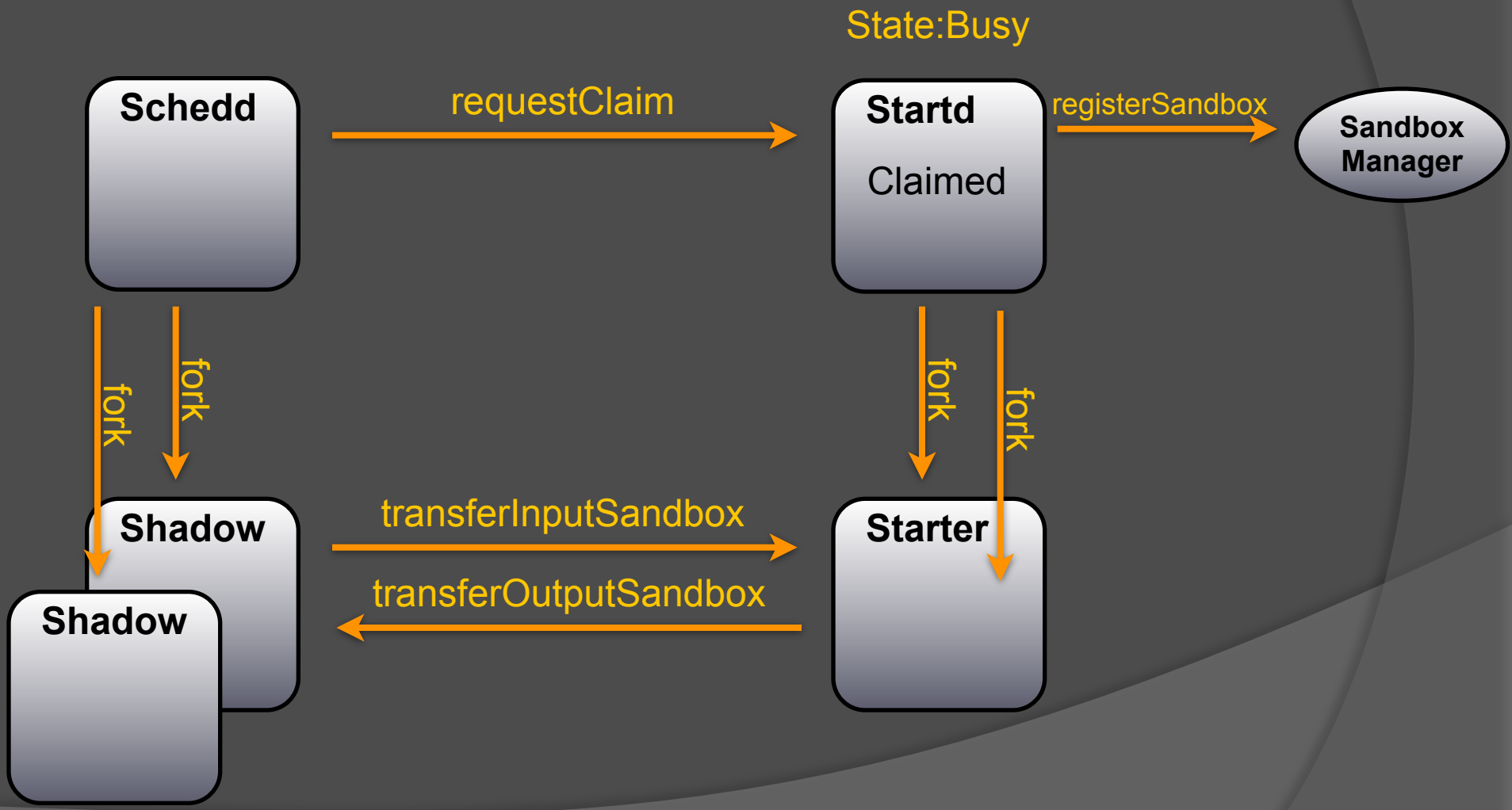
Asynchronous sandbox transfer



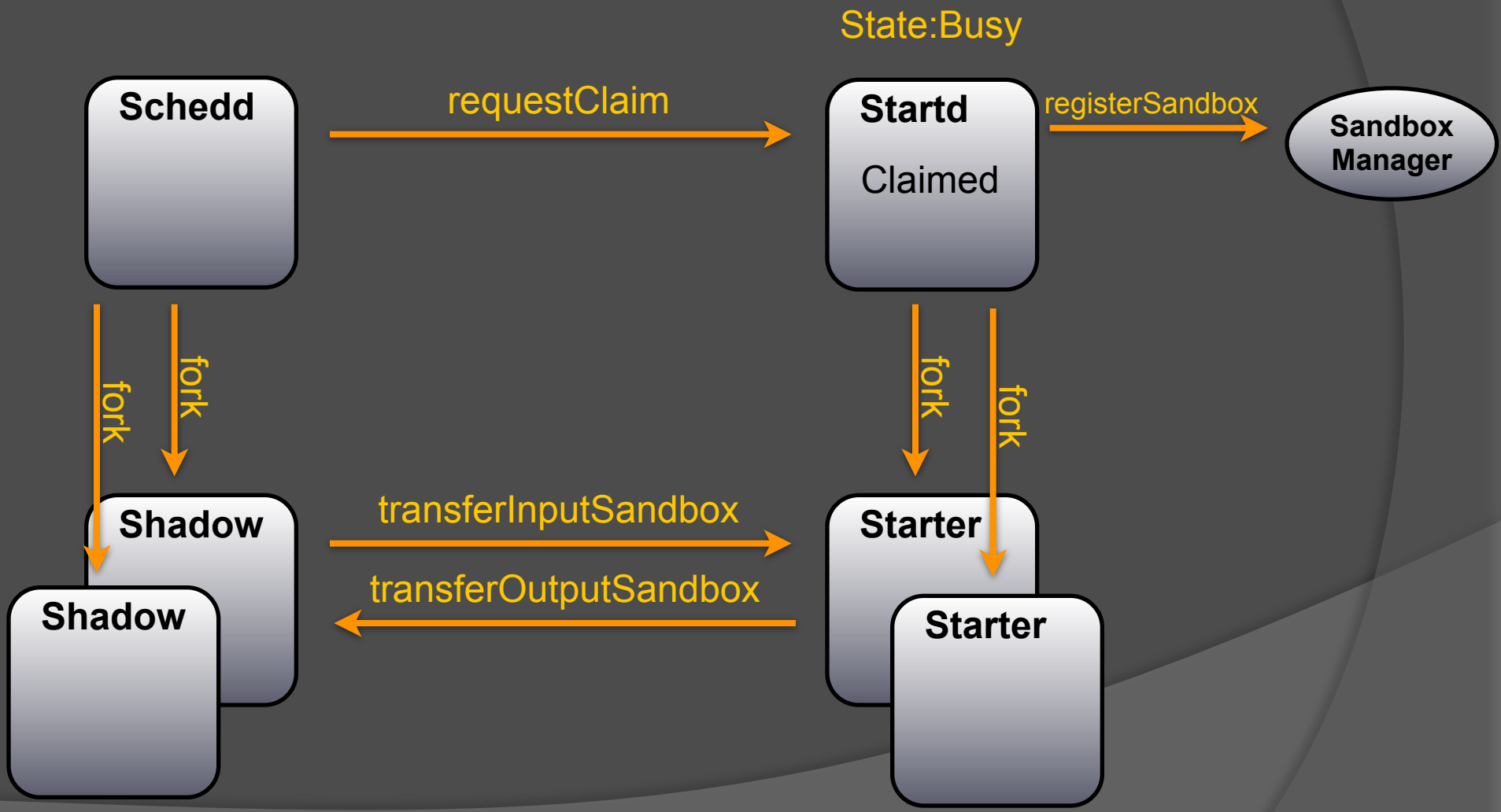
Asynchronous sandbox transfer



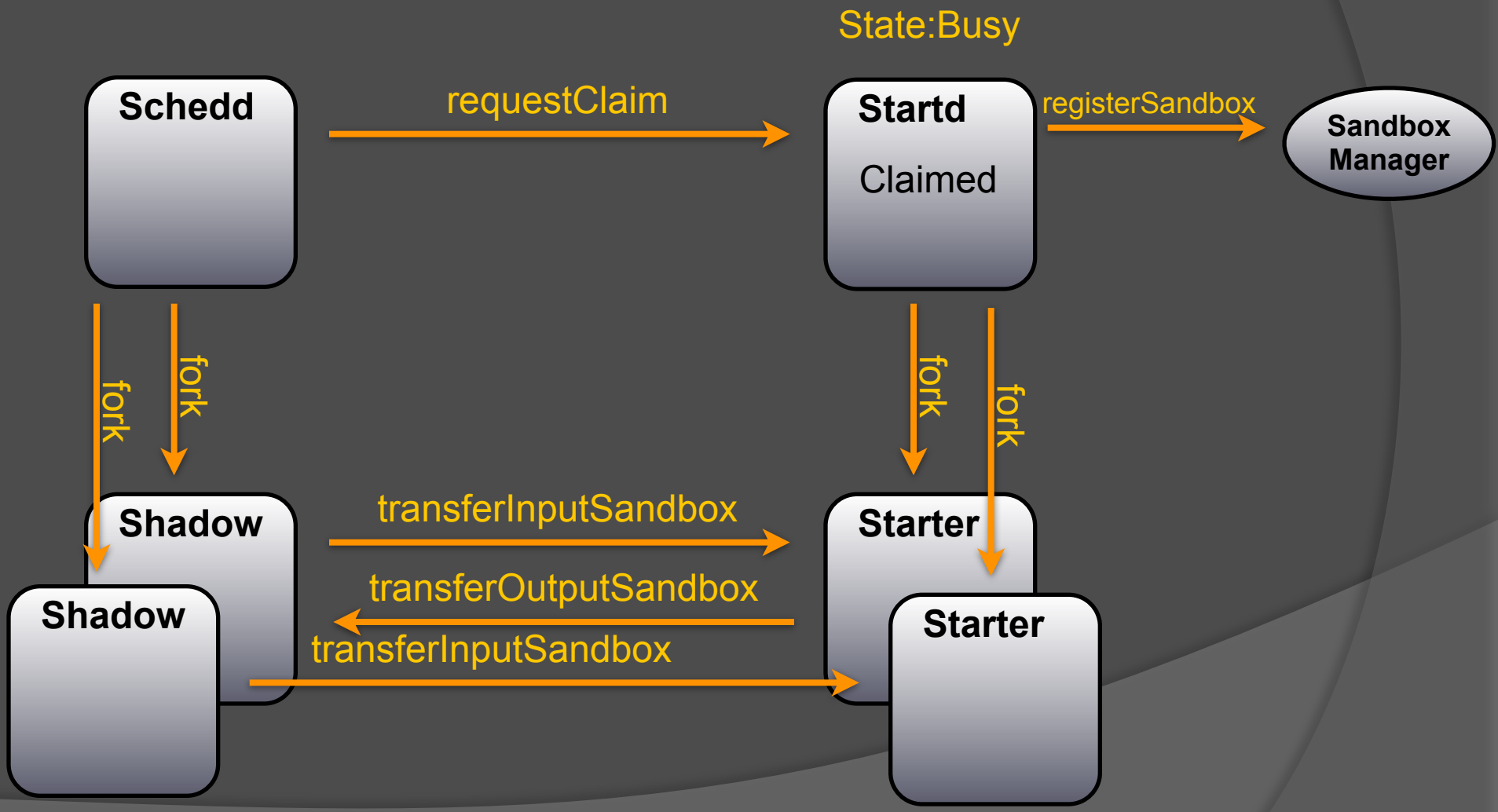
Asynchronous sandbox transfer



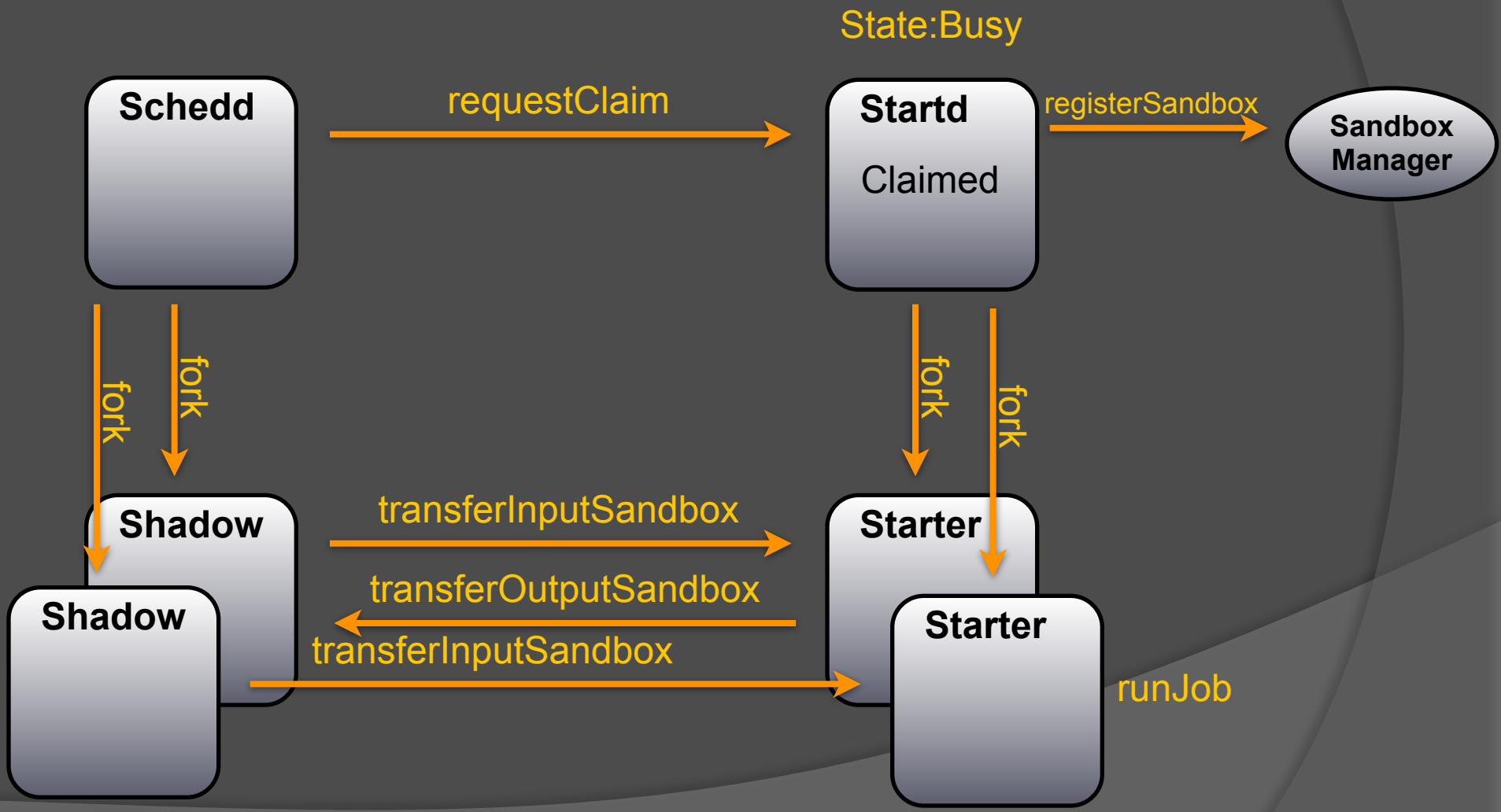
Asynchronous sandbox transfer



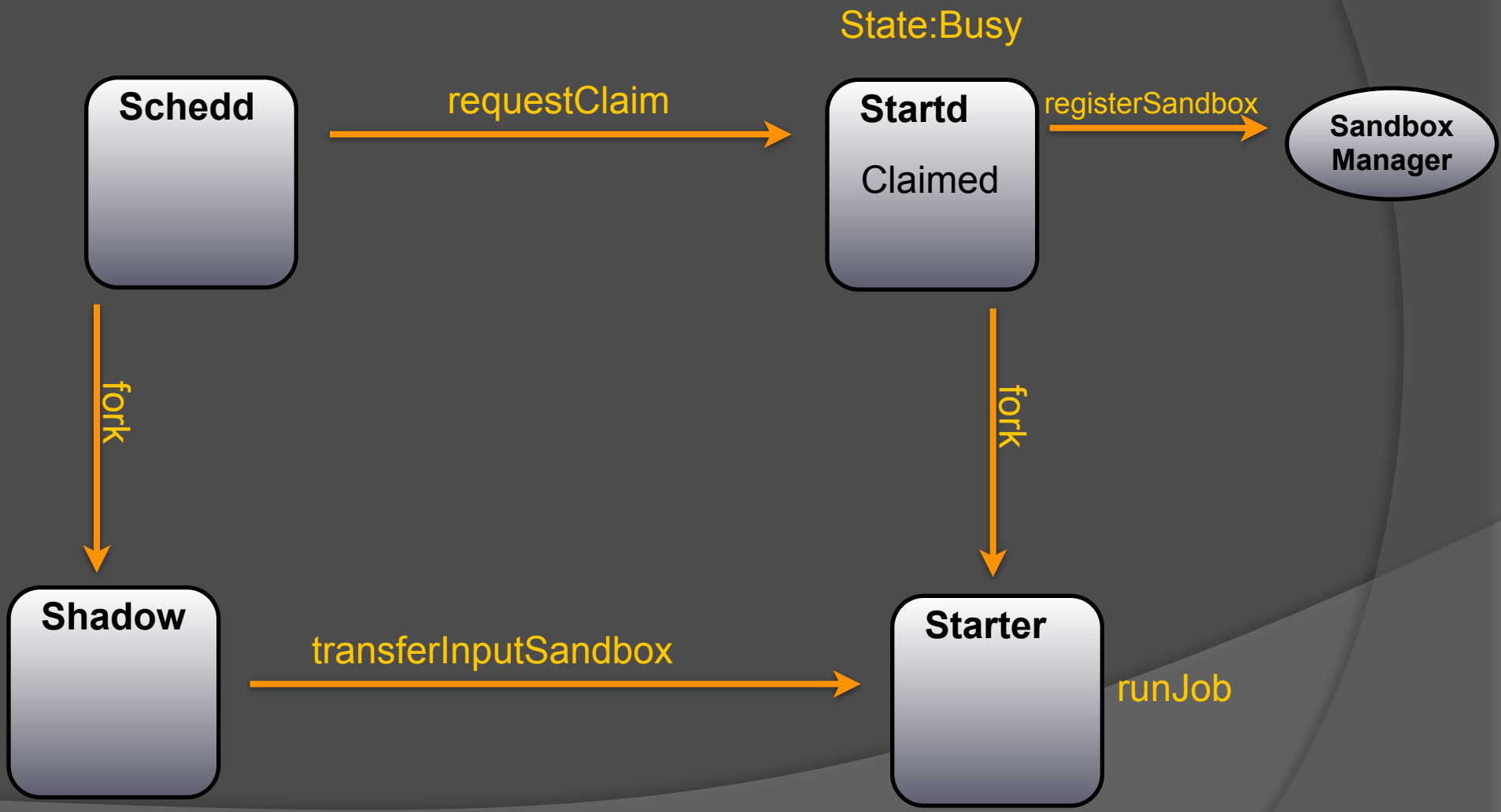
Asynchronous sandbox transfer



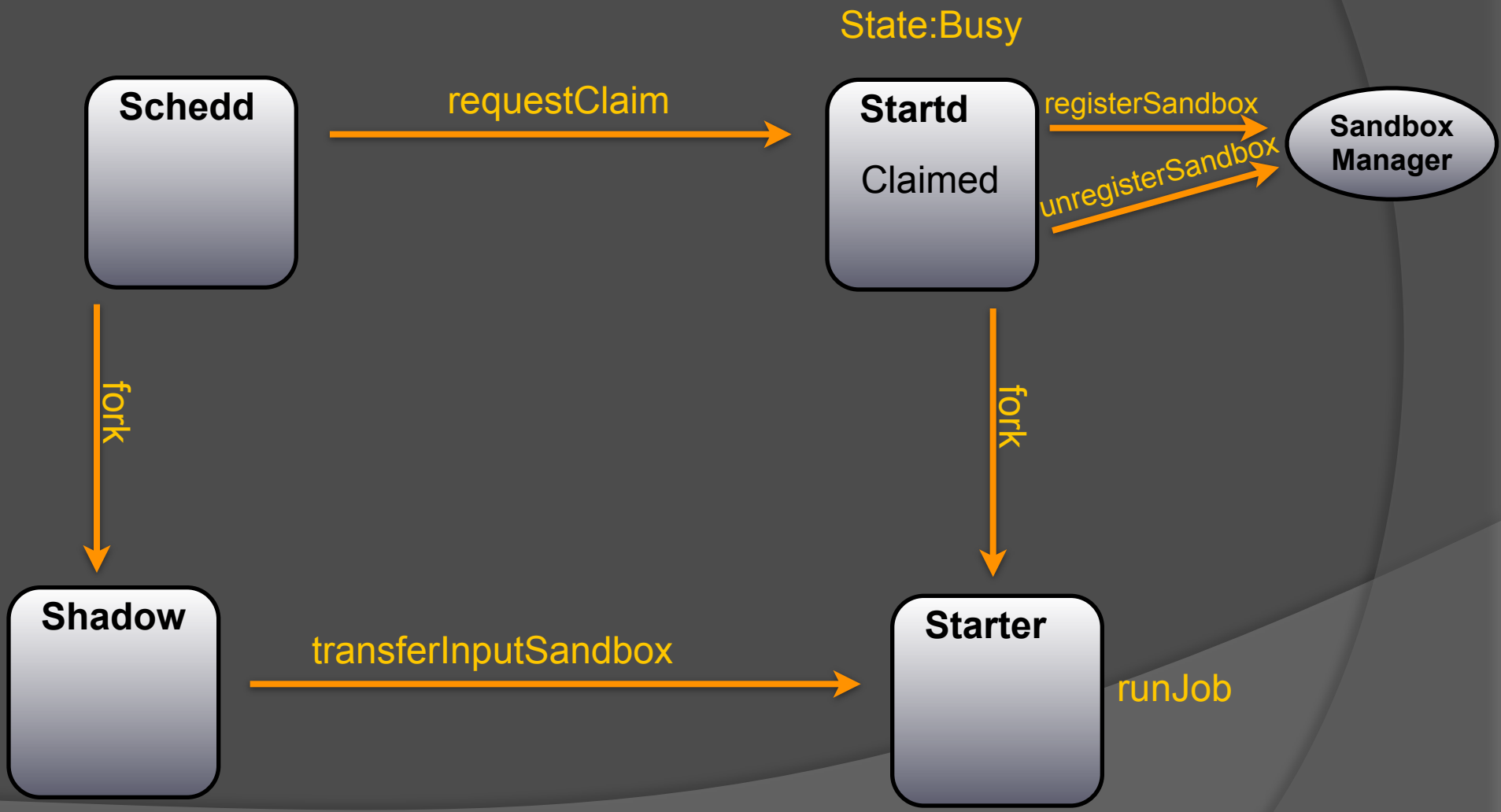
Asynchronous sandbox transfer



Asynchronous sandbox transfer



Asynchronous sandbox transfer



