glideinWMS Training

HTCondor Overview

by Igor Sfiligoi, UC San Diego

Overview

- These slides present a HTCondor overview, with high level views of
  - Deamons involved
  - Communication paths
  - Scalability considerations

HTCondor Daemons

The basics

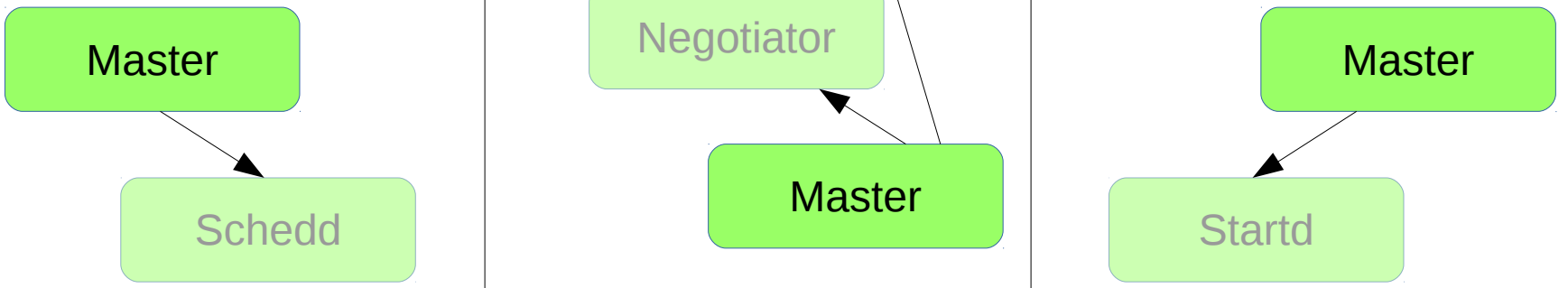
Submit side

Glue

Execute side

HTCondor Daemons

The basics + the master



Submit side

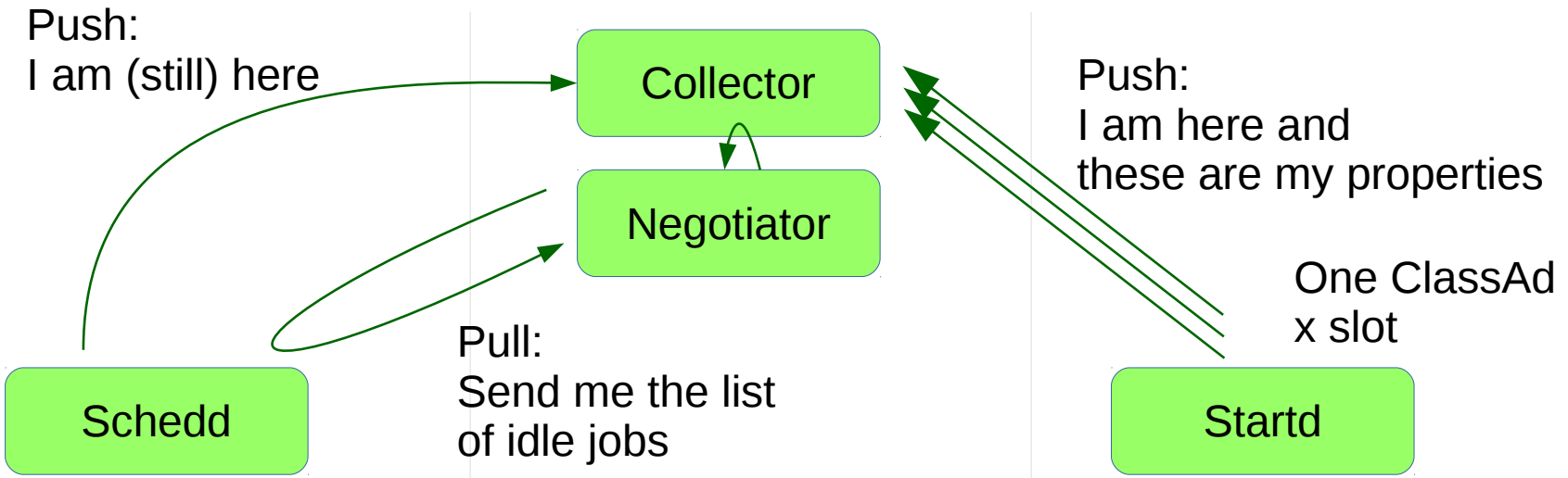
Glue

Execute side

HTCondor Daemons

- One startd per (logical) compute resource
  - Can handle multiple CPUs
- One schedd per submit node
  - Can handle multiple users
- Collector has the list of all other daemons
- Negotiator matches user jobs to machine slots
- Master starts all other processes
  - Will ignore it in the rest of the talk

Communication flow



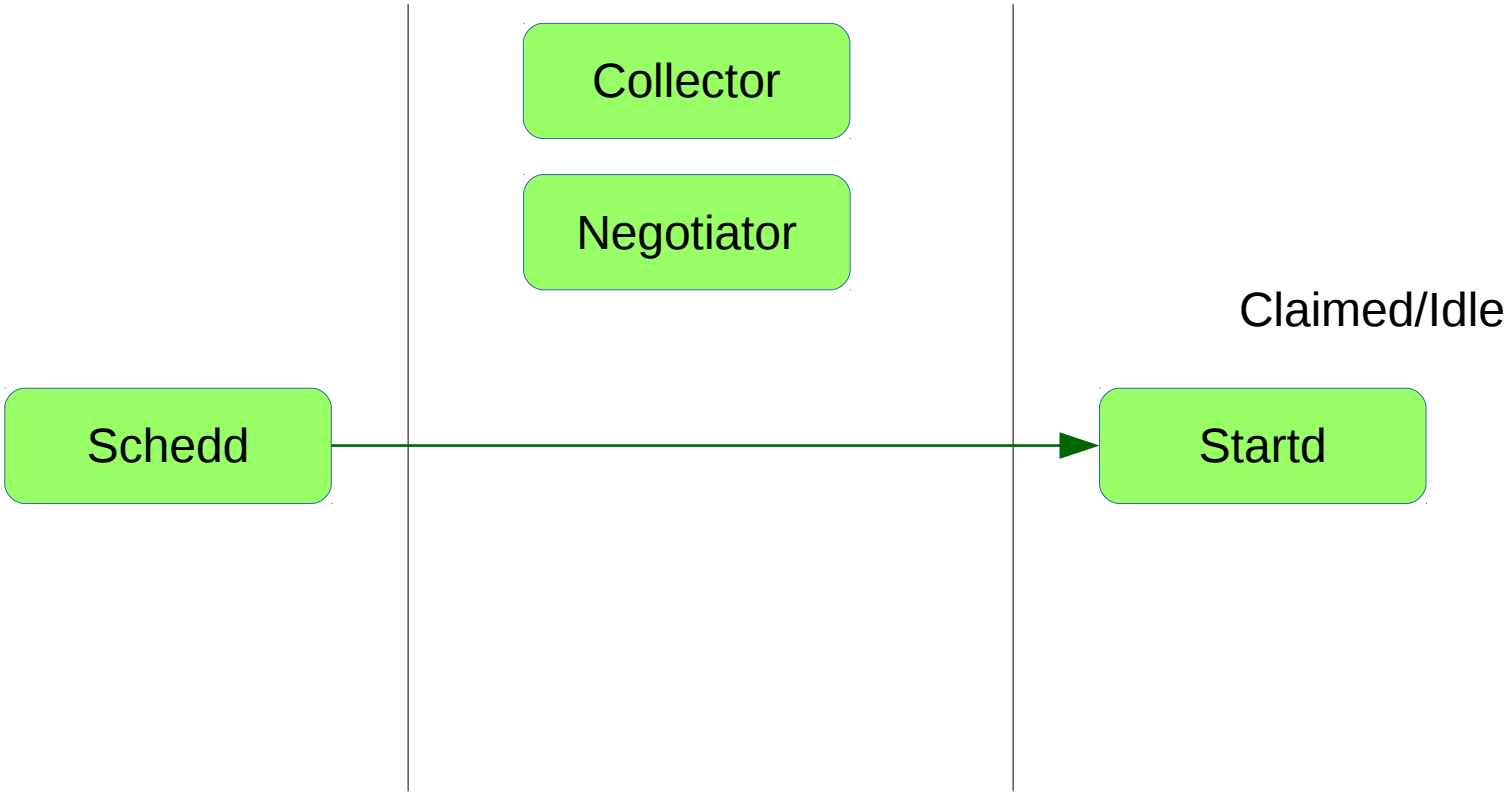
Claiming protocol

- Startds keep their state current in the Collector
  - By periodically pushing updates (every 5 mins by default)
- On a matchmaking cycle, the Negotiator will
  - Pull the startd slot list from the collector
    - In Unclaimed state only (unless preemption enabled)
  - Pull the job list from the schedds
  - Create a priority list of matches
  - Send the matches to relevant schedds

Claiming protocol

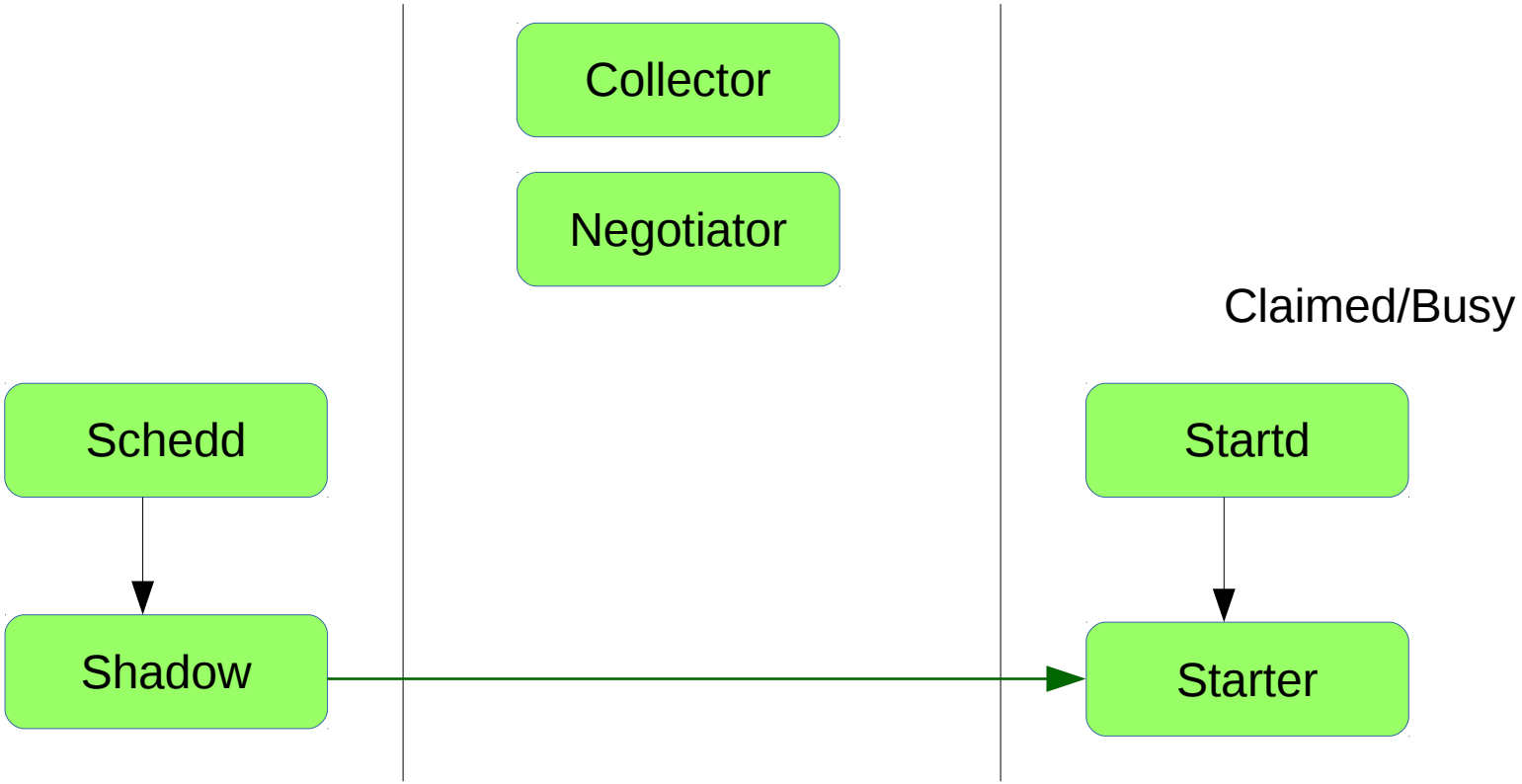
- The schedd will contact the startd
  - Once the connection is accepted, the schedd owns that slots
- The schedd will spawn a shadow
  - Which takes over the connection
  - The schedd moves on to other business
- Similarly, the startd spawns a starter
  - And advertise a Claimed state

