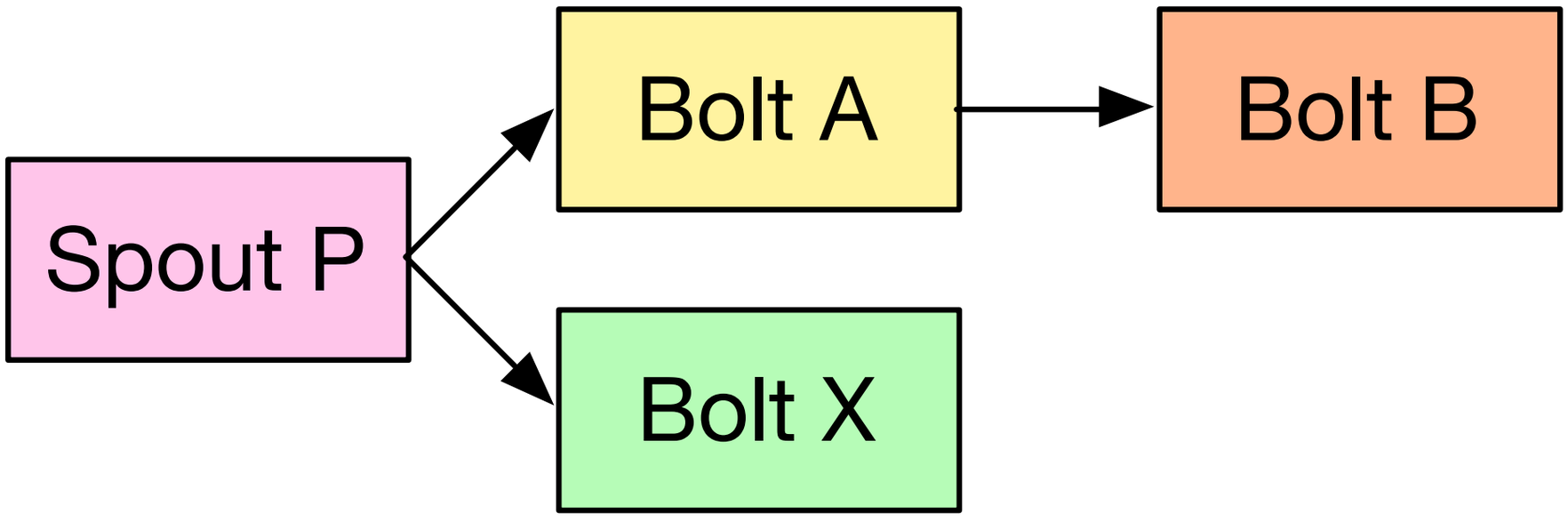
Event Streams at Groupon

Storm, Mesos and Griddle

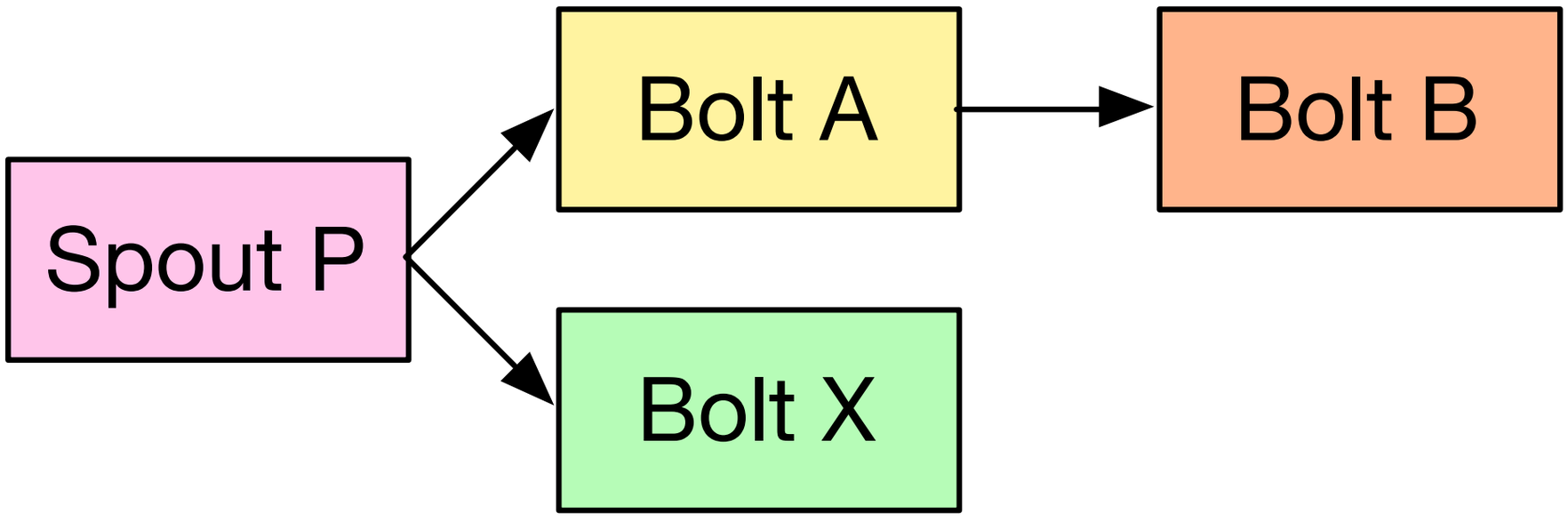
AJ & Erik Weathers

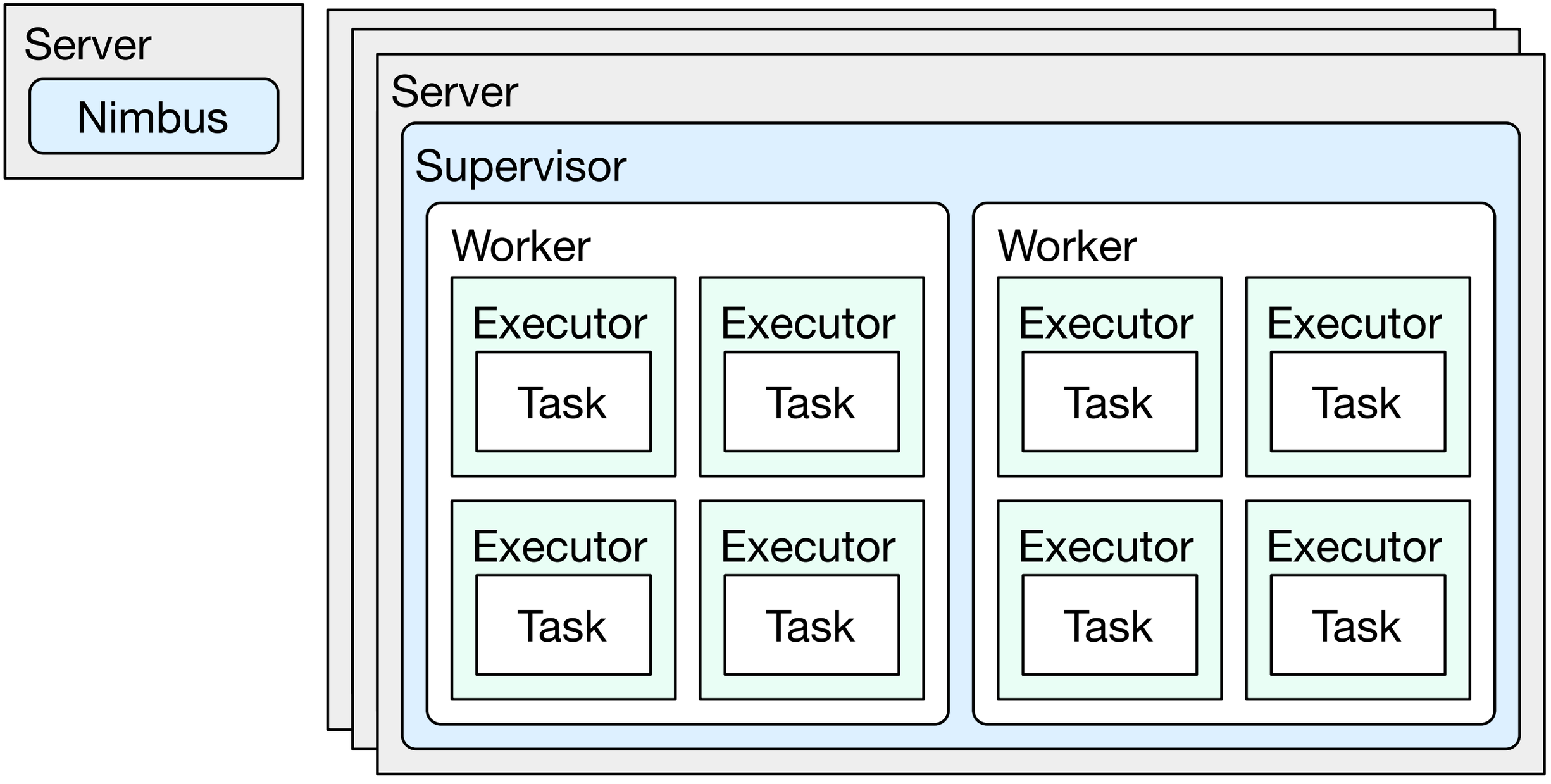
(with special guest Brian McCallister)