Asynchronous sandbox transfer

Changes in Condor: Requirements

Asynchronous sandbox transfer

Changes in Condor: Requirements

- Job needs to change state once execution is completed ("`transferring_output`")

Asynchronous sandbox transfer

Changes in Condor: Requirements

- Job needs to change state once execution is completed ("`transferring_output`")
- *Startd* needs to be able to accept a new job while previous job(s) are still transferring (if in same claim)

Asynchronous sandbox transfer

Changes in Condor: Requirements

- Job needs to change state once execution is completed (“`transferring_output`”)
- *Startd* needs to be able to accept a new job while previous job(s) are still transferring (if in same claim)
- *Schedd* needs to be able to forward next job in claim while previous job is in transfer mode

Asynchronous sandbox transfer

Changes in Condor: Requirements (2)

Asynchronous sandbox transfer

Changes in Condor: Requirements (2)

- Failure tolerance: System needs to deal with failure on submit and execute side

Asynchronous sandbox transfer

Changes in Condor: Requirements (2)

- Failure tolerance: System needs to deal with failure on submit and execute side
- File transfer failures need to be detected and accordingly handled: Do we rerun the whole job? How often do we retry? How long do we keep the output data around?

Asynchronous sandbox transfer

Changes in Condor: Requirements (2)

- Failure tolerance: System needs to deal with failure on submit and execute side
- File transfer failures need to be detected and accordingly handled: Do we rerun the whole job? How often do we retry? How long do we keep the output data around?
- Another open question: How many file transfers should run concurrently?

Asynchronous sandbox transfer

Changes in Condor: where are we at?

Asynchronous sandbox transfer

Changes in Condor: where are we at?

- Job state is changed to `transferring_output` once execution completed

Asynchronous sandbox transfer

Changes in Condor: where are we at?

- Job state is changed to `transferring_output` once execution completed
- *Startd* changes state to `claimed/idle` once job execution completed and is therefore able to accept next job in claim

Asynchronous sandbox transfer

Changes in Condor: where are we at?

- Job state is changed to `transferring_output` once execution completed
- *Startd* changes state to `claimed/idle` once job execution completed and is therefore able to accept next job in claim
- *Startd* maintains a sandbox manager object