Communication flow



HTCondor Daemons

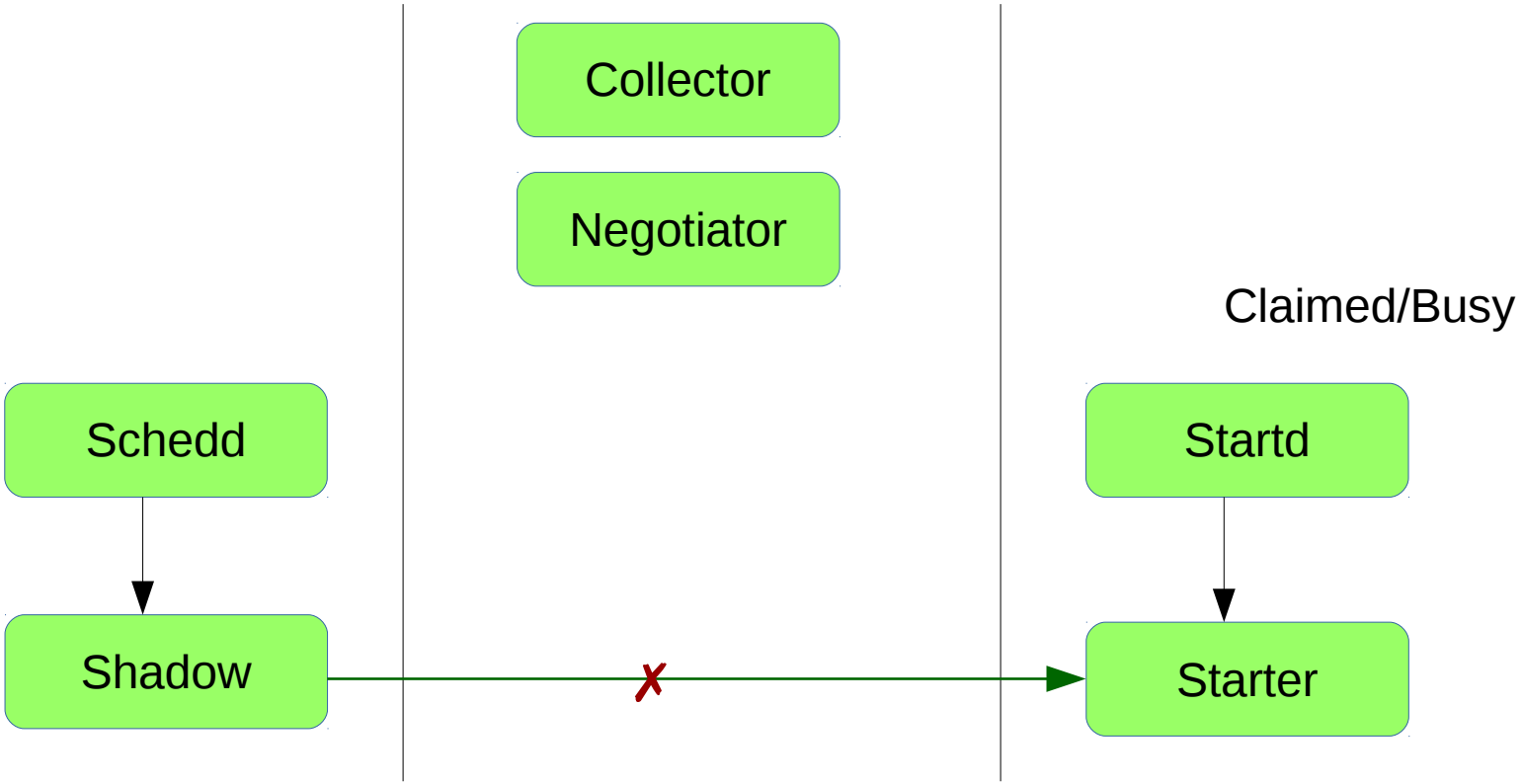
Stage 2



A shadow and a starter are created for every running job

HTCondor Daemons

Stage 2



If the network connection is lost, either side can re-establish it.

HTCondor Daemons

- The shadow takes ownership of a running job
  - One per job
- The starter takes ownership of a claimed slot
  - One per slot
- Together they babysit the two sides until the jobs is done and the slot can be un-claimed

- Corollary:
  - Each schedd node will have O(10k) shadows

Claiming protocol

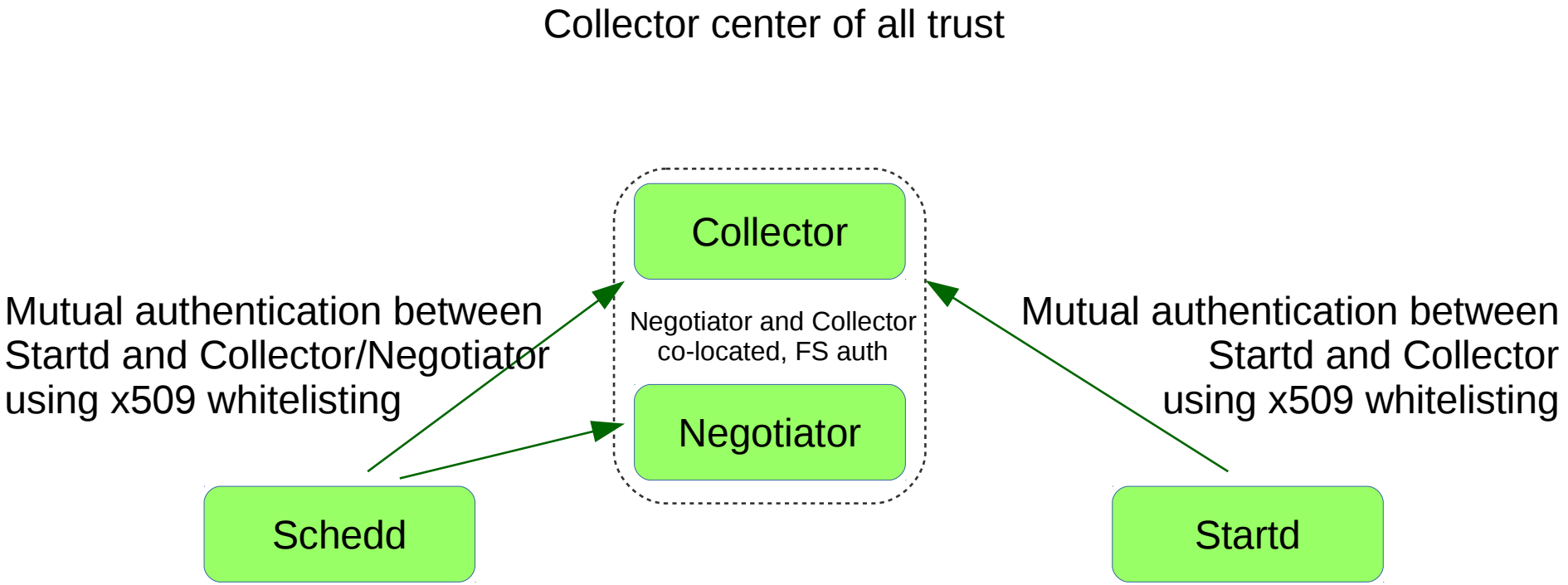
- Once the job terminates
  - The starter goes away
  - The schedd will send another job to the startd, unless
    - The lease has expired
    - There are no more suitable jobs
  - The existing shadow can be reused
    - But does not need to
- If the schedd does not send a new job
  - Startd goes into UnClaimed state

Matchmaking and latency

- The Negotiator pulls the startd slot list from the Collector
  - In a single transaction
- The Negotiator pulls the job list from the schedds
  - Basically, one at a time!
  - But it does cluster similar jobs together at Schedd level
  - The idea being that it will not ask for more, if either the user runs out of priority or no more slots are available
- Negotiator thus sensitive to Network latencies
  - Matching Schedds far away may be limited by network latency not Negotiator CPU use

Security considerations

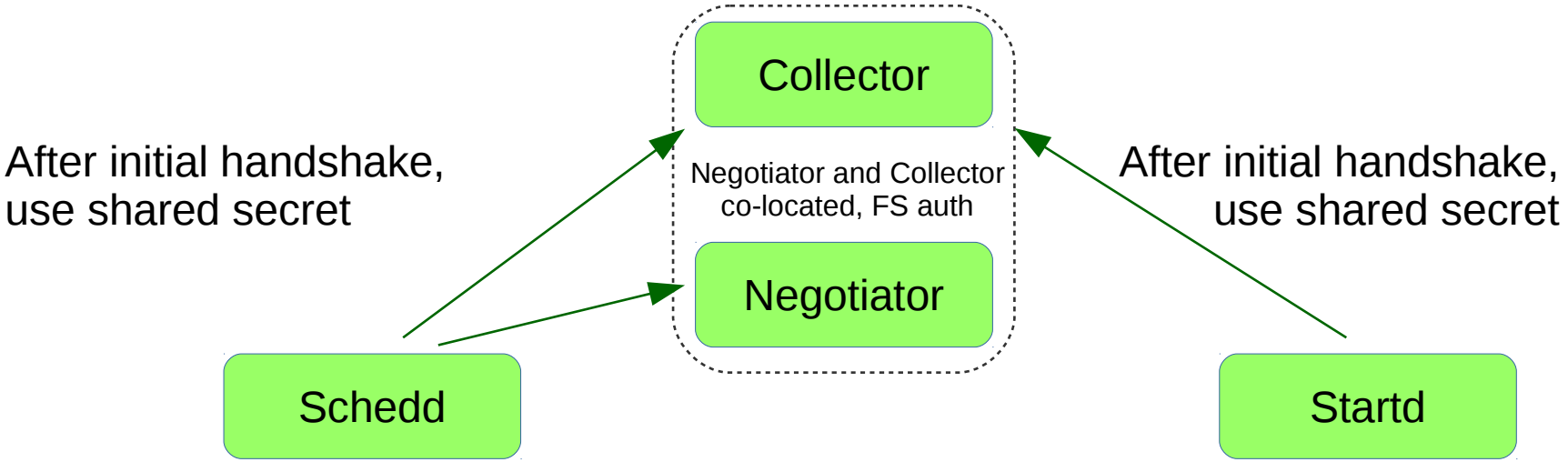
The glidein use case



Security considerations

The glidein use case

Collector center of all trust

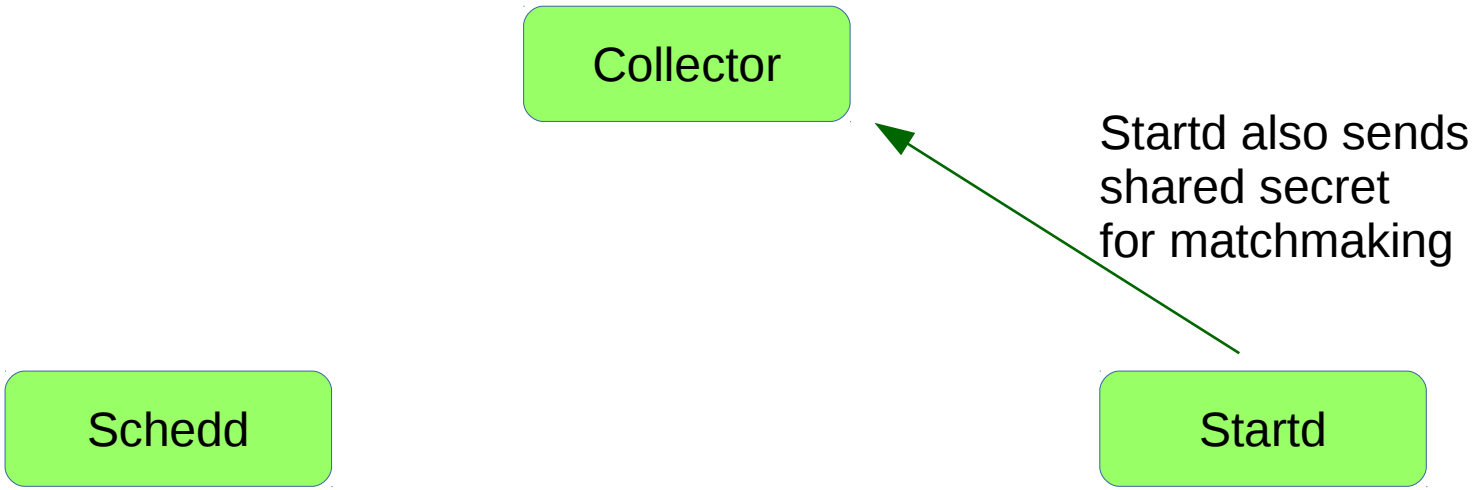


Full x509 expensive, used only on daemon restart
and periodically once every few days