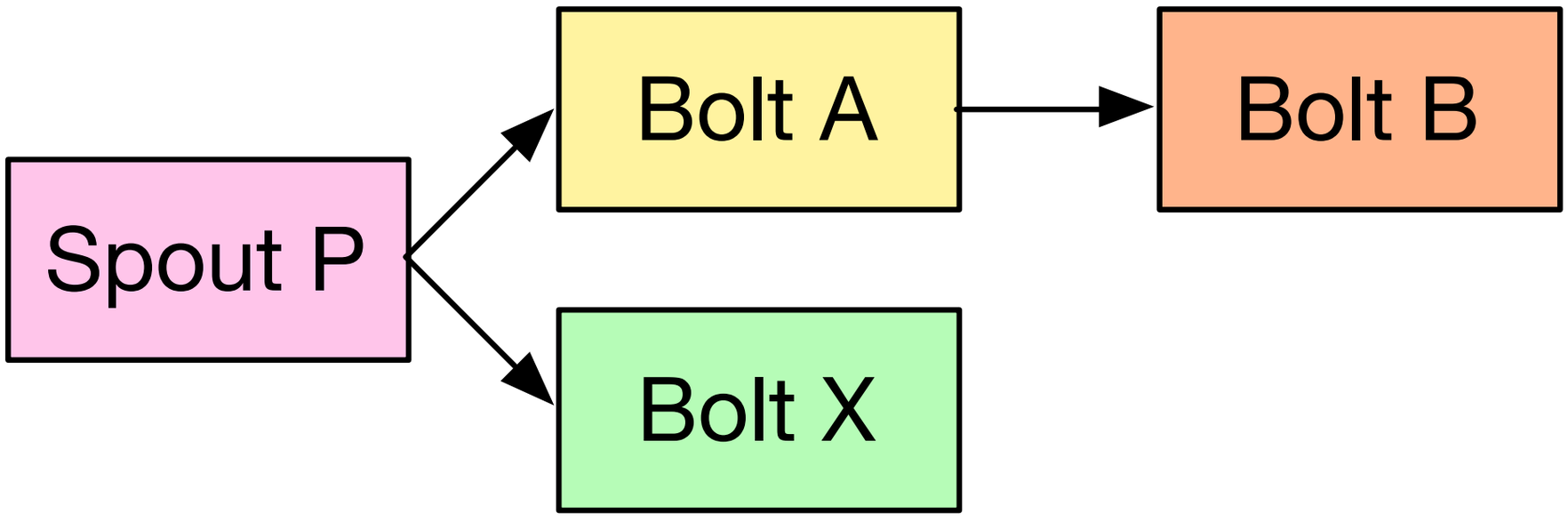
Storm

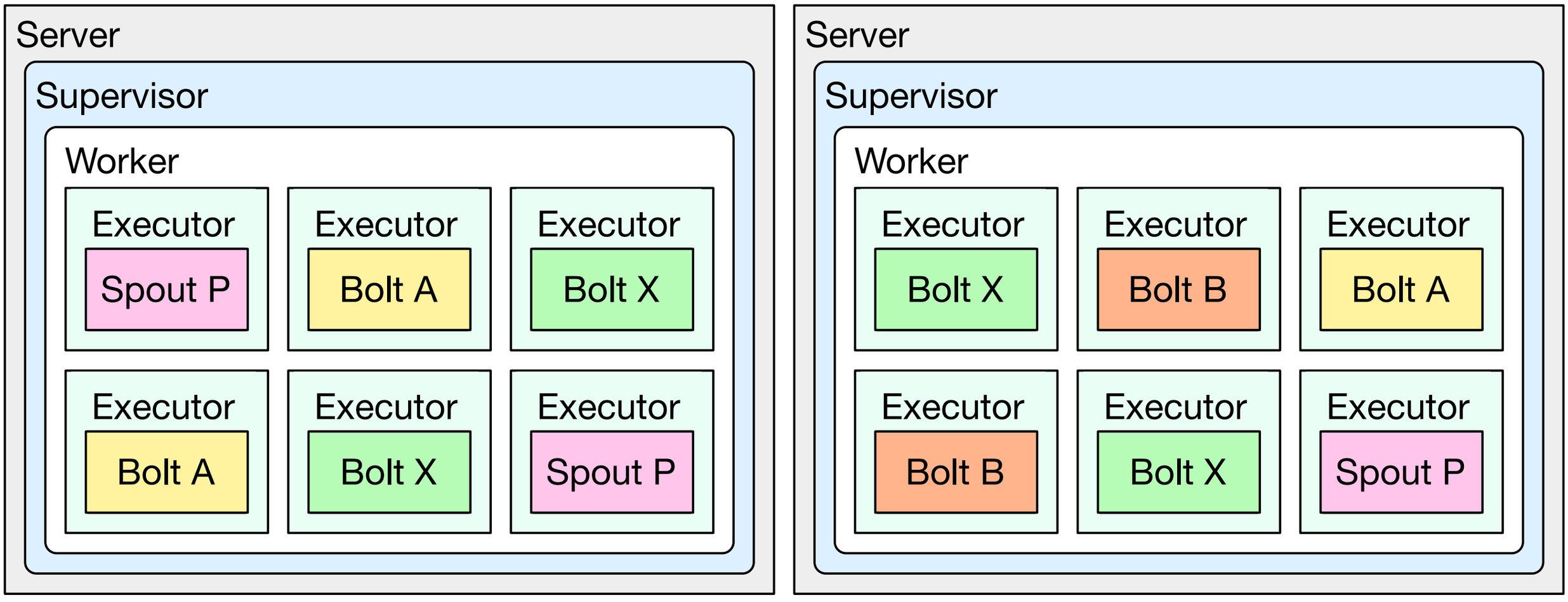

Bolt A
Bolt B
Spout P
Bolt X

```
@Override
public void execute(final Tuple tuple)
{
    Span span = (Span) tuple.getValueByField("span");
    Trace trace = cache.getUnchecked(span.trace_id);
    trace.addSpan(span);

    collector.emit(new Values(trace.getId(), trace));
}
```