Asynchronous sandbox transfer

Prototype assumptions

Asynchronous sandbox transfer

Prototype assumptions

- No failures

Asynchronous sandbox transfer

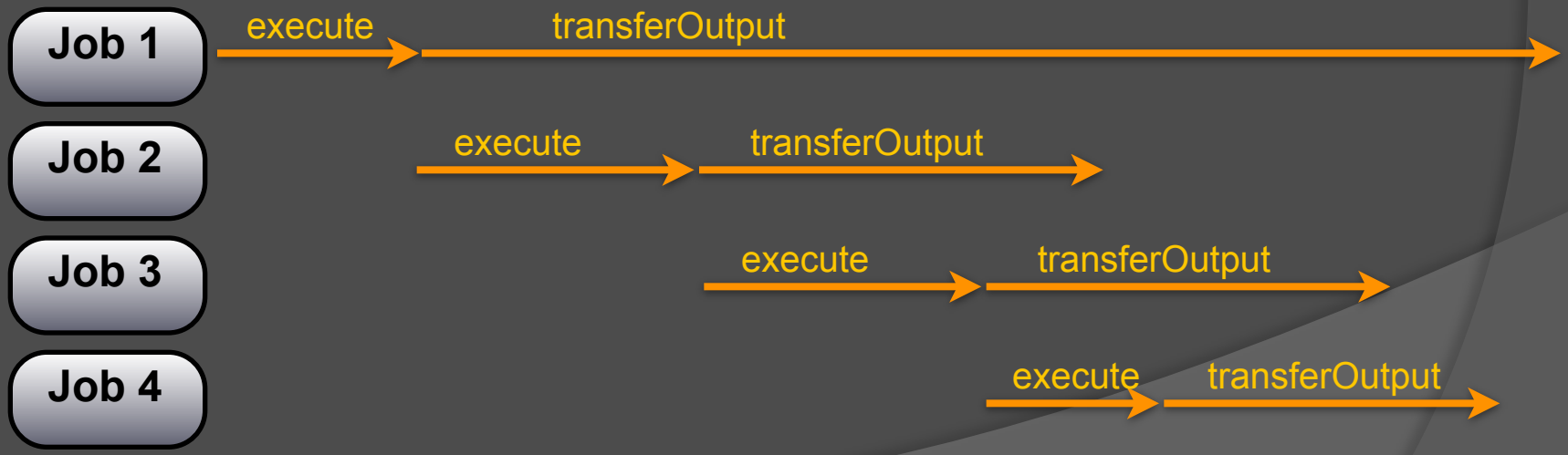
Prototype assumptions

- No failures
- First job in claim longer in transfer mode than all the following ones

Asynchronous sandbox transfer

Prototype assumptions

- No failures
- First job in claim longer in transfer mode than all the following ones



Asynchronous sandbox transfer

Prototype limitations

Asynchronous sandbox transfer

Prototype limitations

- Not production-ready yet: *Schedd* does only know about one job per claim, others run “hidden”

Asynchronous sandbox transfer

Prototype limitations

- Not production-ready yet: *Schedd* does only know about one job per claim, others run “hidden”
- Therefore all jobs in the claim are killed once the “known” job terminates (claim is deactivated)

Asynchronous sandbox transfer

Prototype limitations

- Not production-ready yet: *Schedd* does only know about one job per claim, others run “hidden”
- Therefore all jobs in the claim are killed once the “known” job terminates (claim is deactivated)
- Failure on either side leads to rerun (“hidden” jobs cannot reconnect to respective *Starters*)

Asynchronous sandbox transfer

Prototype performance results

Asynchronous sandbox transfer

Prototype performance results

- Setup: Series of *n* identical jobs, execution time *t*, output transfer time *u*, total time *v* = *t*+*u*

Asynchronous sandbox transfer

Prototype performance results

- Setup: Series of *n* identical jobs, execution time *t*, output transfer time *u*, total time *v* = *t*+*u*
- First job in series has a very long running output transfer time to cover rest of series (see limitations)

Asynchronous sandbox transfer

Prototype performance results

- Setup: Series of *n* identical jobs, execution time *t*, output transfer time *u*, total time *v* = *t*+*u*
- First job in series has a very long running output transfer time to cover rest of series (see limitations)
- No input data transfer