Security considerations

The glidein use case

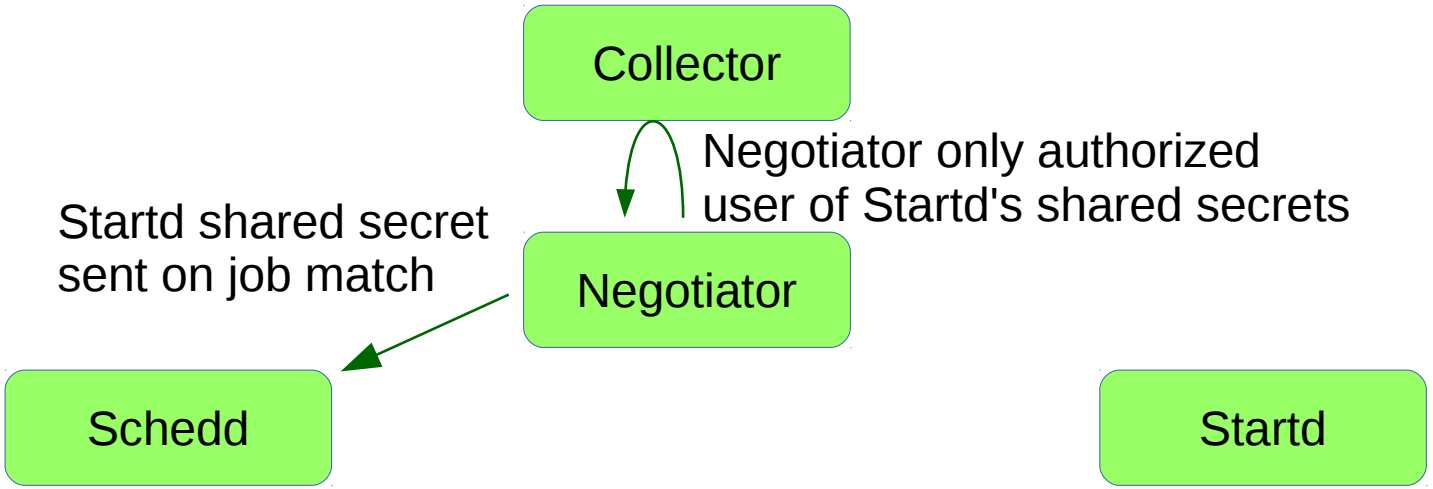
Collector center of all trust



Security considerations

The glidein use case

Collector center of all trust

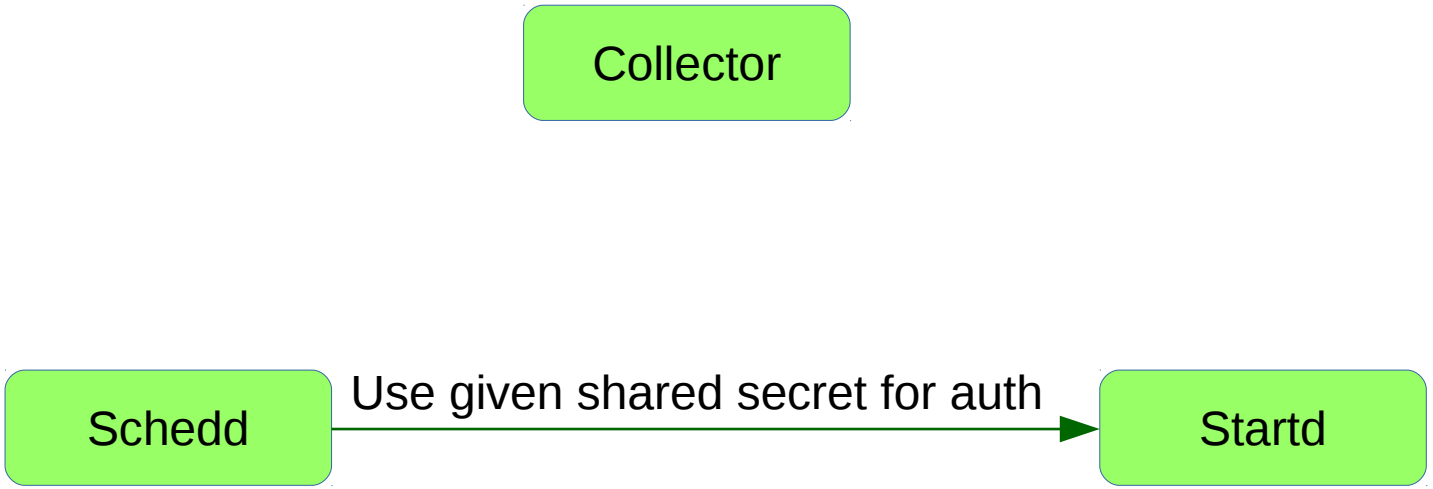


Schedd may get many secrets,
one per matched job

Security considerations

The glidein use case

Collector center of all trust

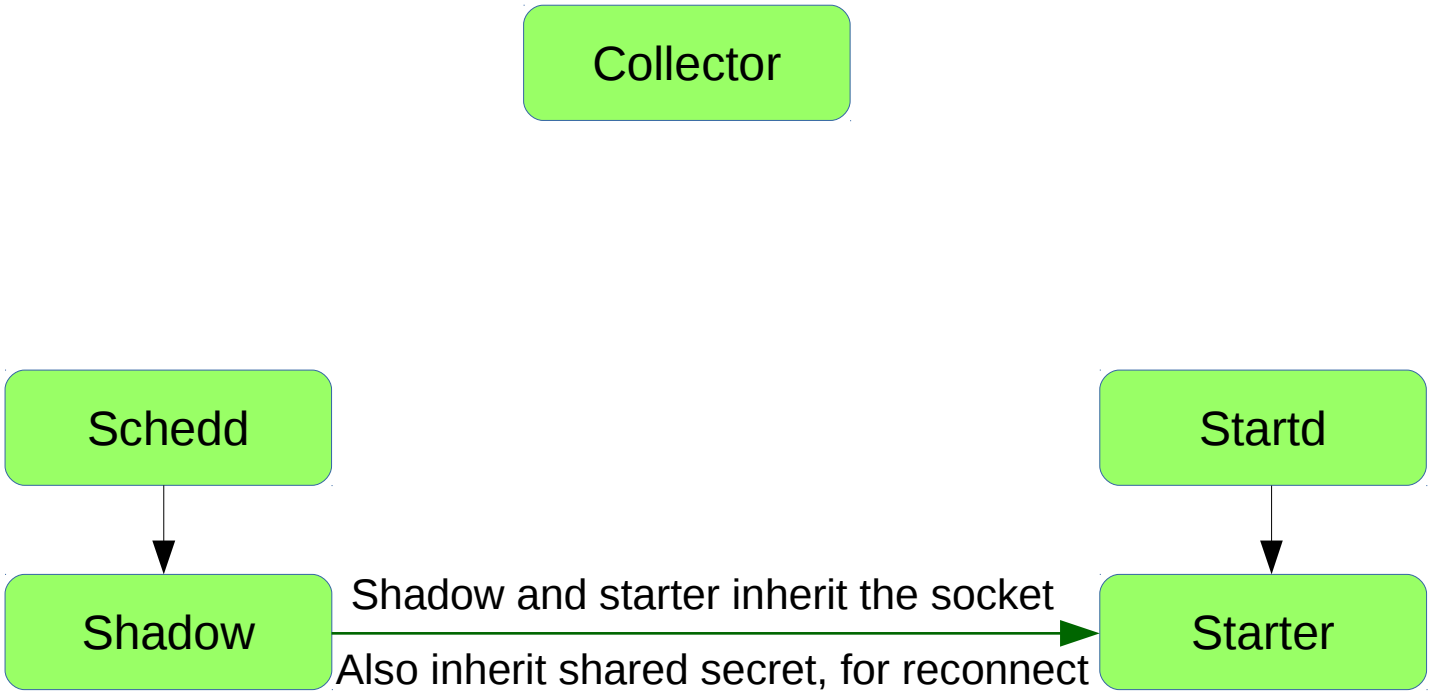


No other credentials
in play

Security considerations

The glidein use case

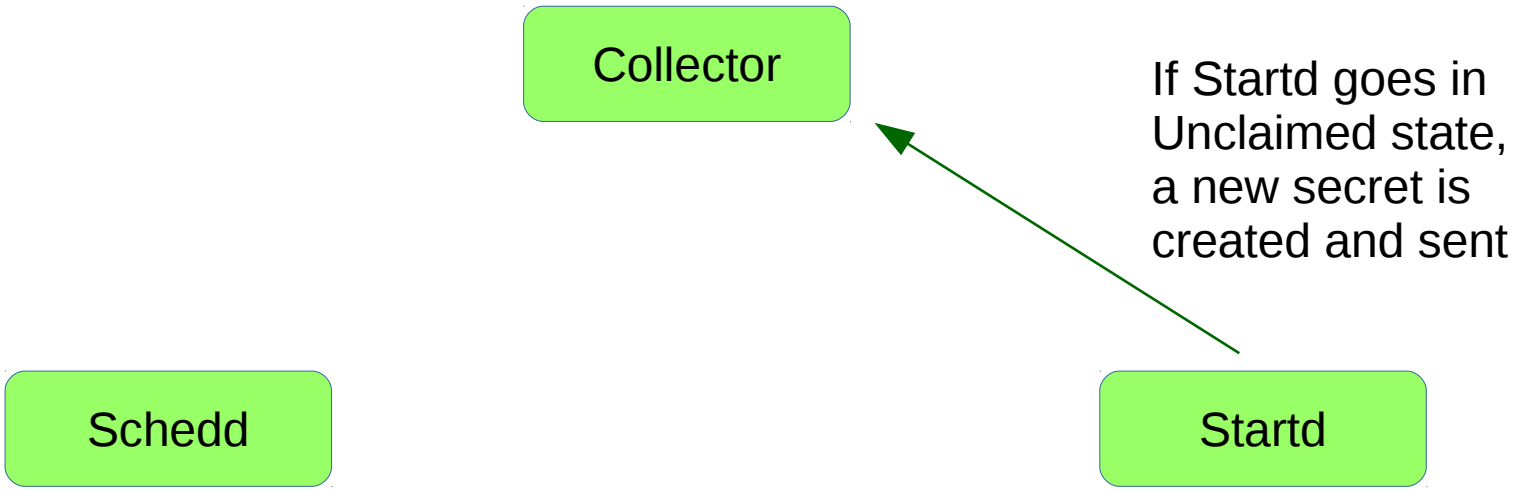
Collector center of all trust



Security considerations

The glidein use case

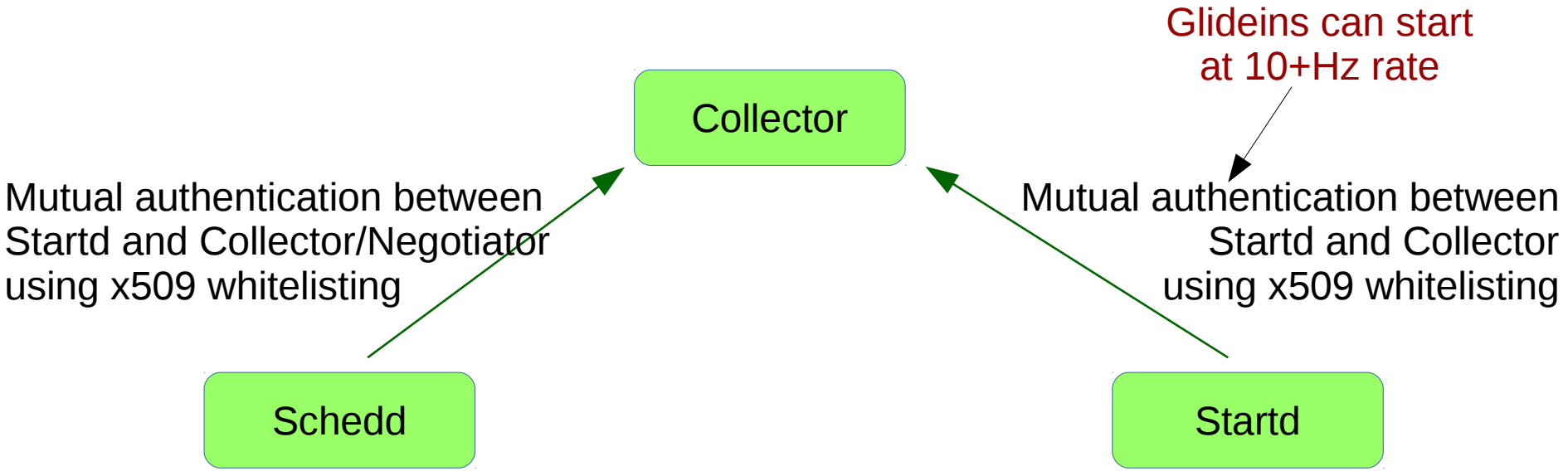
Collector center of all trust



Security cost and scalability

The glidein use case

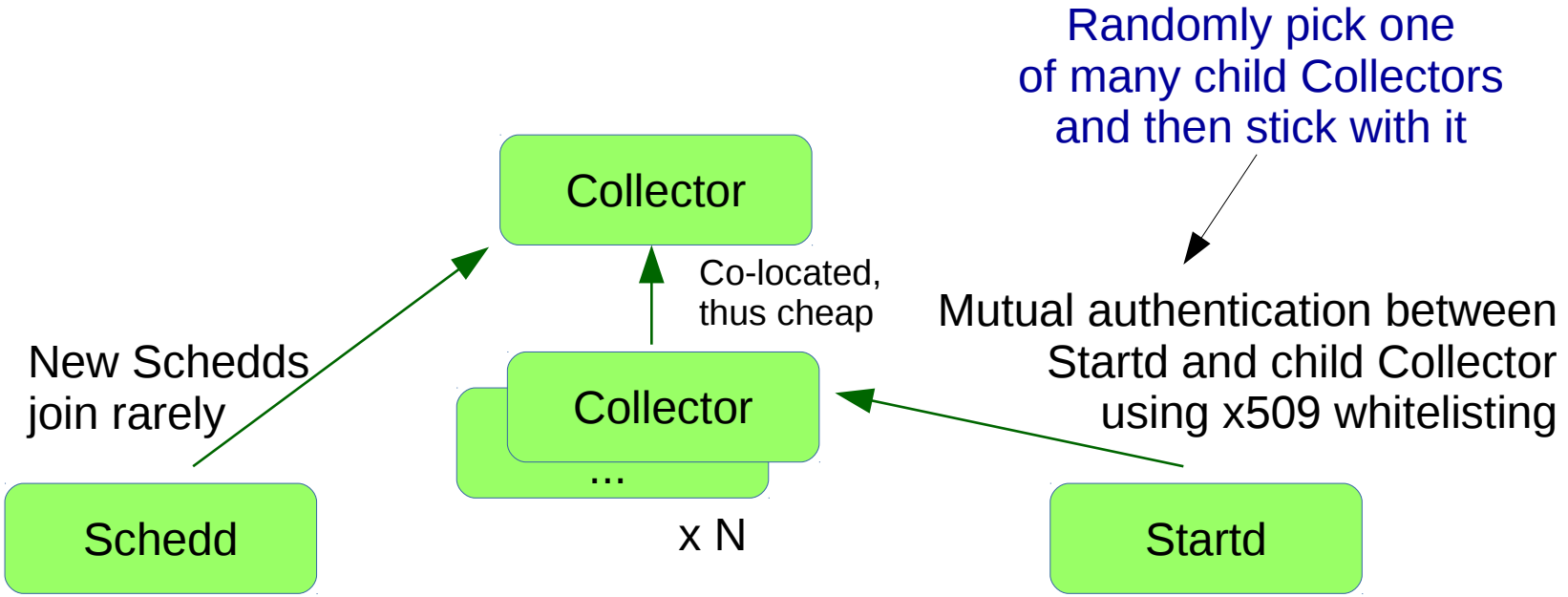
x509 too expensive for a single central service
(both due to CPU use, and network latency issues)