Bolt A
Bolt B
Spout P
Bolt X

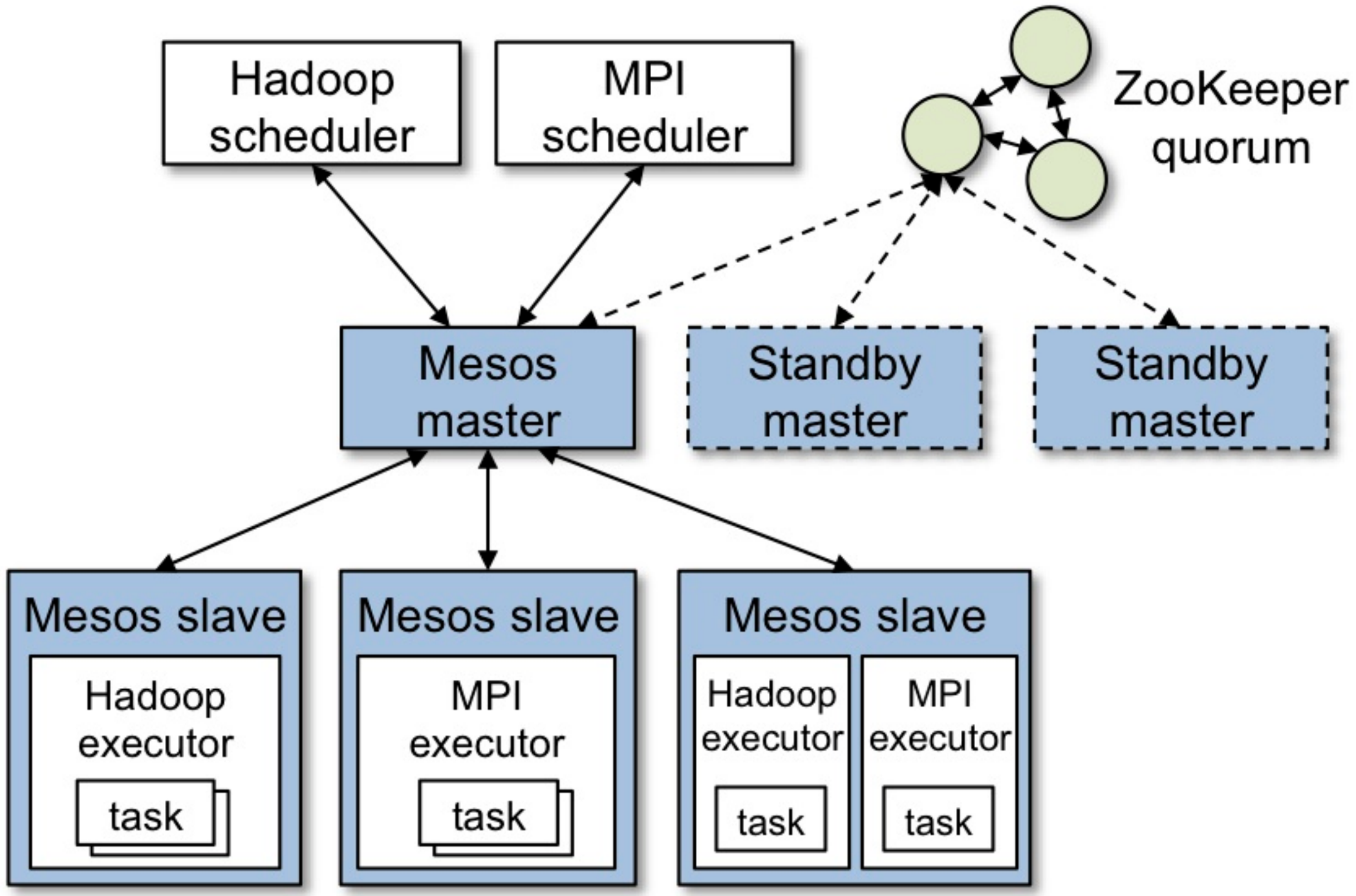

Server
Nimbus
Server
Server
Supervisor
Worker
Executor
Task
Executor
Task
Executor
Task
Executor
Task
Worker
Executor
Task
Executor
Task
Executor
Task
Executor
Task

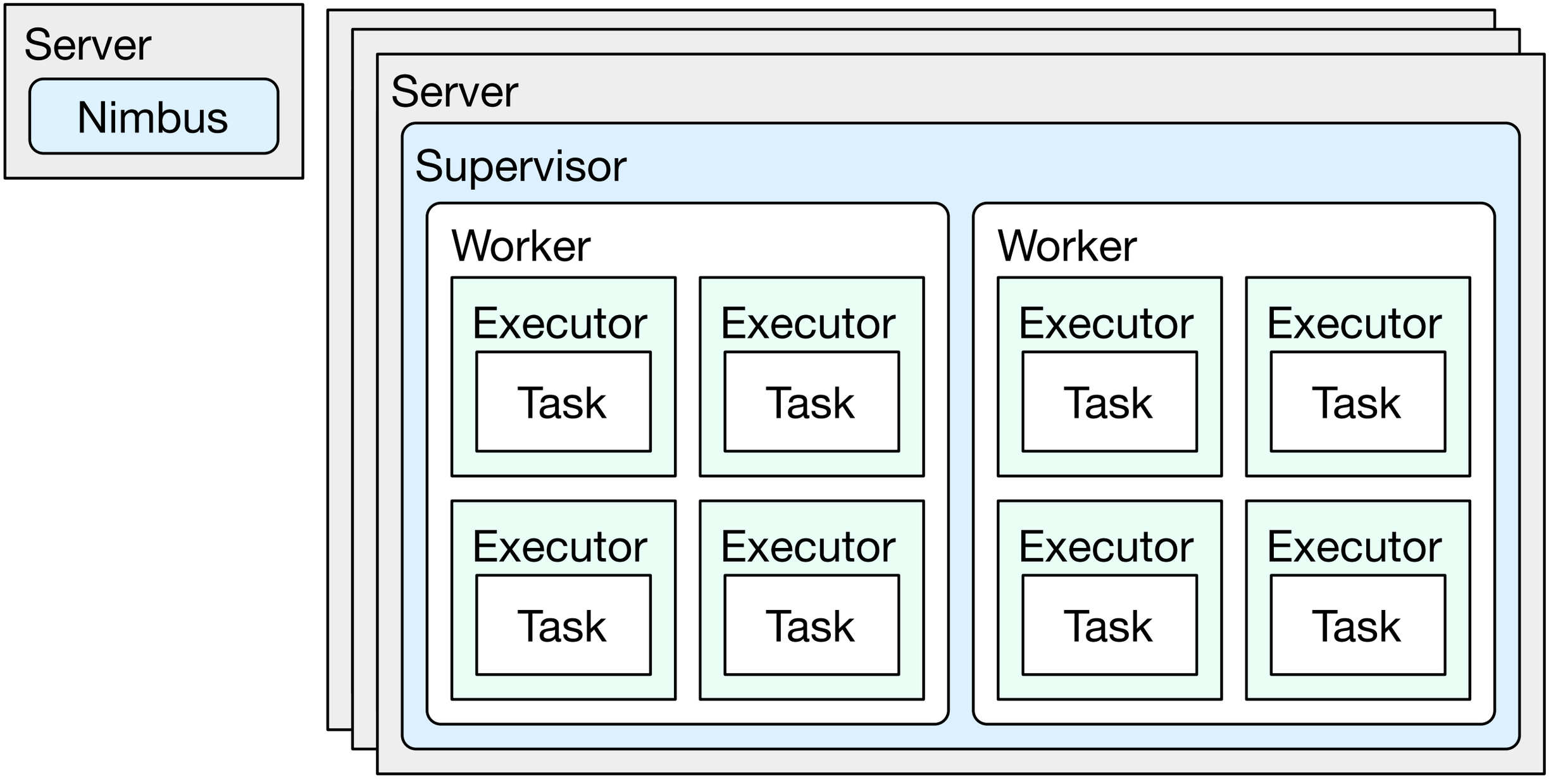

Bolt A
Bolt B
Spout P
Bolt X


Server
Supervisor
Worker
Executor
Spout P
Executor
Bolt A
Executor
Bolt X
Executor
Bolt A
Executor
Bolt X
Executor
Spout P
Server
Supervisor
Worker
Executor
Bolt X
Executor
Bolt B
Executor
Bolt A
Executor
Bolt B
Executor
Bolt X
Executor
Spout P