Asynchronous sandbox transfer

Prototype performance results

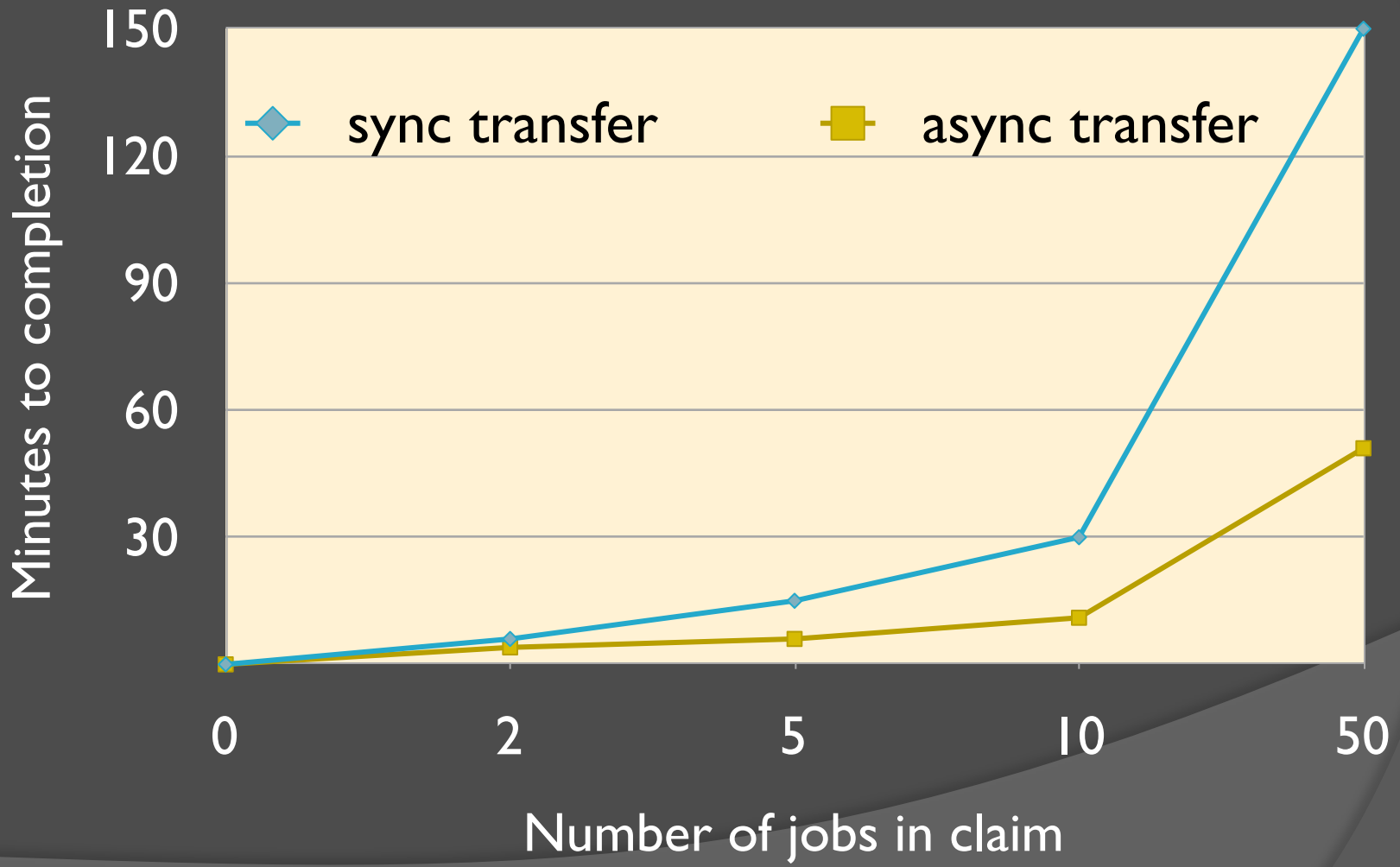
- Setup: Series of *n* identical jobs, execution time *t*, output transfer time *u*, total time *v* = *t*+*u*
- First job in series has a very long running output transfer time to cover rest of series (see limitations)
- No input data transfer
- Short running job, producing lots of output data (*u* = *2t*)

Asynchronous sandbox transfer

Prototype performance results

Asynchronous sandbox transfer

Prototype performance results



Asynchronous sandbox transfer

What needs to be done

- Needed: Concept of multiple activations per claim
- *Schedd* must maintain a list of current activations per claim (currently there is only one activation possible)
- *Startd* needs to be able to map a particular activation to a particular *Starter*

Asynchronous sandbox transfer

When will we be able to use it?

Asynchronous sandbox transfer

When will we be able to use it?

- Soon!

Asynchronous sandbox transfer

When will we be able to use it?

- Soon!
- Feature for 7.7.x development series!

Asynchronous sandbox transfer

What else?

Asynchronous sandbox transfer

What else?

- As usual: We like to make our users happy.

Asynchronous sandbox transfer

What else?

- As usual: We like to make our users happy.
- Tell us what you like about it and how you would like to use it.

Acknowledgments

- The work is being done as a part of the CEDPS project
- This work is an ongoing collaboration between the Computing Division, Fermilab and the Condor team at University of Wisconsin-Madison
- The GlideinWMS project was developed and is supported by Fermilab Computing Division and the CMS Experiment
- The Condor project is developed and supported by Condor Team at University of Wisconsin at Madison
- Fermilab is operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.
- The Open Science Grid (OSG) collaborates with CEDPS for solutions on data movement and stage-out