Security cost and scalability

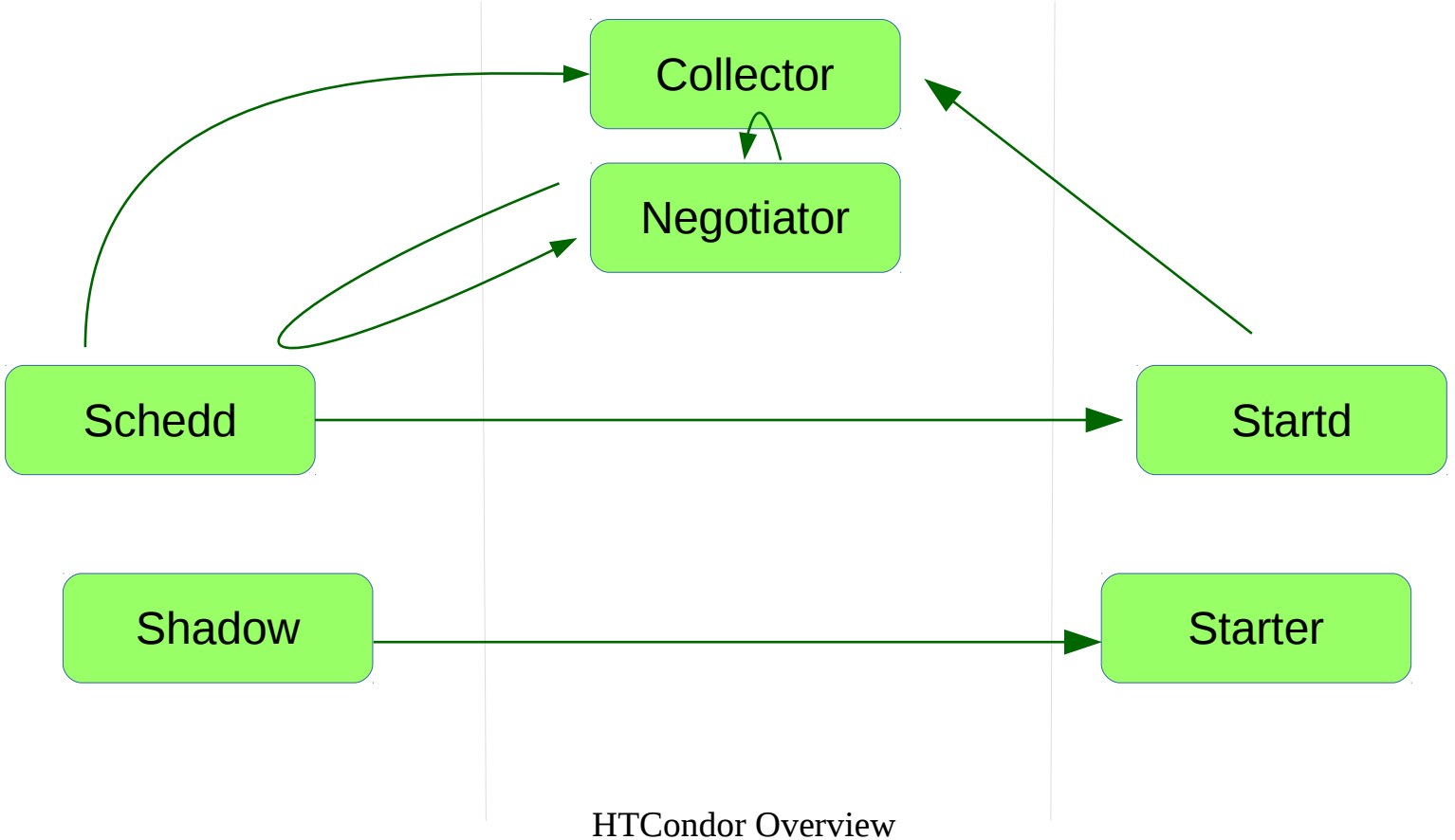
The glidein use case

Spread the load over multiple child Collectors
Child collectors forward all ads



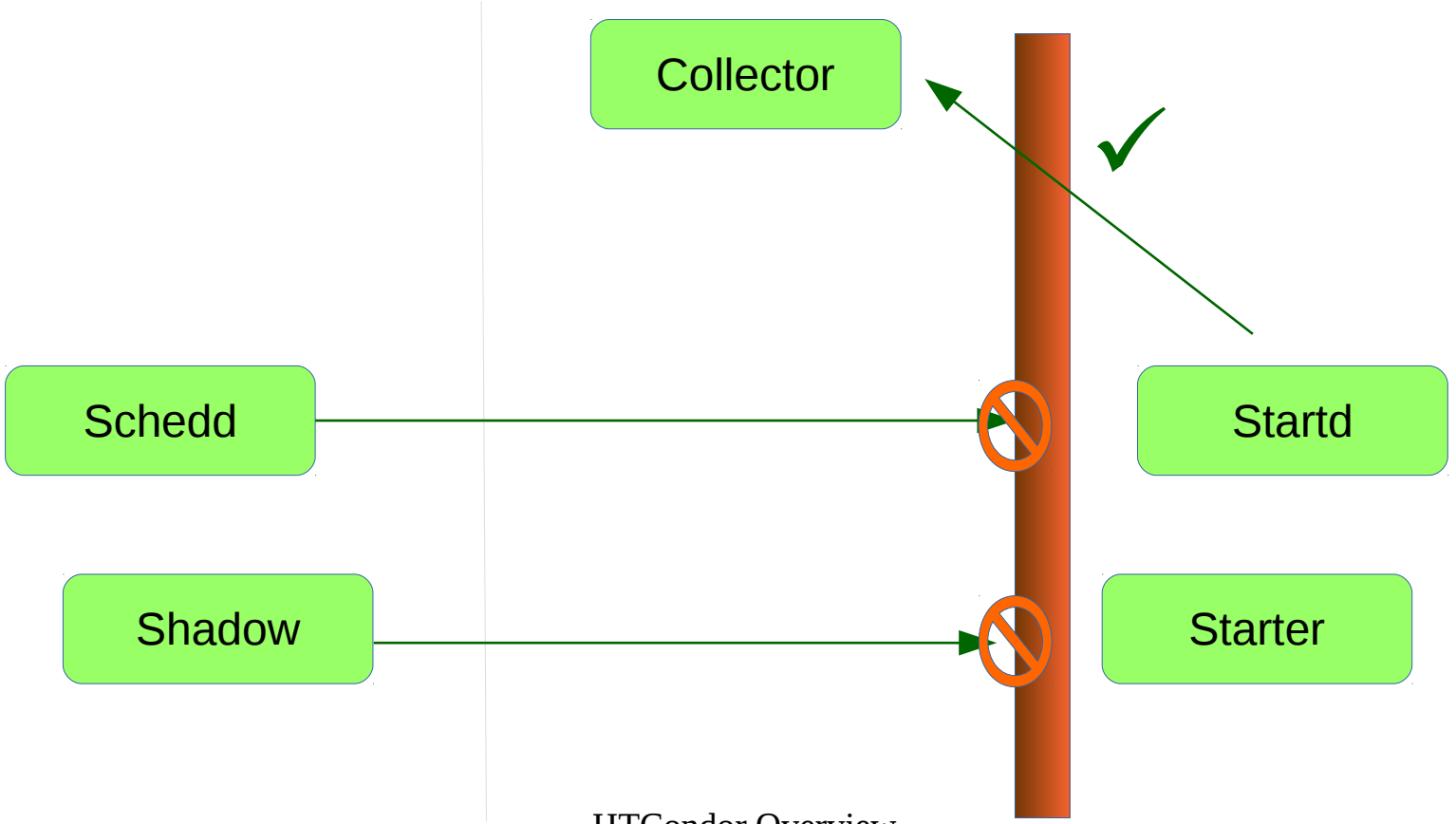
Network/Firewall considerations

HTCondor is conceptually a Peer-to-Peer system
Everyone talks to everyone



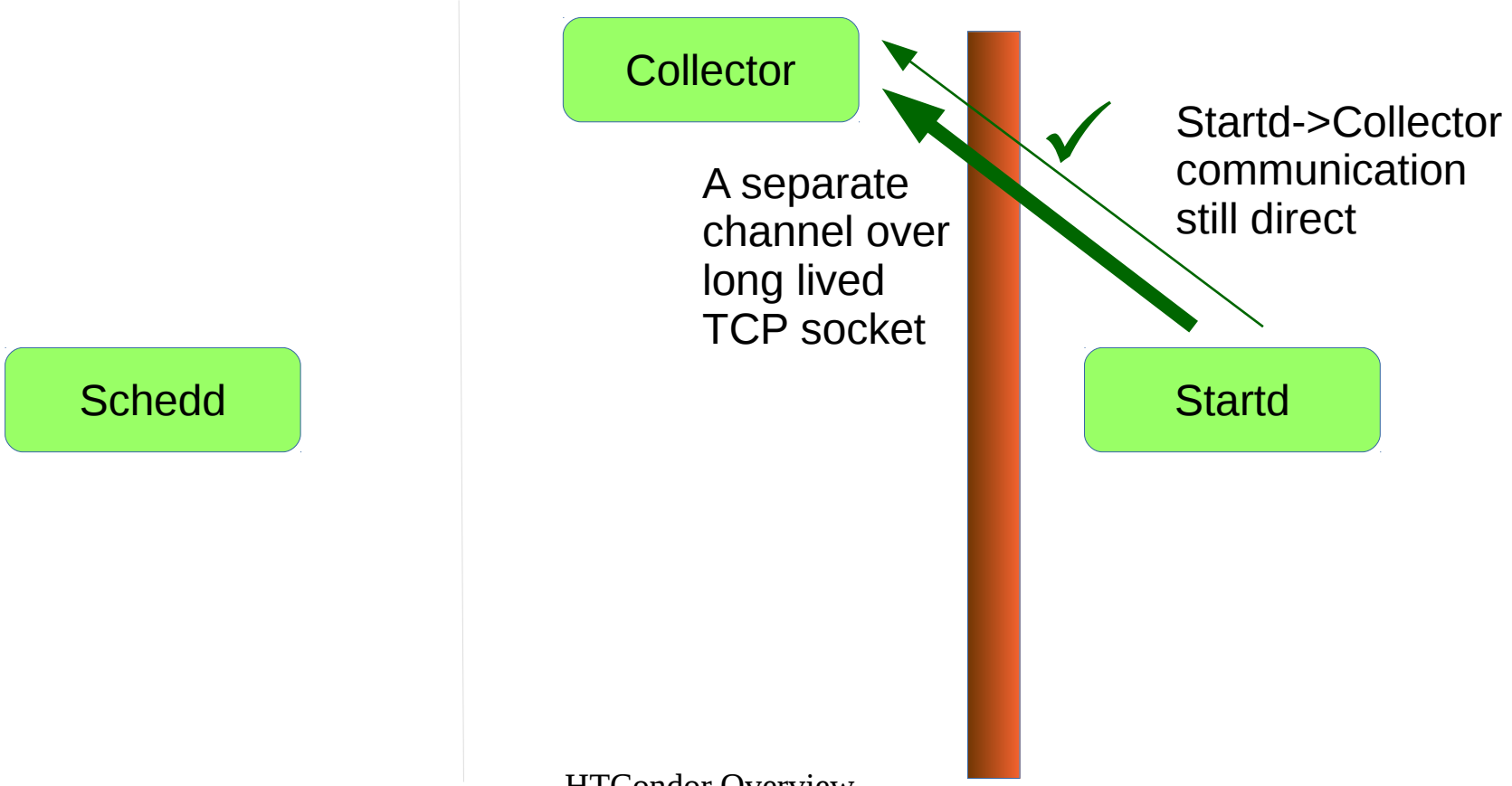
Network/Firewall considerations

Execute nodes often behind firewalls and/or NATs



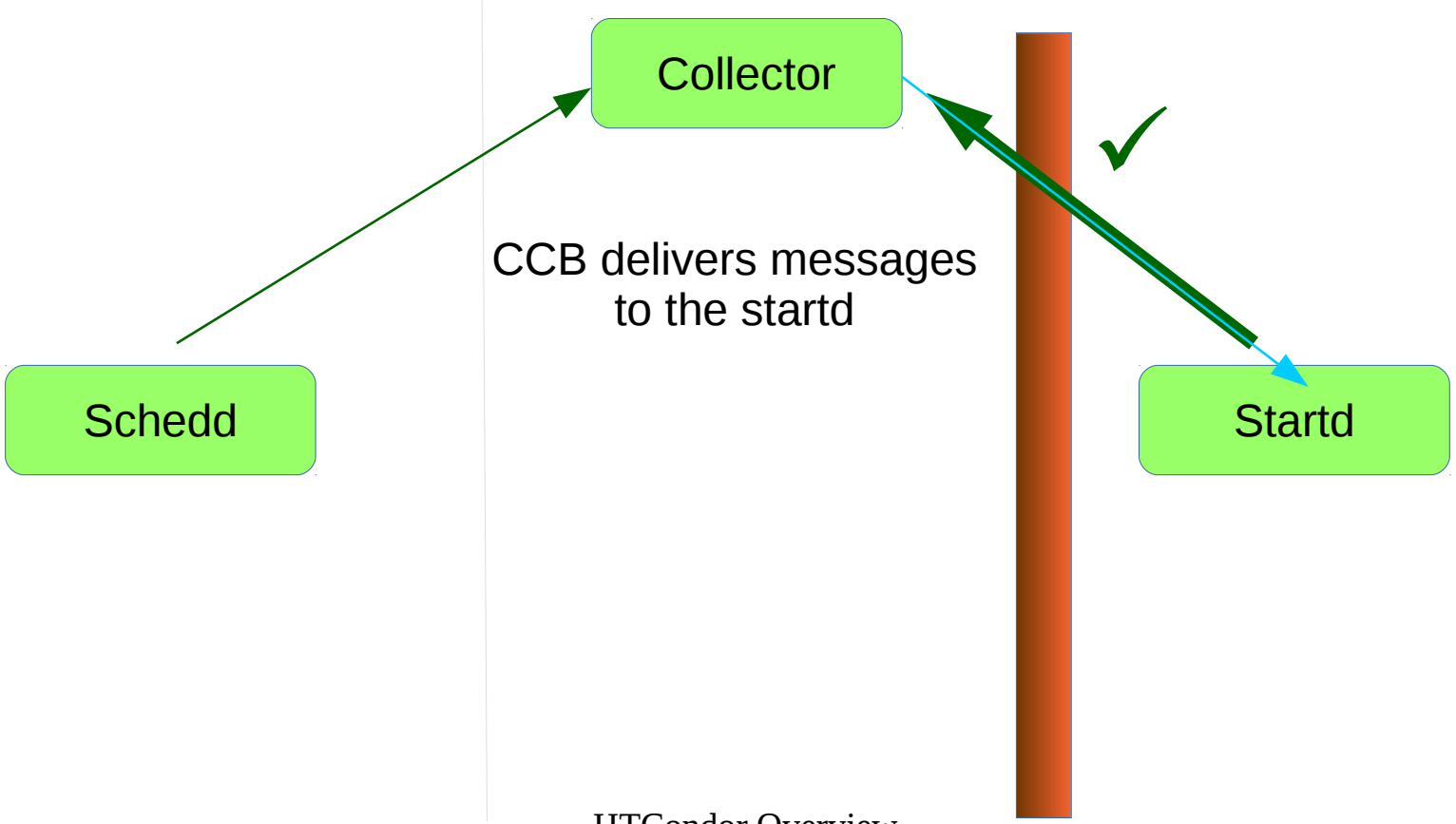
Network/Firewall considerations