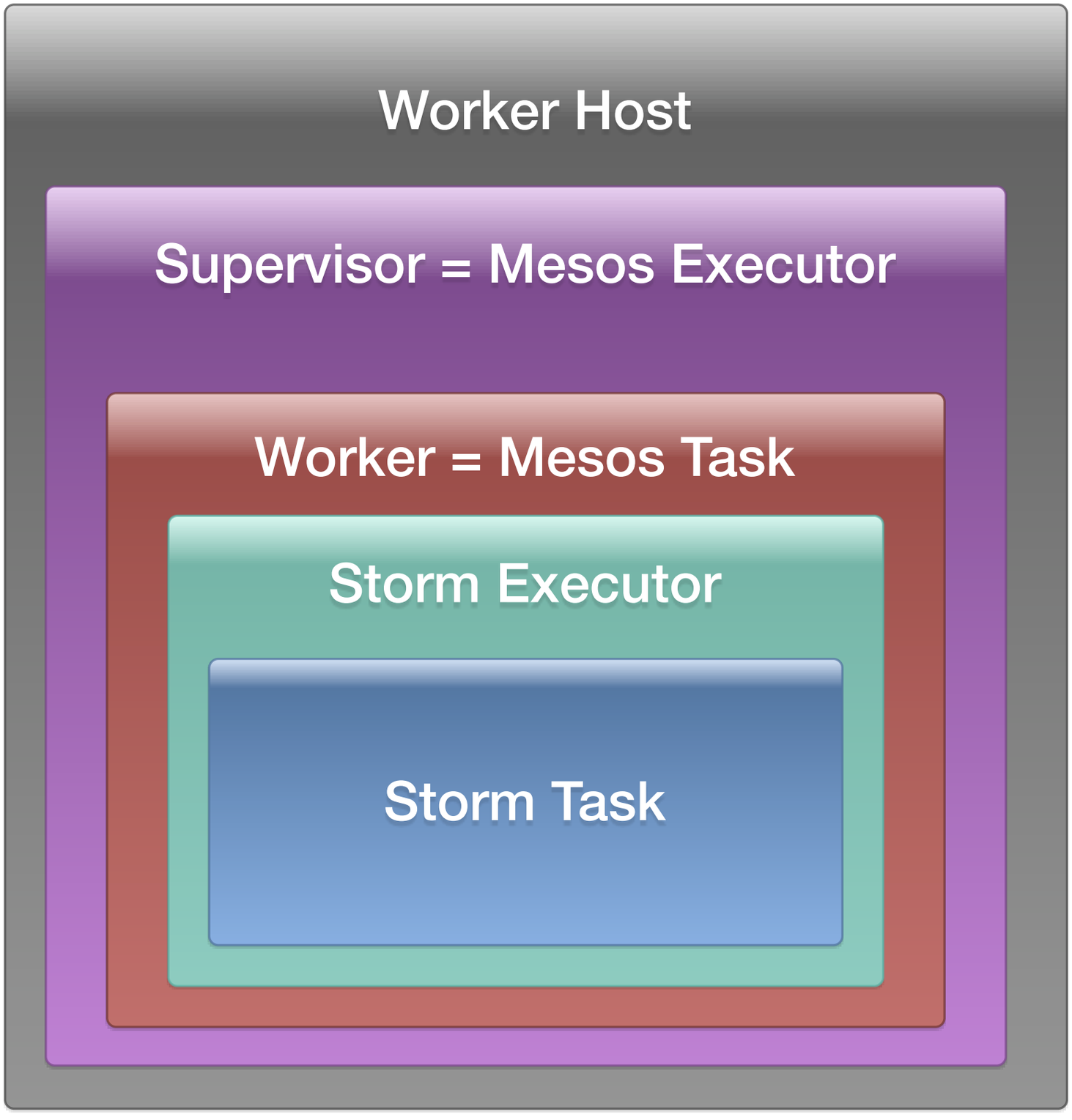
Storm on Mesos

Mesos


Hadoop scheduler
MPI scheduler
ZooKeeper quorum
Mesos master
Standby master
Standby master
Mesos slave
Hadoop executor
task
Mesos slave
MPI executor
task
Mesos slave
Hadoop executor
task
MPI executor
task


Server
Nimbus
Server
Server
Supervisor
Worker
Executor
Task
Executor
Task
Executor
Task
Executor
Task
Worker
Executor
Task
Executor
Task
Executor
Task
Executor
Task

Terminology Clarity

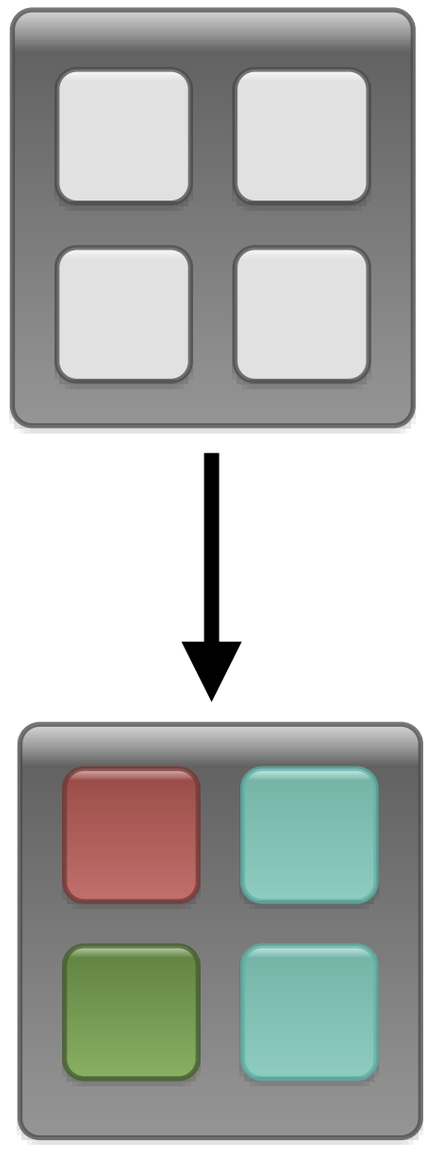


Storm Mesos Framework

- Bridges Storm & Mesos
- Implements interfaces from each
  - Storm's *INimbus*
  - Mesos's *Executor* and *Scheduler*
- Storm nimbus & supervisor daemons run within the Framework processes

Resource Model: Storm

- Worker Slots
- {host, port}
- CPU, Mem ??
- static set of Slots in native Storm

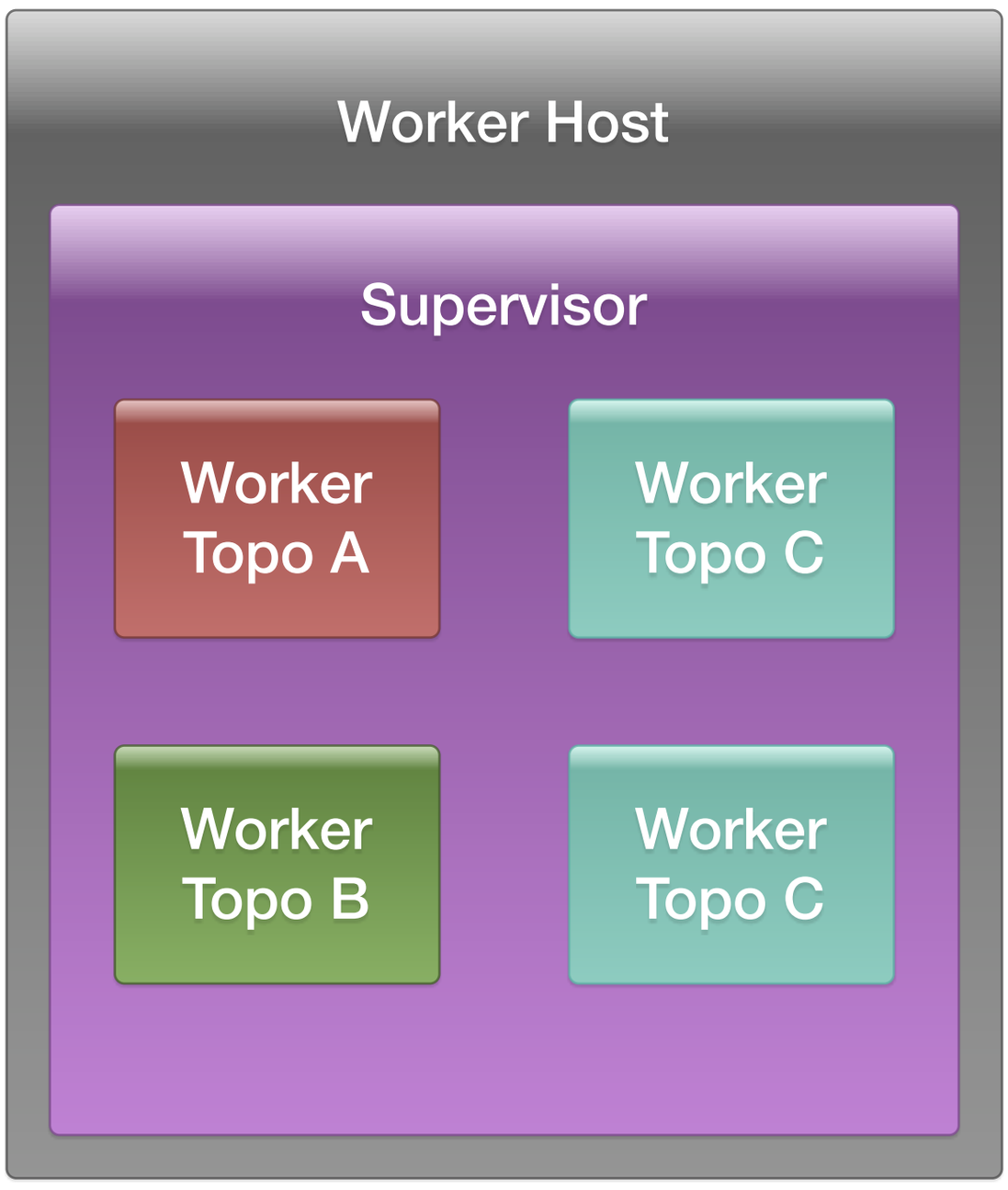


Resource Model: Mesos

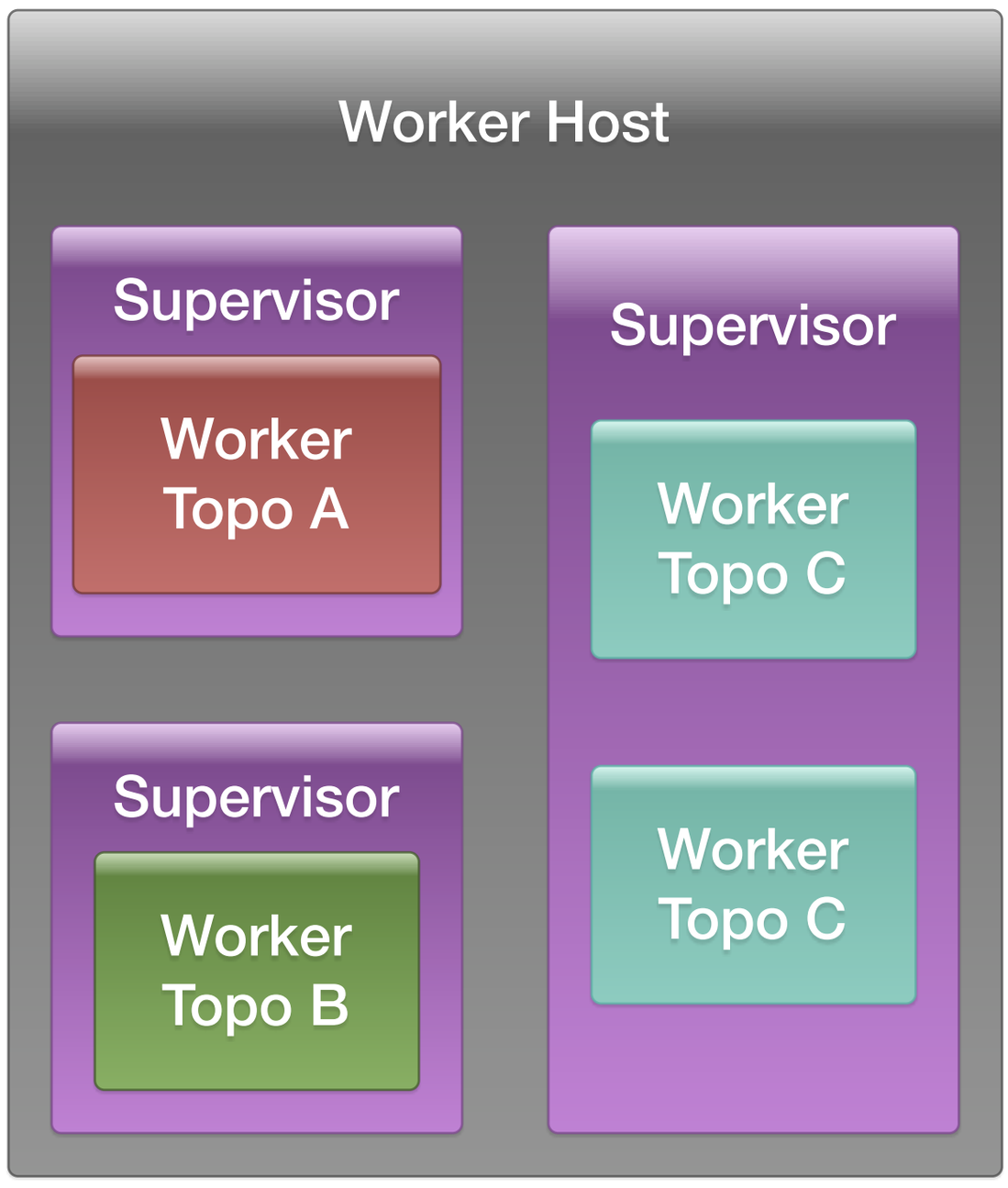
- Schedulers receive Offers of CPU, Mem, etc.
- Executors
- Launch Tasks