CCB protocol creates a tunnel
Collector implements the CCB



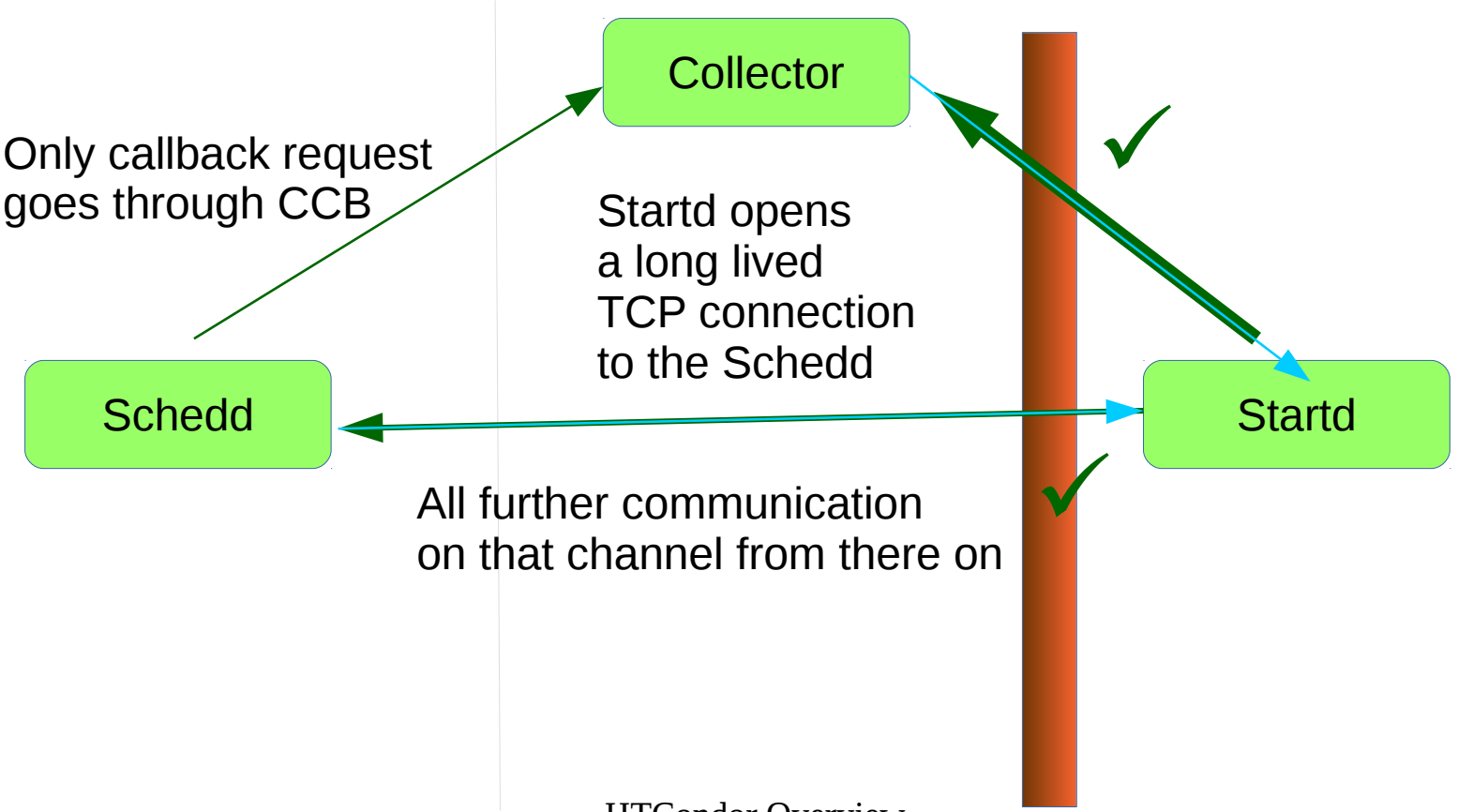
Network/Firewall considerations

CCB protocol creates a tunnel
Collector implements the CCB



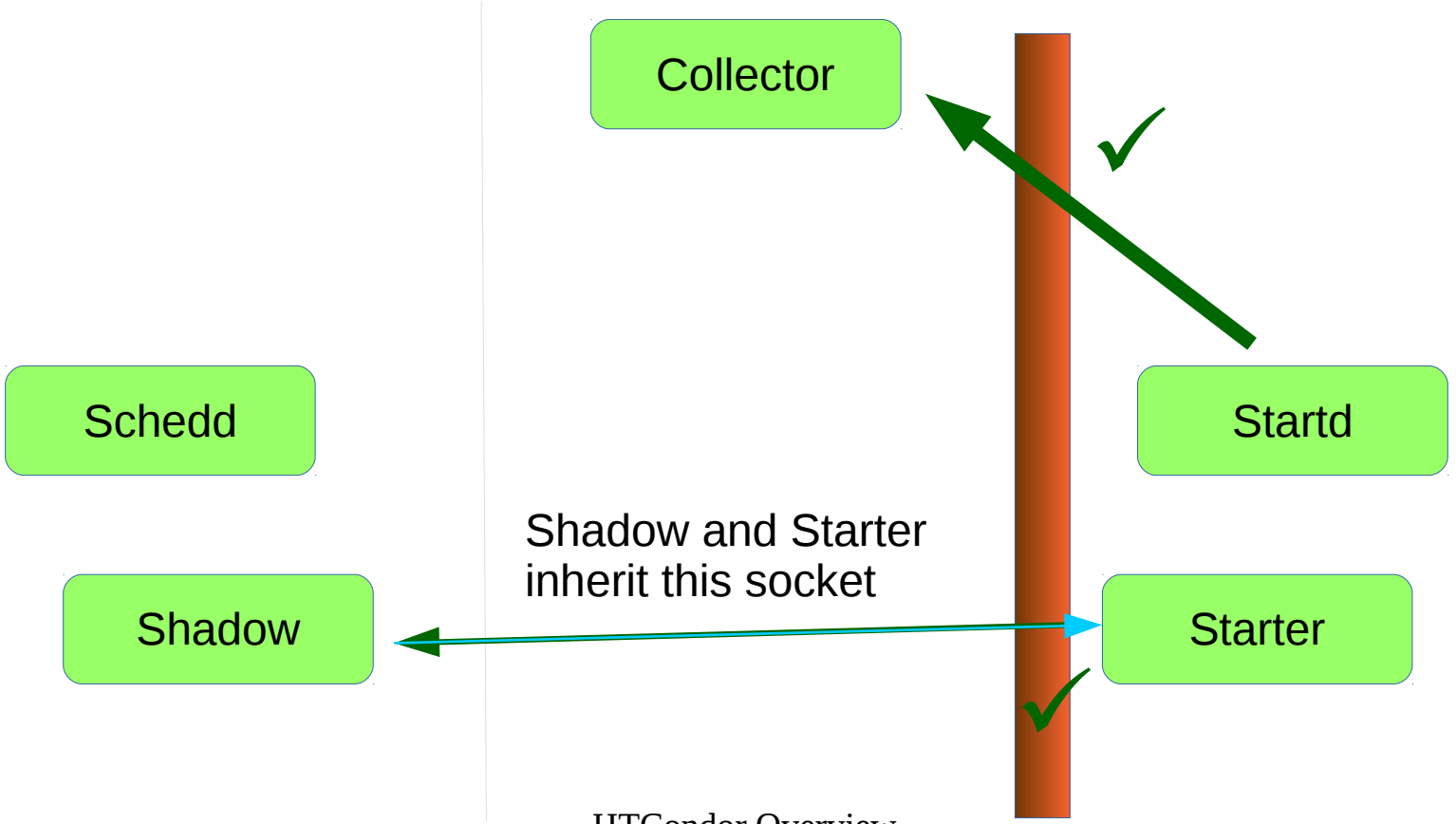
Network/Firewall considerations

CCB protocol creates a tunnel
Collector implements the CCB



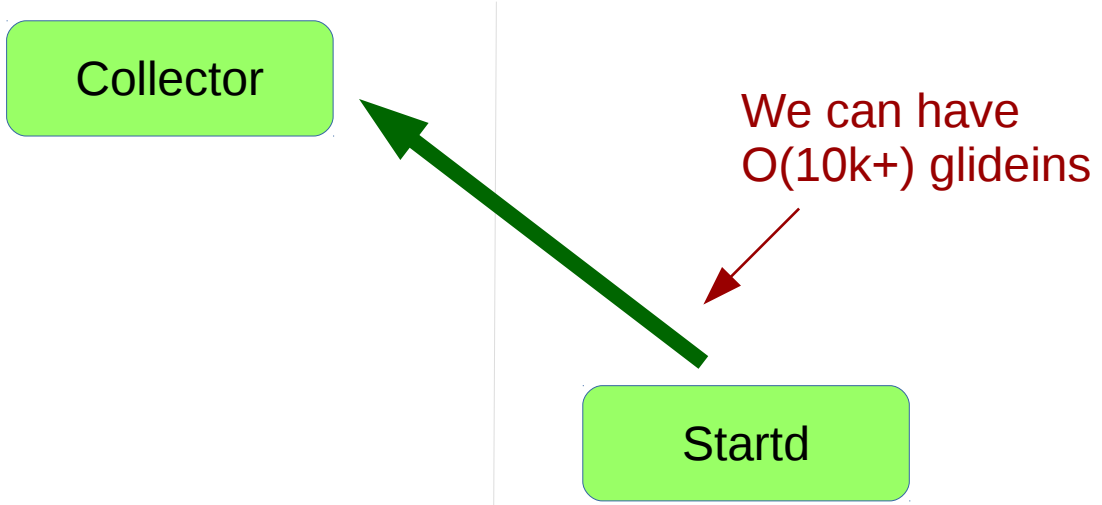
Network/Firewall considerations

CCB protocol creates a tunnel
Collector implements the CCB



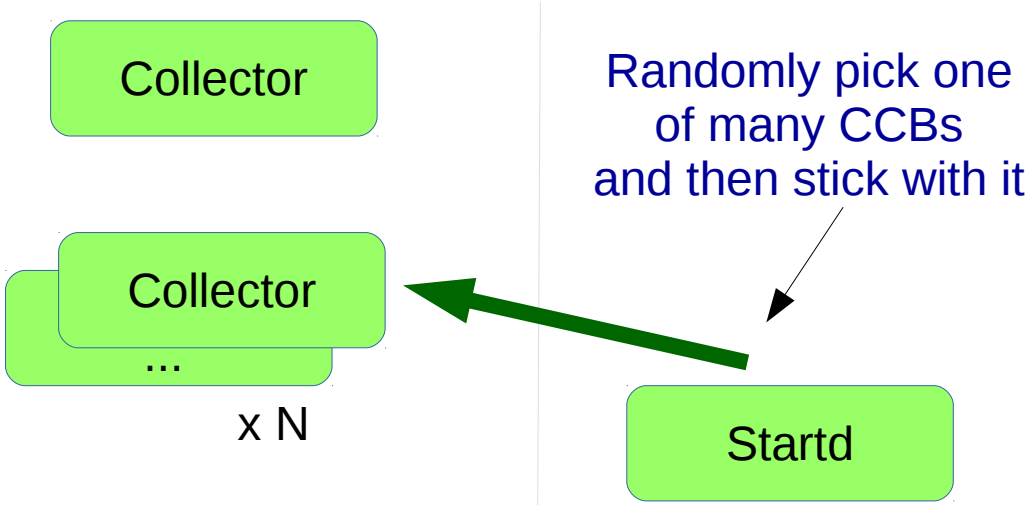
CCB and scalability