More Supervisors

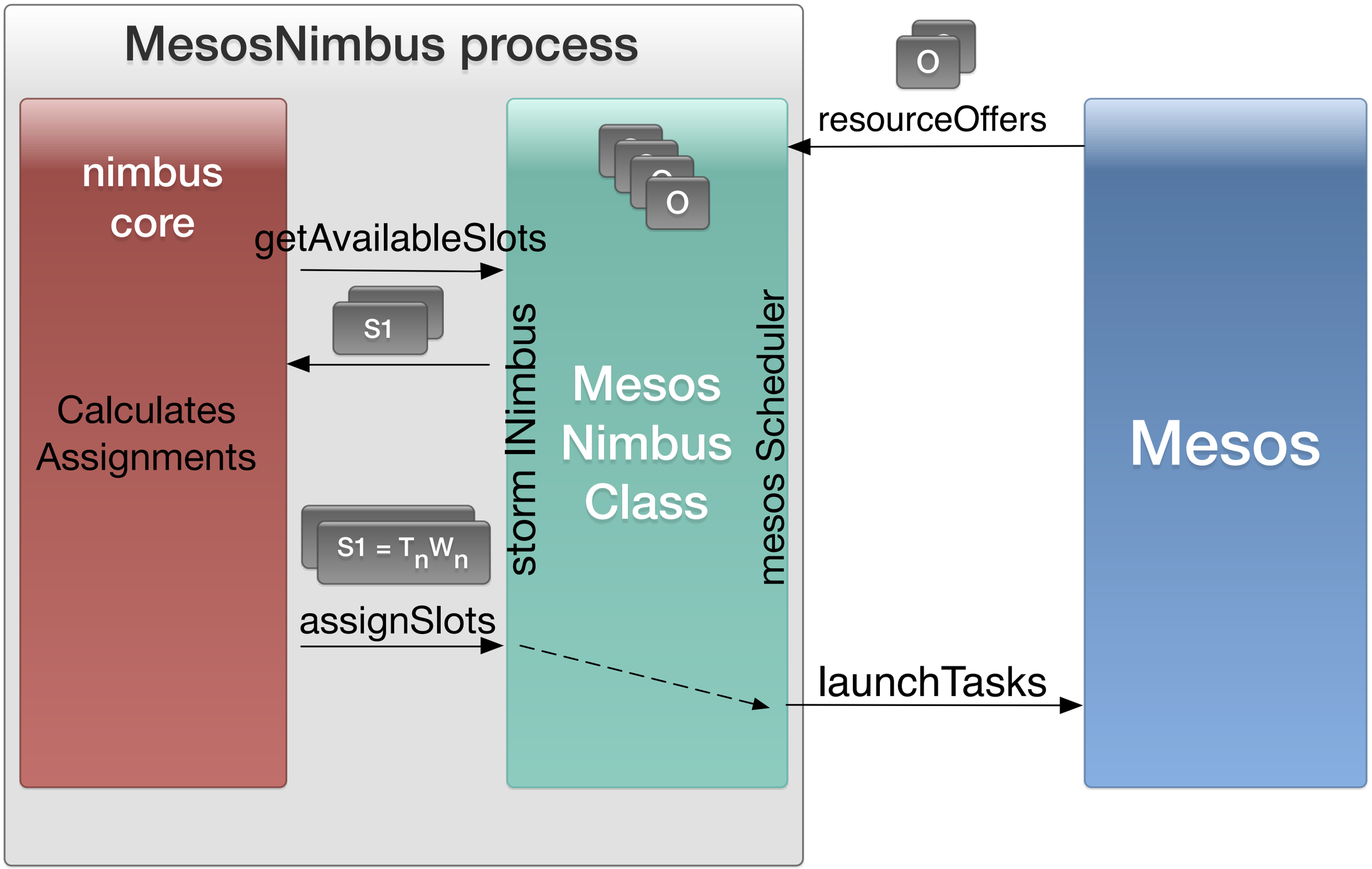
Native Storm



Storm on Mesos



MesosNimbus



Groupon Storm-as-a-Service

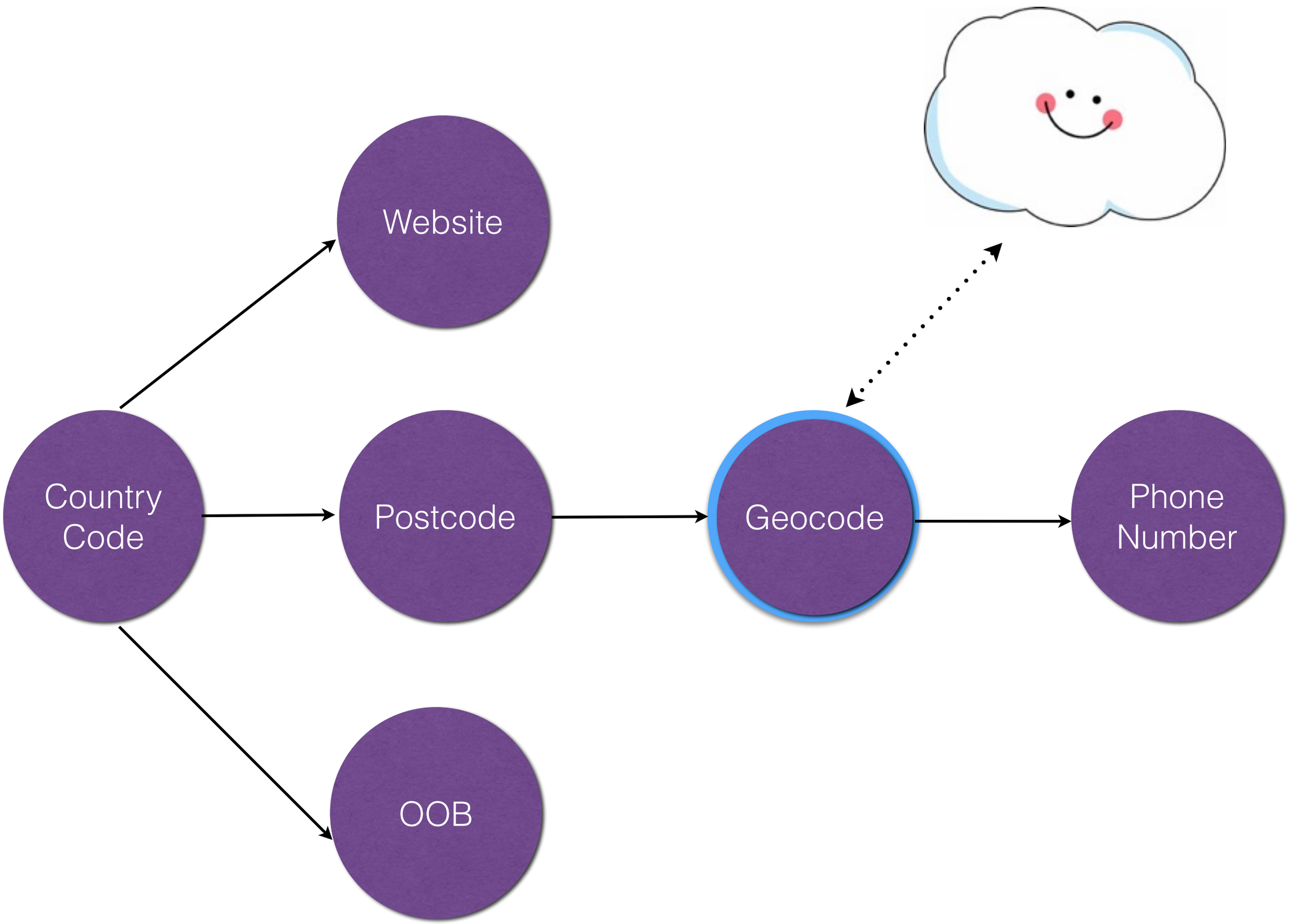
- Mesos cluster dedicated to Storm
- Submitter application for gatekeeping
- Rsync Nimbus local state
- Logging library for sending to Splunk & Kafka
- Metrics library for sending to Monitoring
  - implements Storm's *IMetricsConsumer* interface

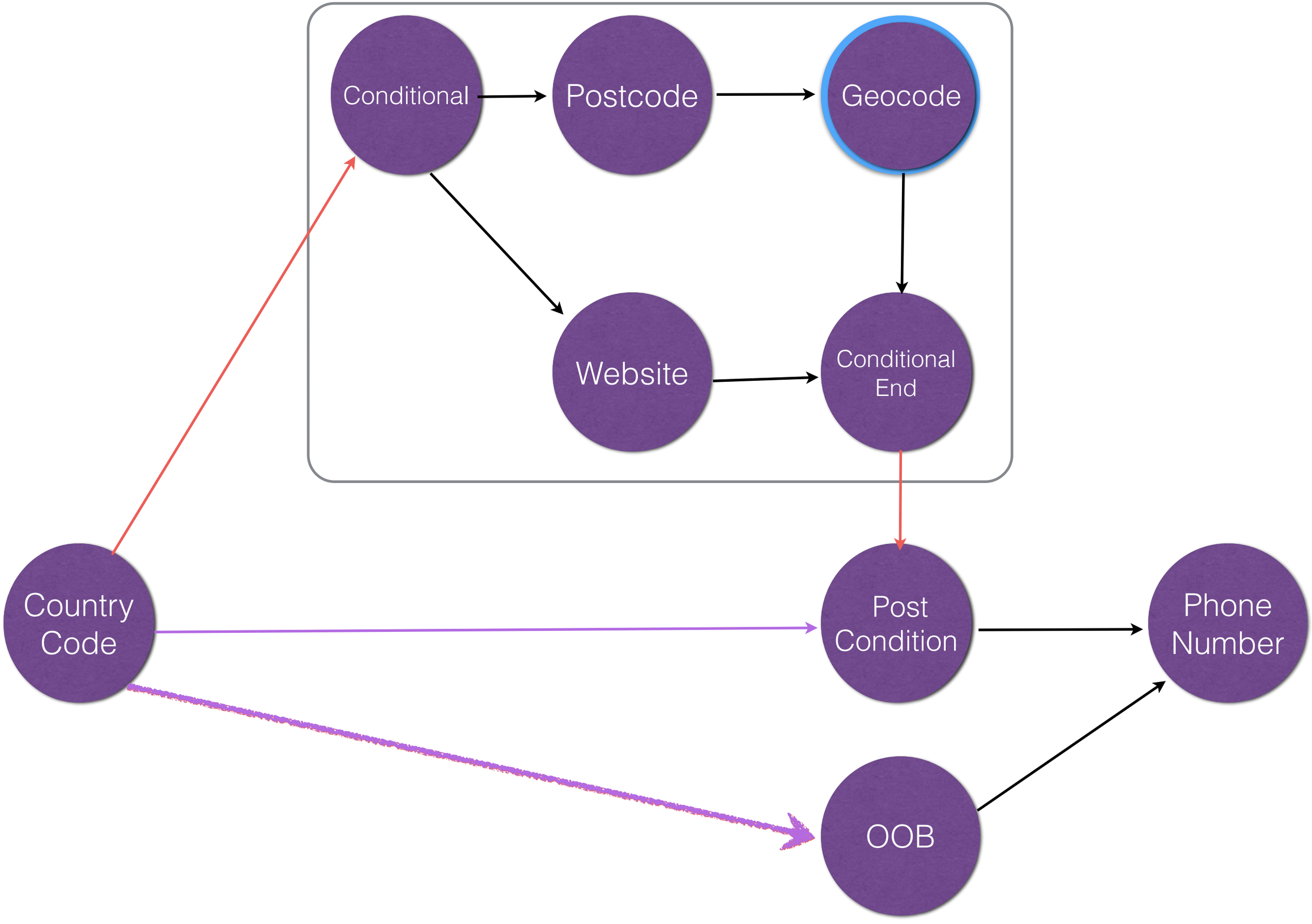
Pros vs. Native Storm

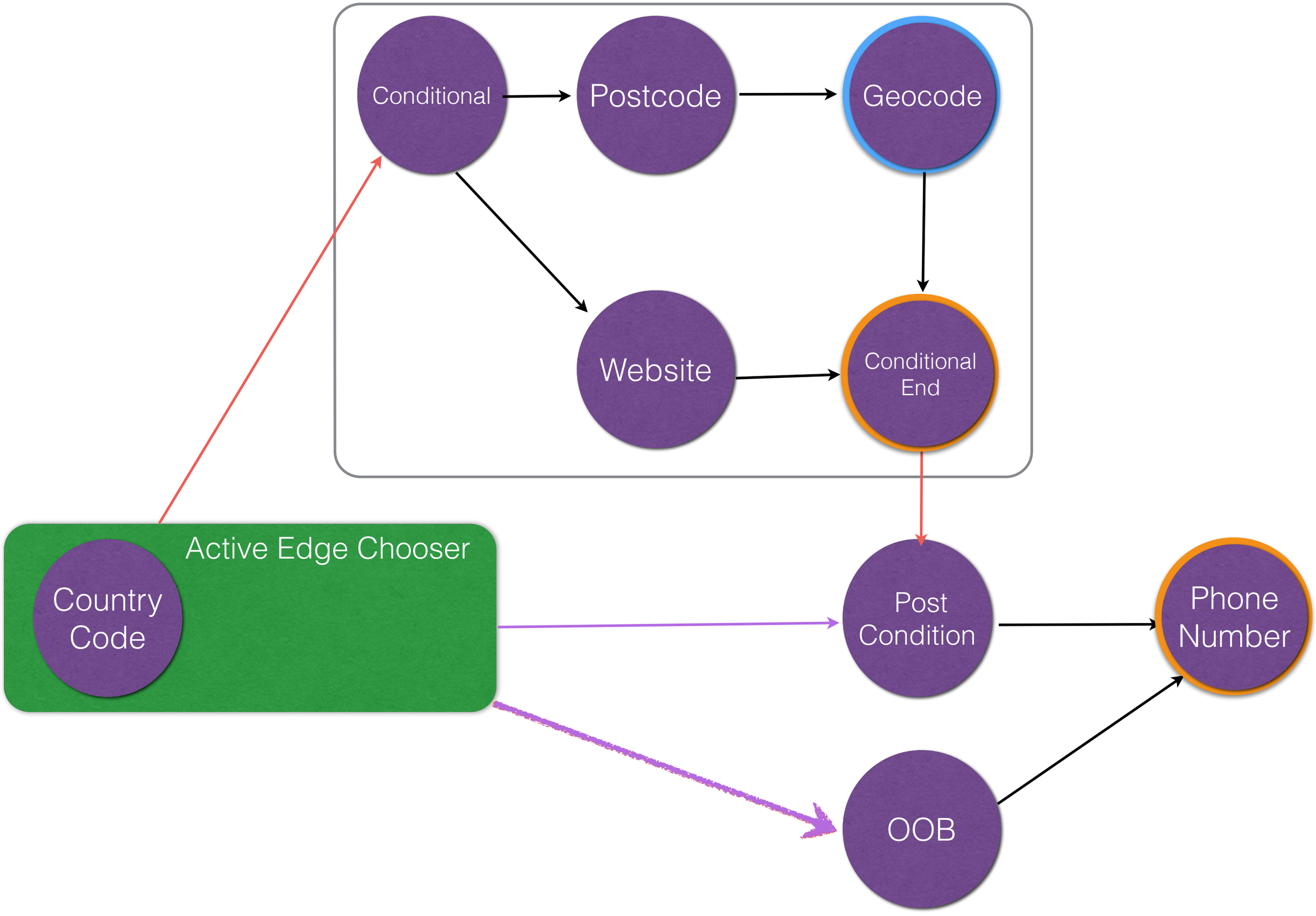
- isolation for multi-tenancy
  - Storm's isolation scheduler is static
- flexibility for number & size of worker processes
- avoid a bunch of separate under-utilized clusters
- team acts as centralized resource for Storm usage and debugging
- consistent operational visibility

Griddle

- DSP-like workflow
- Adjacency Graph Syntax
- Mechanical Sympathy


Website
Country Code
Postcode
Geocode
Phone Number
OOB


Conditional
Postcode
Geocode
Website
Conditional End
Country Code
Post Condition
Phone Number
OOB


Conditional
Postcode
Geocode
Website
Conditional End
Active Edge Chooser
Country Code
Post Condition
Phone Number
OOB

```
class_alias TRIGGER com.groupon.griddle.lib.Trigger
# other aliases elided

# Let's get started
vertex start of COUNTRY_CODE_INFERRER

# Vertices active only for certain countries
vertex begin_country_dependent of TRIGGER
vertex geocoder of GEOCODER
vertex website of WEBSITE_NORMALIZER
vertex postcode of POSTCODE_NORMALIZER
vertex end_country_dependent of TRIGGER aggregates_inputs

# Signal end of conditional processing
vertex country_dependence_done of TRIGGER

# Active for all countries
vertex oo_business of OUT_OF_BUSINESS
# Depends on country which can be mutated by geocoder
vertex phone_number of PHONE_NUM_NORMALIZER aggregates_inputs

# If country deactivated, start will emit to {oo_business, country_dependence_done}
# else will emit to {begin_country_dependent, oo_business}
emit_to {oo_business, begin_country_dependent, country_dependence_done}
        from start with_chooser ACTIVE_EDGE_CHOOSER

# Country dependent adjacencies
emit_to {postcode, website} from begin_country_dependent
emit_to {geocoder} from postcode
emit_to {end_country_dependent} from geocoder
emit_to {end_country_dependent} from website
emit_to {country_dependence_done} from end_country_dependent

# phone number normalizer due to aggregates_inputs will act as post
# deactive branch joining vertex
emit_to {phone_number} from oo_business
emit_to {phone_number} from country_dependence_done
```

```
# Let's get started
vertex start of COUNTRY_CODE_INFERRER

# Vertices active only for certain countries
vertex begin_country_dependent of TRIGGER
vertex geocoder of GEOCODER
vertex website of WEBSITE_NORMALIZER
vertex postcode of POSTCODE_NORMALIZER
```