A single central service cannot really handle all the load
Processes usually limited to O(1k) sockets



CCB and scalability

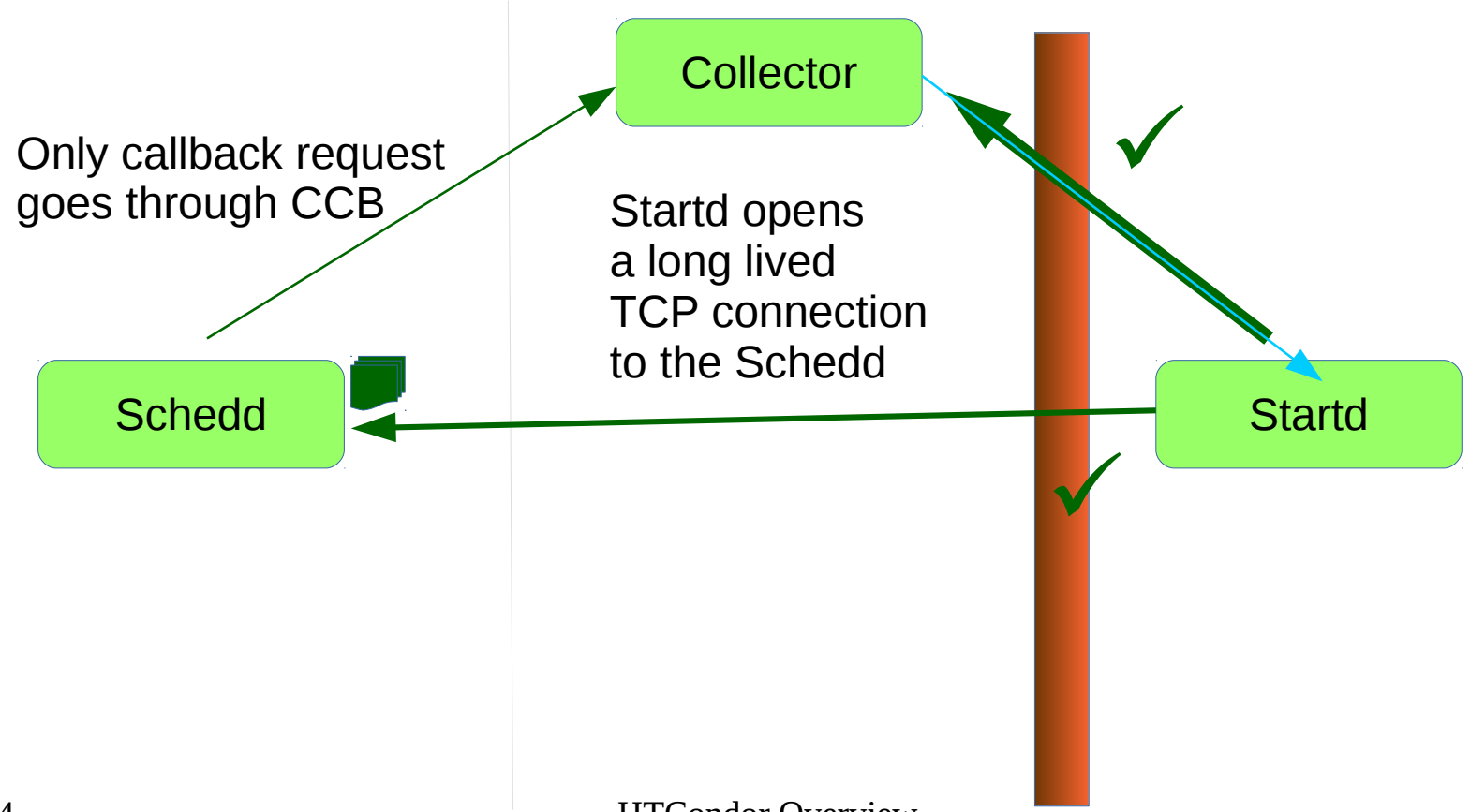
No real need to use a single CCB,
could use any number of dedicated CCB Collectors



The standard strategy is
to just piggy-back on
the “Child Collector” paradigm

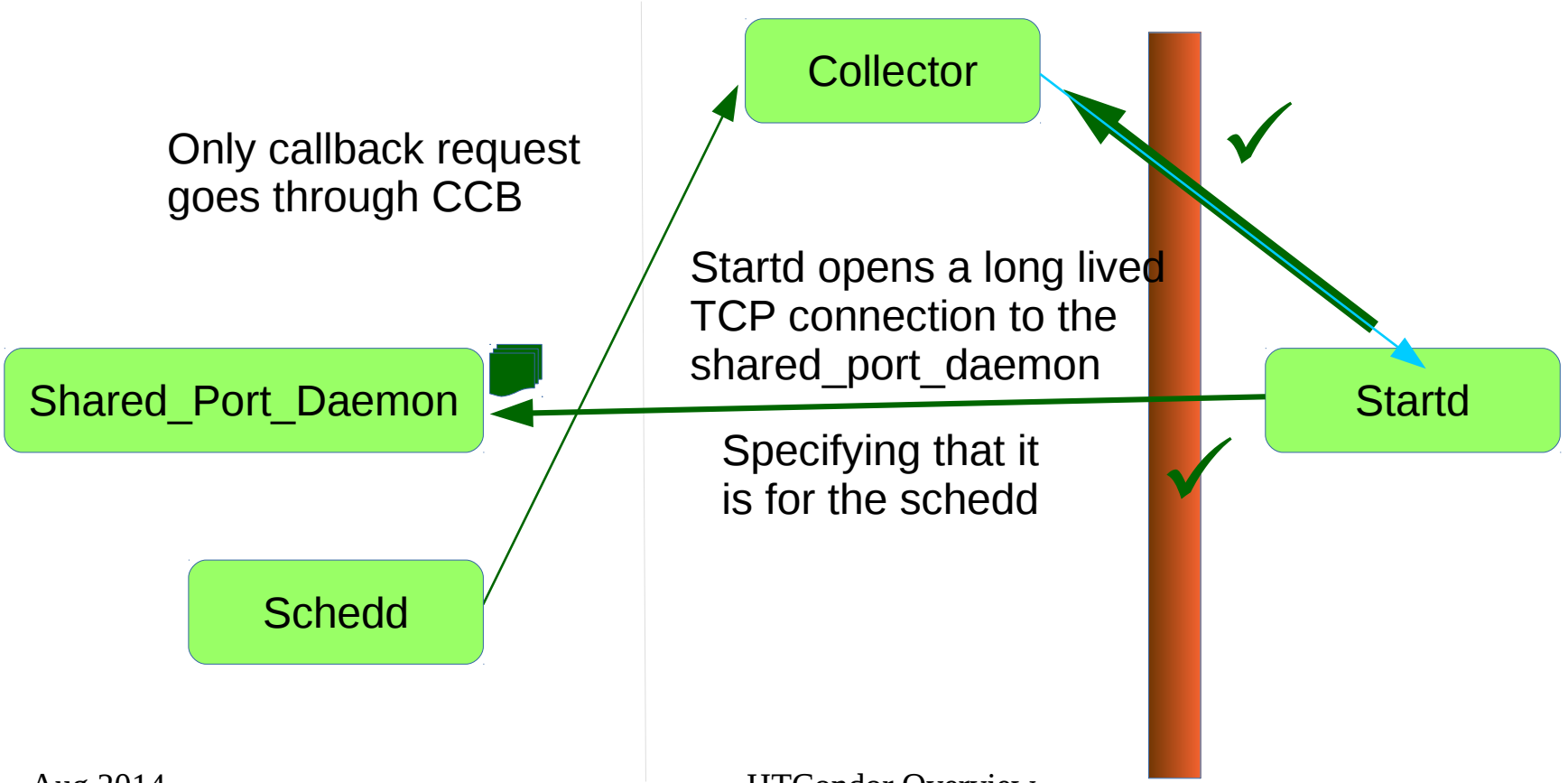
CCB and scalability

The Schedd now needs to accept incoming connections
Default HTCondor mechanism of “one port x connection” does not scale
(only ~30k usable ports in IP)



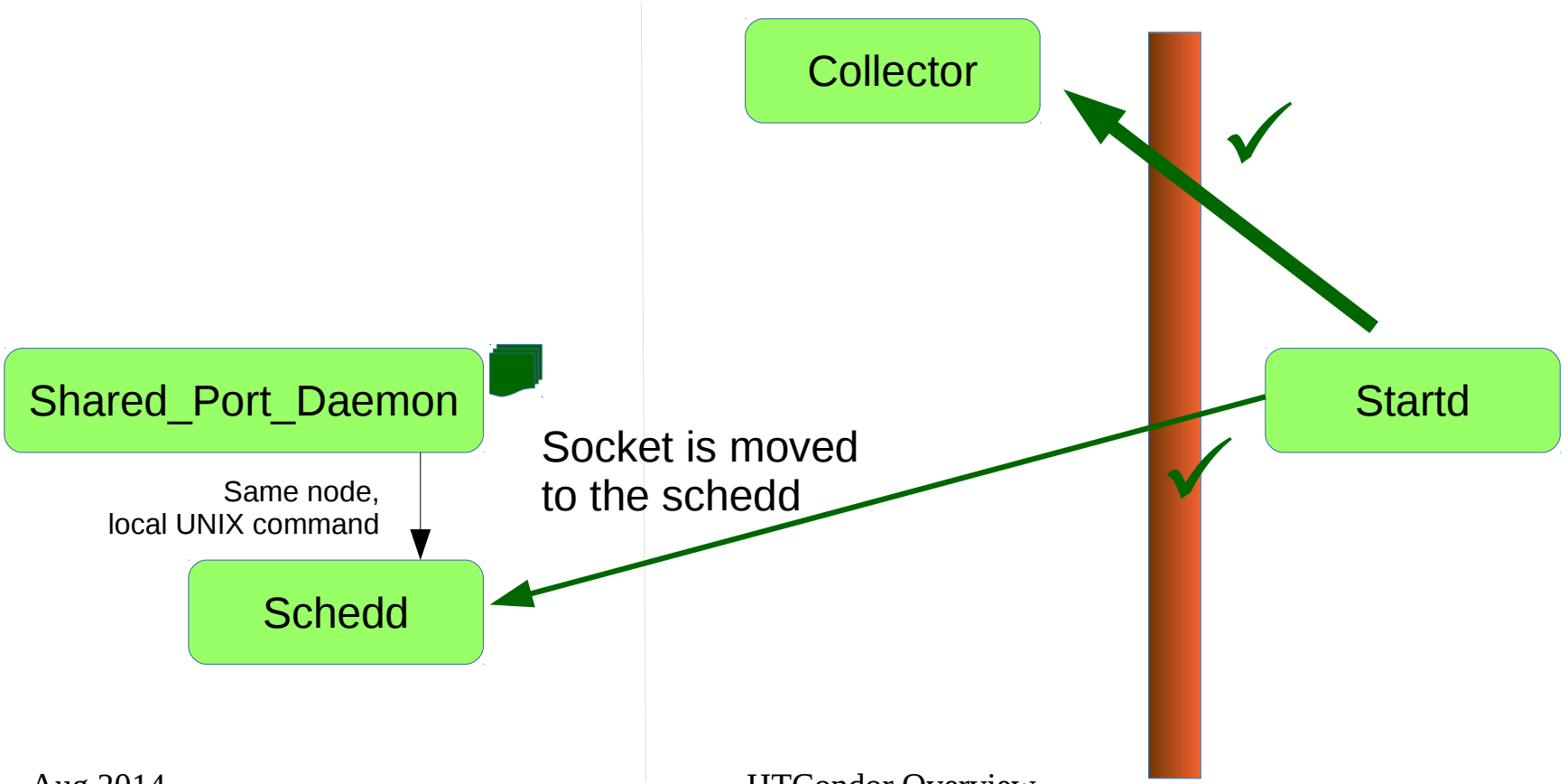
CCB and scalability

HTCondor added shared_port_daemon to multiplex requests on a single port



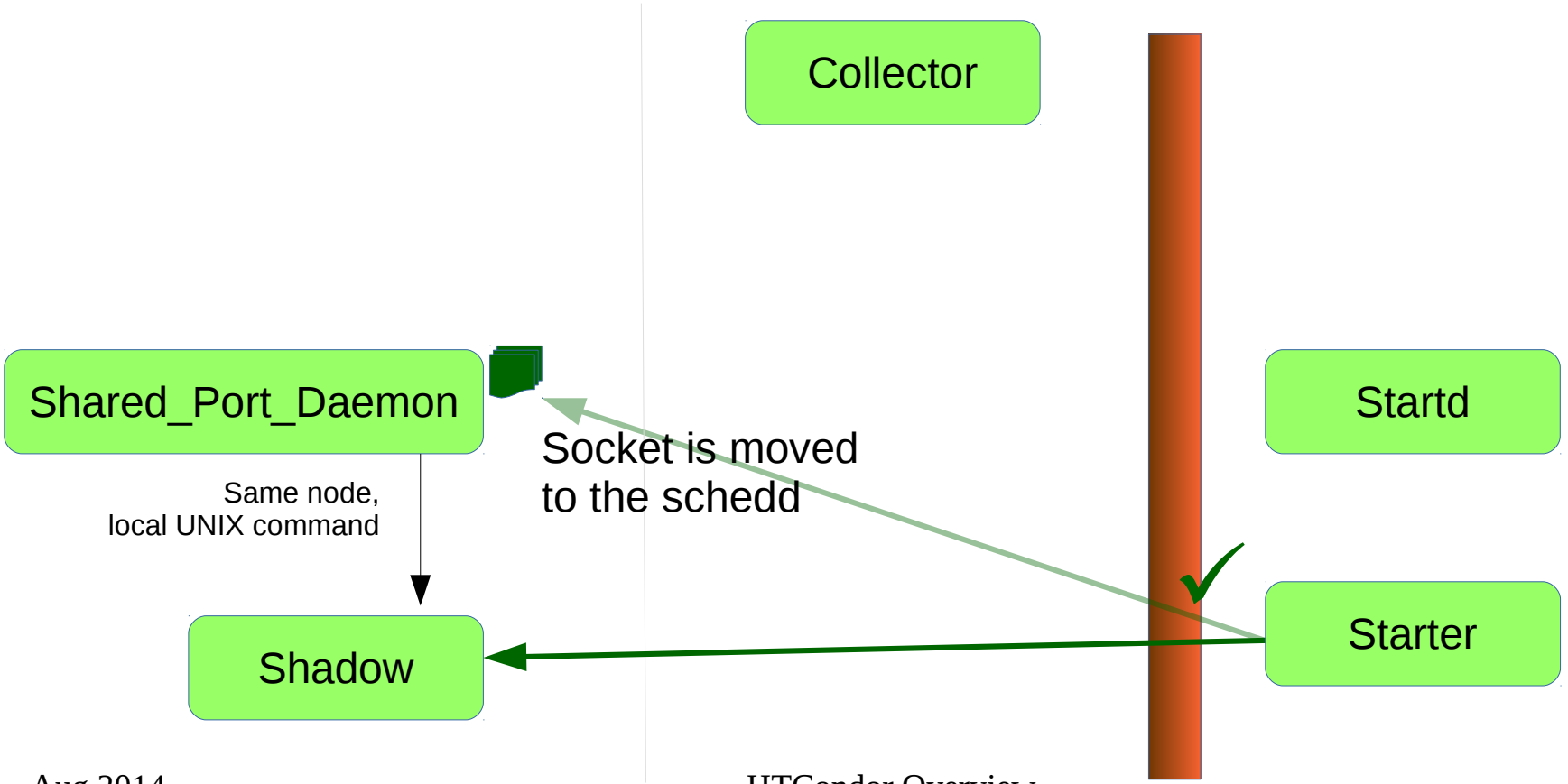
CCB and scalability

HTCondor added shared_port_daemon to multiplex requests on a single port



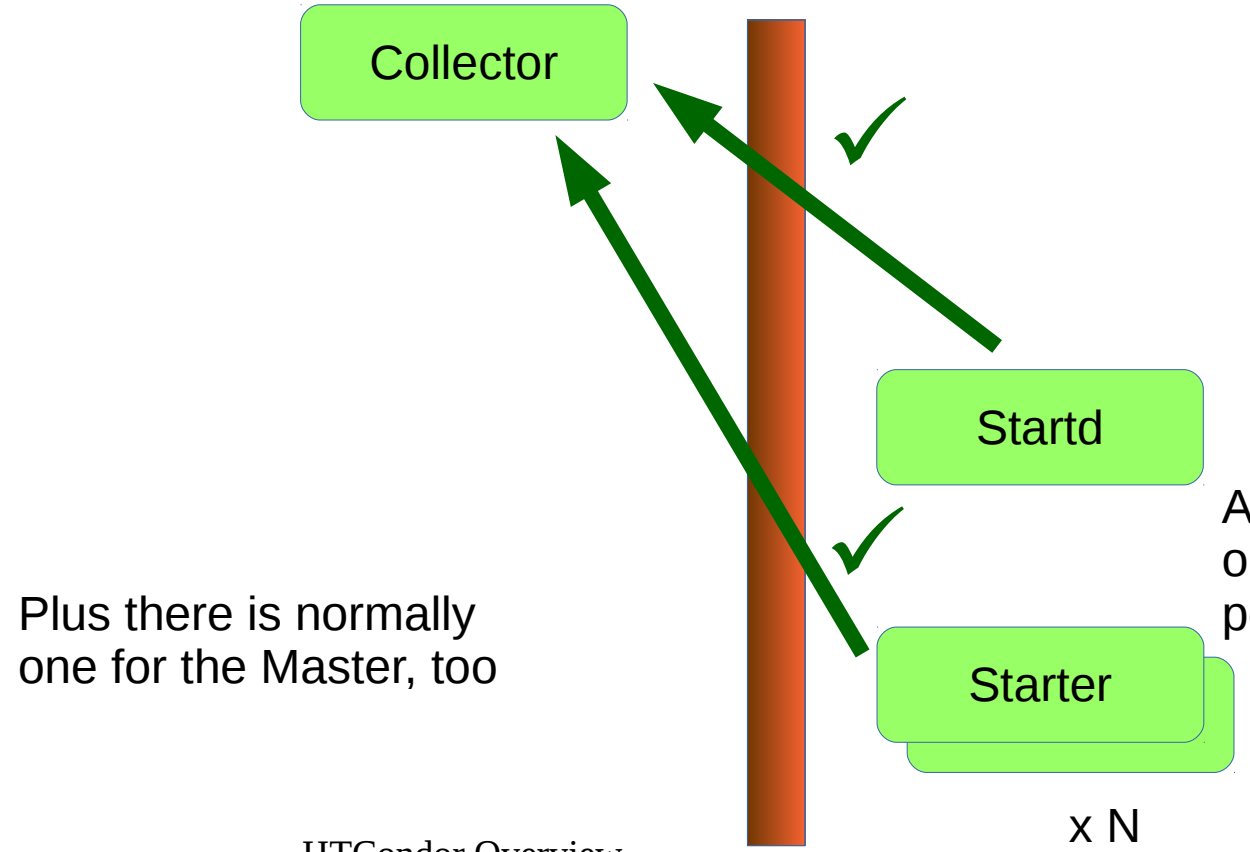
CCB and scalability

Can be used by starter to contact the Shadow, too



CCB and scalability

Starter also accepts incoming connections
Thus needs a CCB connection

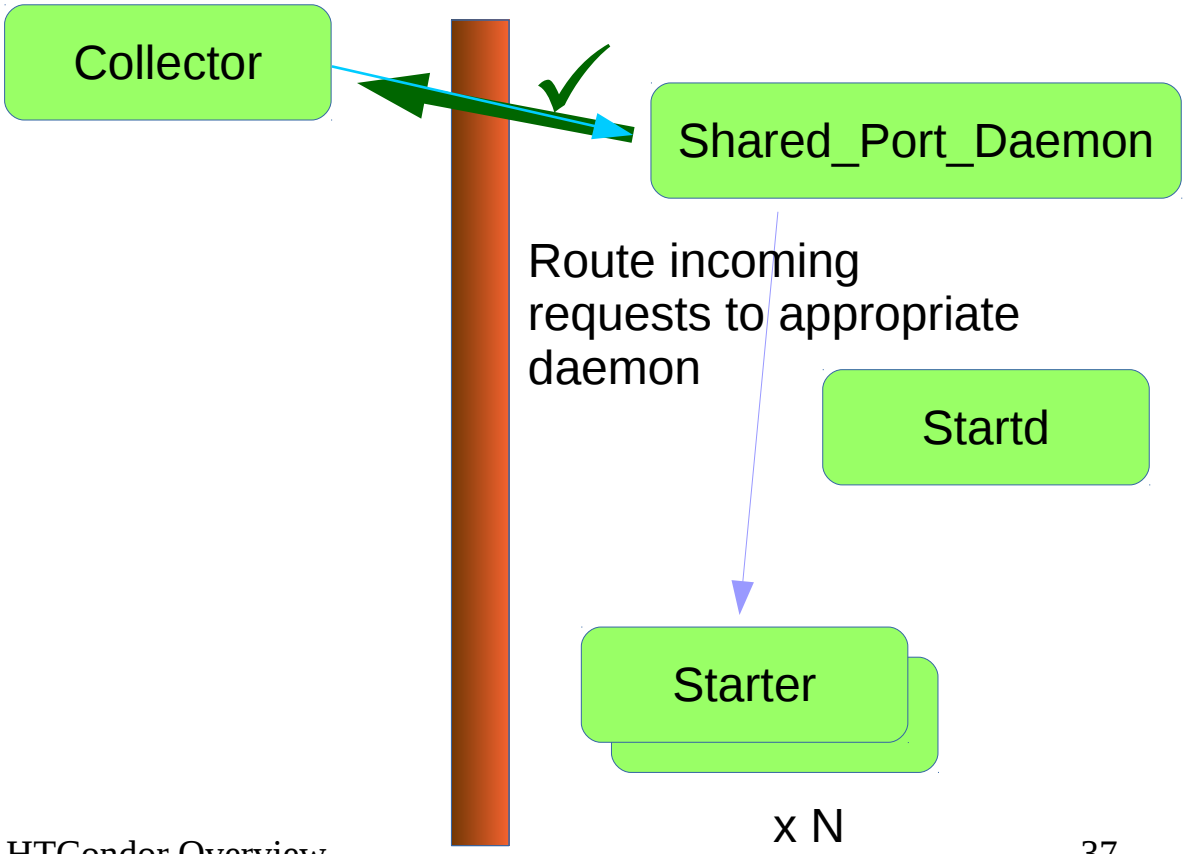


Plus there is normally one for the Master, too

And there is one Starter per slot

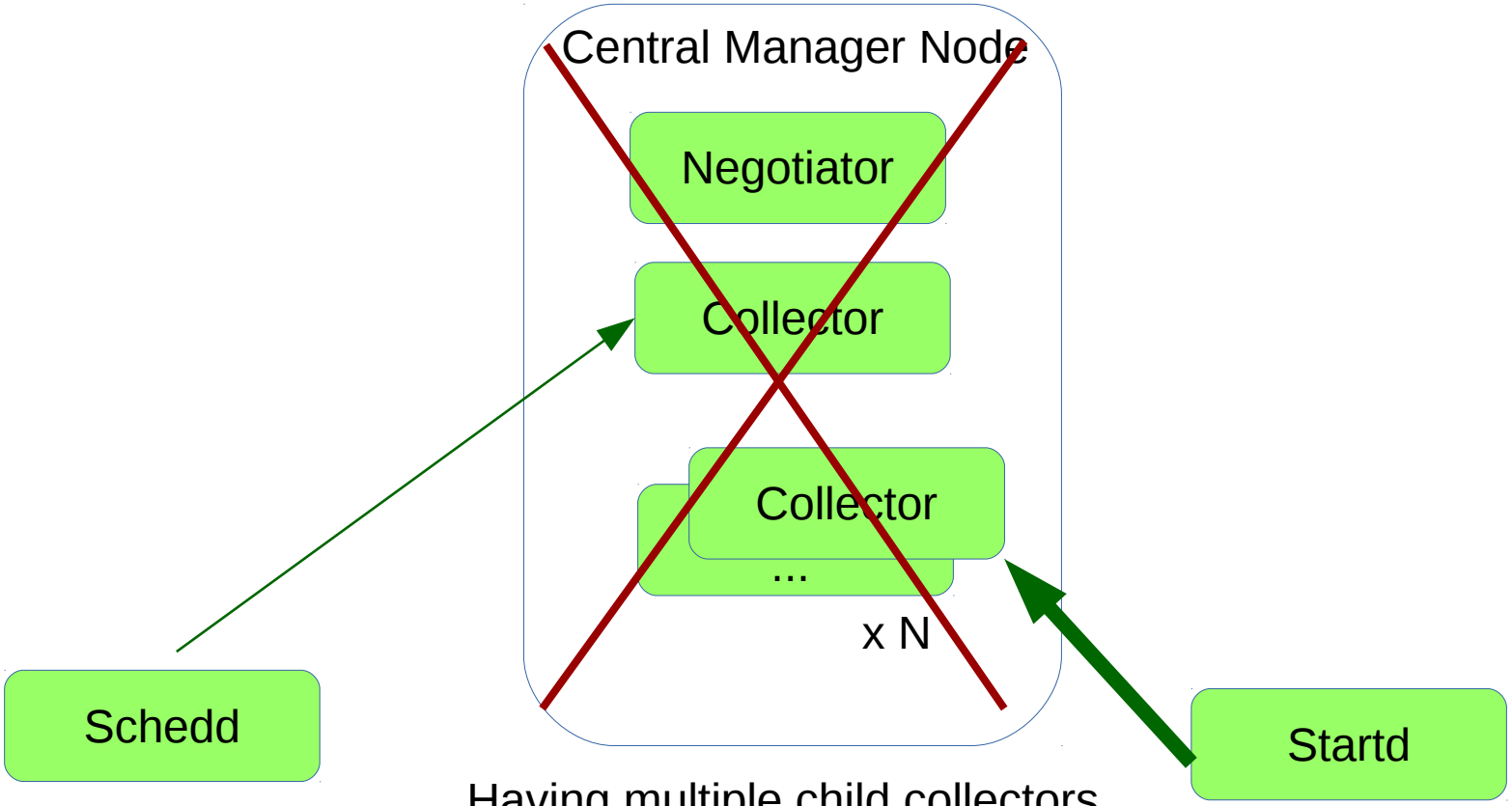
CCB and scalability

Adding a shared_port_daemon will cut number of CCB connections to exactly one