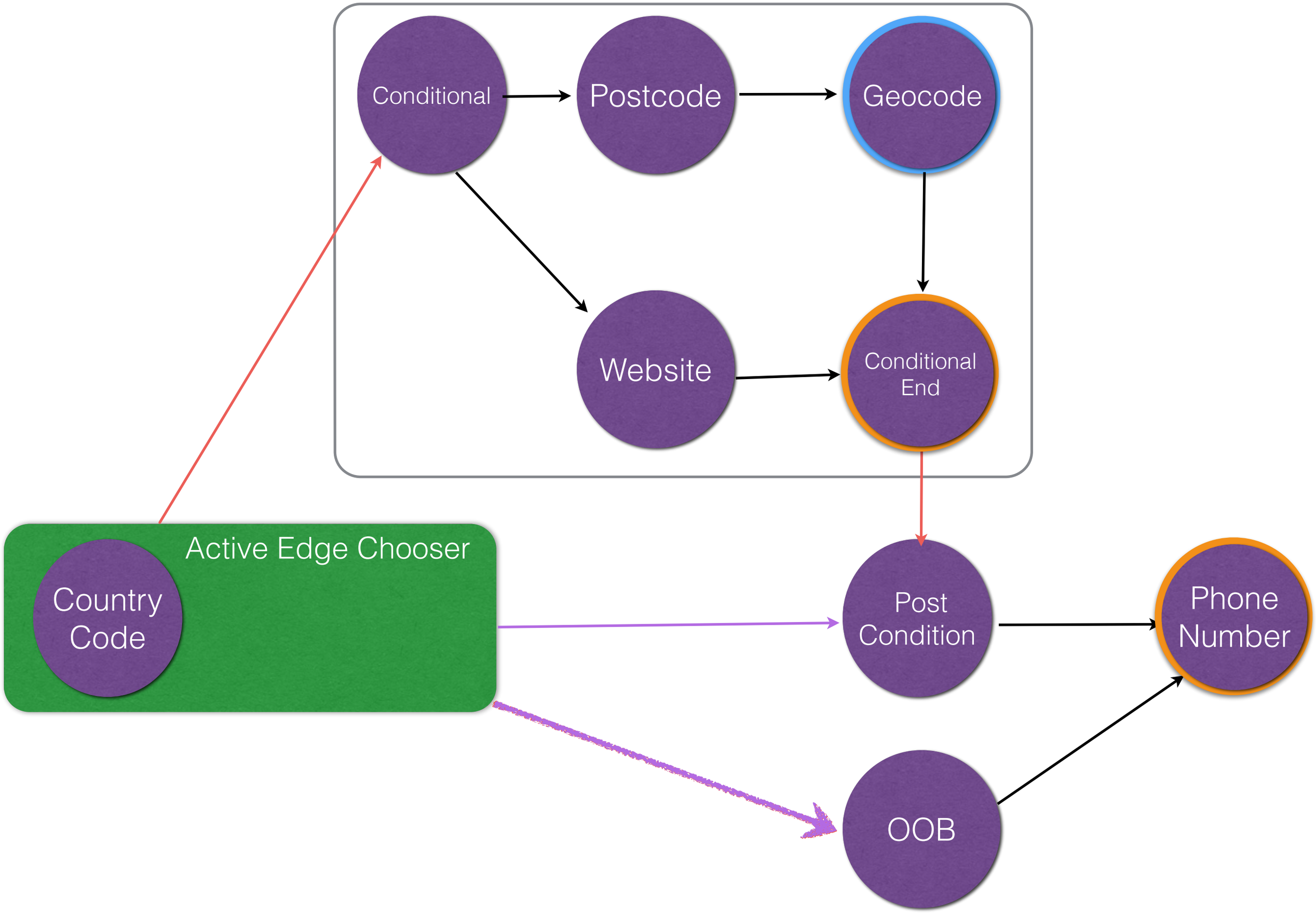
```
# Let's get started
vertex start of COUNTRY_CODE_INFERRER

# Vertices active only for certain countries
vertex begin_country_dependent of TRIGGER
vertex geocoder of GEOCODER
vertex website of WEBSITE_NORMALIZER
vertex postcode of POSTCODE_NORMALIZER
```

```
# Country dependent adjacencies
emit_to {postcode, website}
        from begin_country_dependent
emit_to {geocoder} from postcode
emit_to {end_country_dependent} from geocoder
emit_to {end_country_dependent} from website
emit_to {country_dependence_done}
        from end_country_dependent
```

```
emit_to {geocoder} from postcode
```

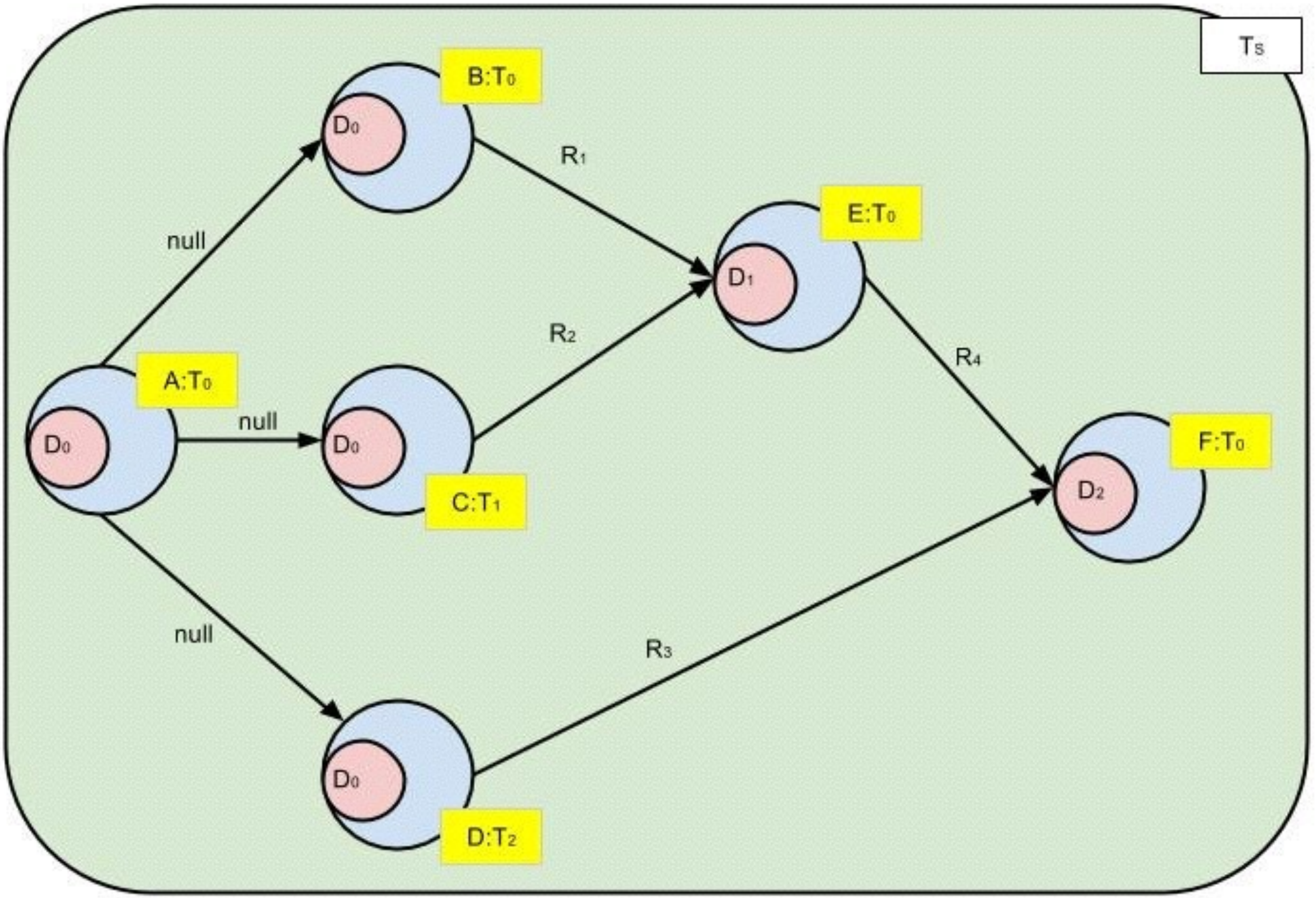
```
emit_to {oo_business,
         begin_country_dependent,
         country_dependence_done}
from start with_chooser ACTIVE_EDGE_CHOOSER
```

```
emit_to {oo_business,
         begin_country_dependent,
         country_dependence_done}
from start with_chooser ACTIVE_EDGE_CHOOSER
```


Conditional
Postcode
Geocode
Website
Conditional End
Active Edge Chooser
Country Code
Post Condition
Phone Number
OOB

What does the DSL give you

- Griddle compiler creates a binary graph
- Binary graph processed with runtime that provides optimal concurrency


Ts
B:T0
D0
R1
null
E:T0
D1
R2
R4
A:T0
null
D0
D0
F:T0
D2
C:T1
null
R3
D0
D:T2

The End

Future Work: Storm

- make parallelism configurable at runtime
- debuggability (stderr/out logging, history)
- metrics scalability
- replacement IScheduler to avoid big topologies starving small topologies