High Availability setup

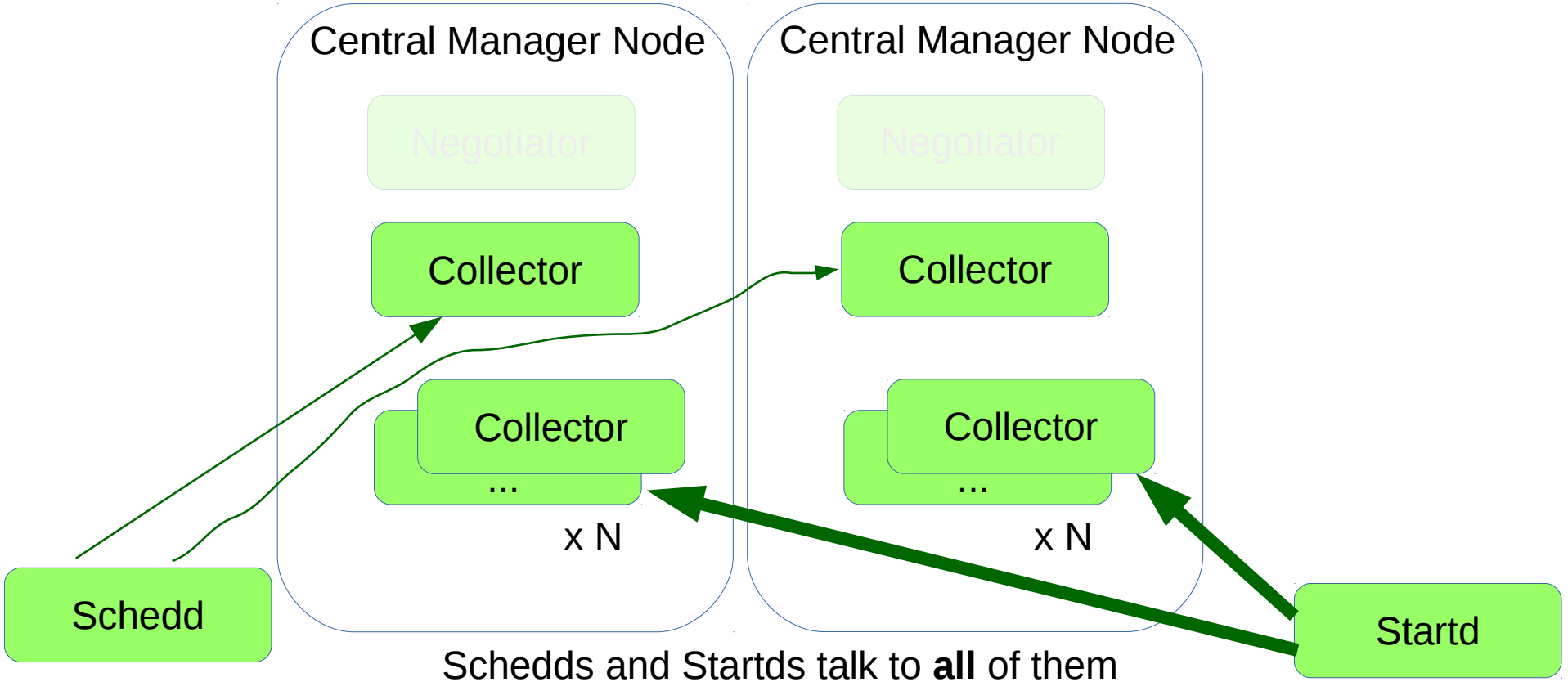
Using a single CM node risky; if it dies, the pool dies with it.



Having multiple child collectors does not help

High Availability setup

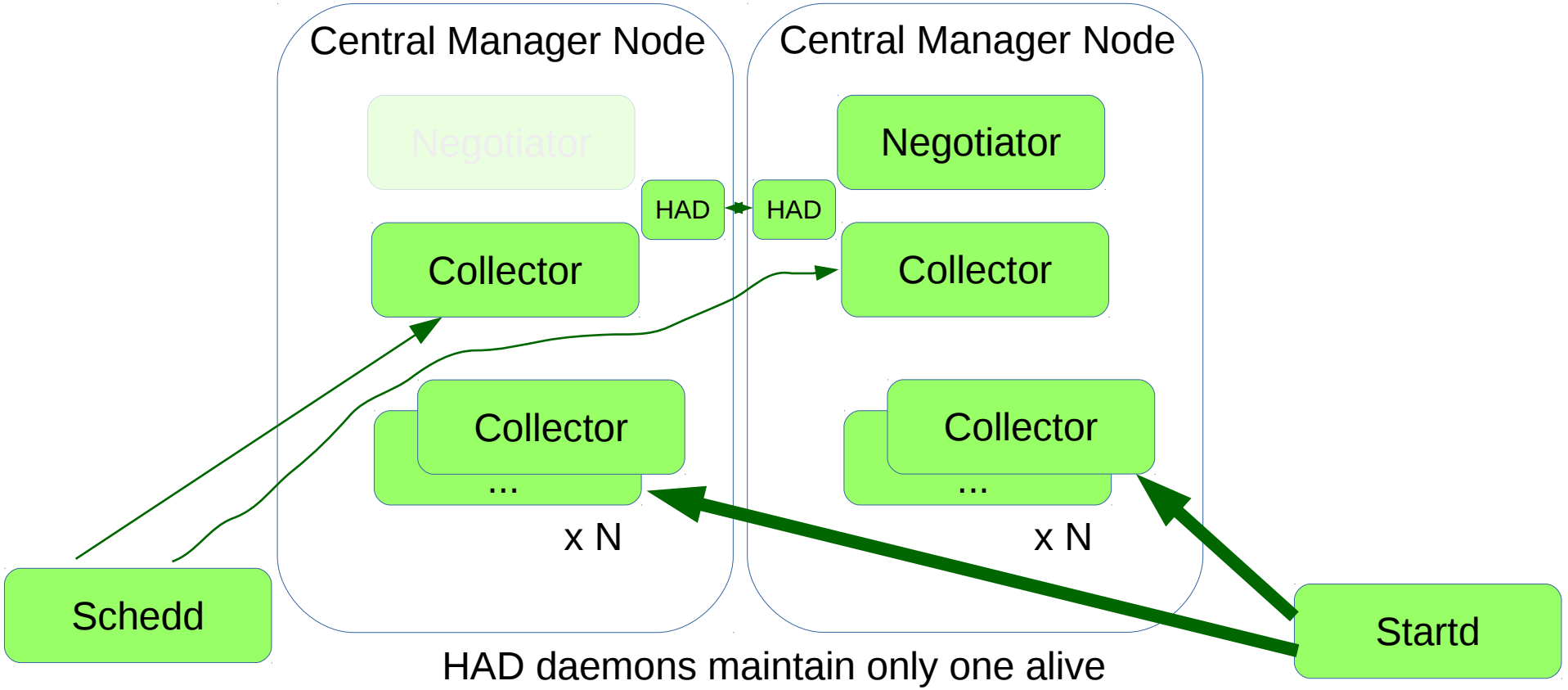
HTCondor allows for 2 or more CM nodes



Schedds and Startds talk to **all** of them
Including one CCB per CM node

High Availability setup

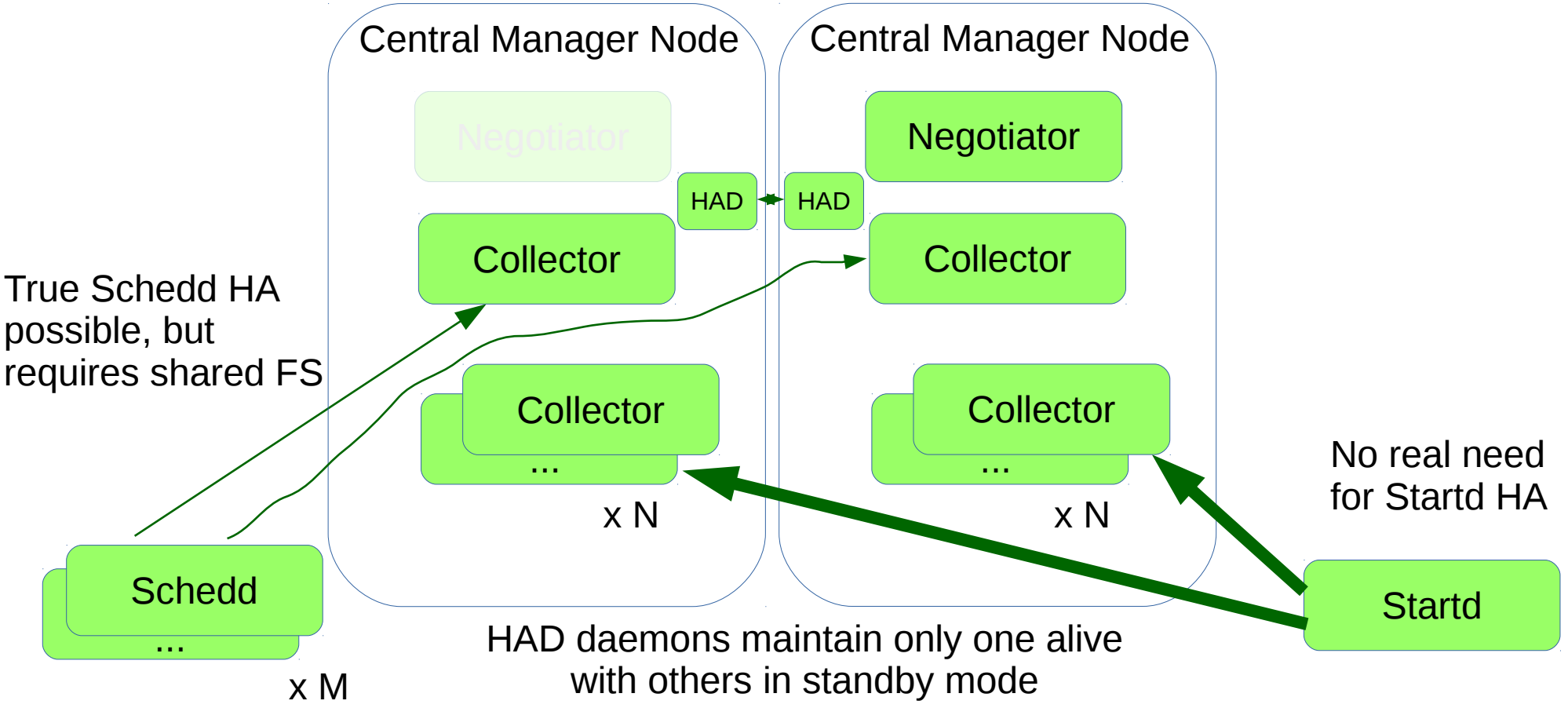
There can be only one active Negotiator,
to make user priority decision



HAD daemons maintain only one alive
with others in standby mode

High Availability setup

Schedd “HA” typically just “partition the jobs between many schedds”
Temporary Schedd downtimes result in other schedds taking over the slots



The end