TIBCO BusinessEvents 5.1

Load Balancer

Ashwin Jayaprakash

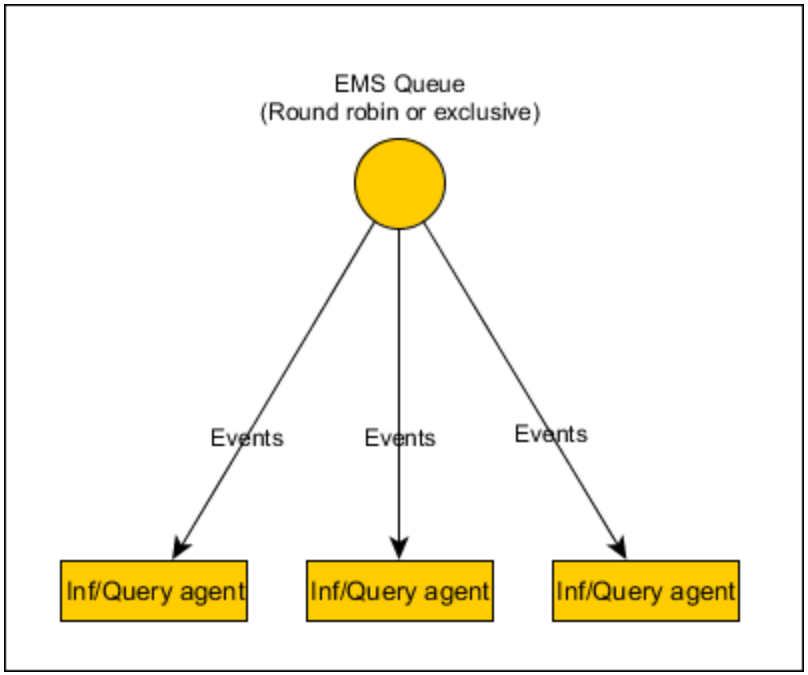
Sep 2012

Topics

- What is it?
- How does it work?
- Some details
- Quick demo
- Q & A

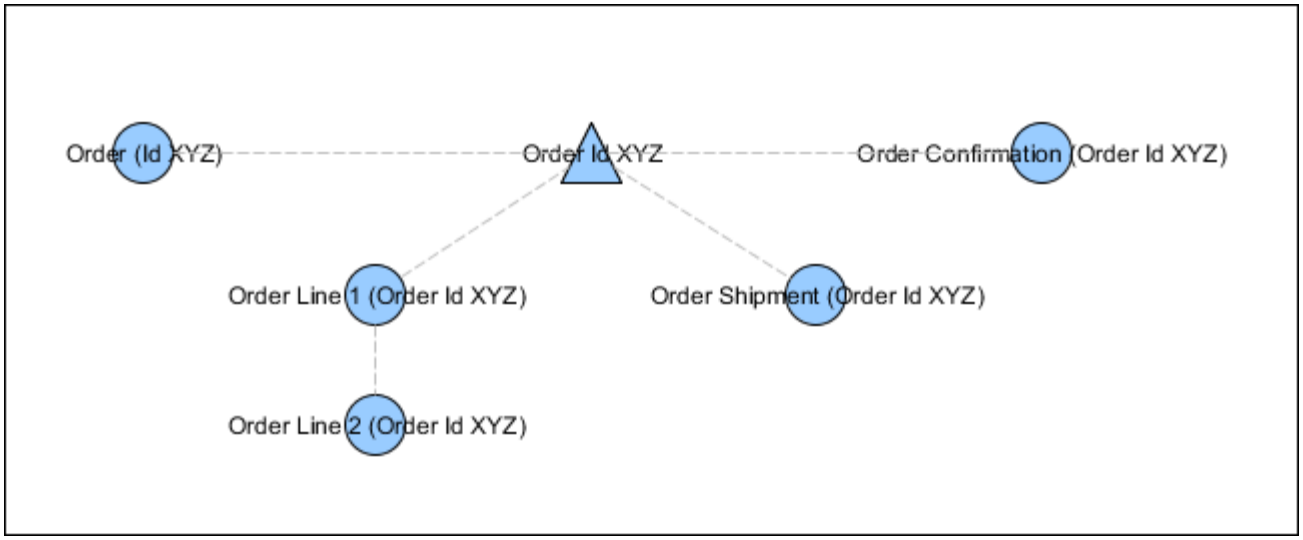
Load balancing - Method 1

BE 3.0 style

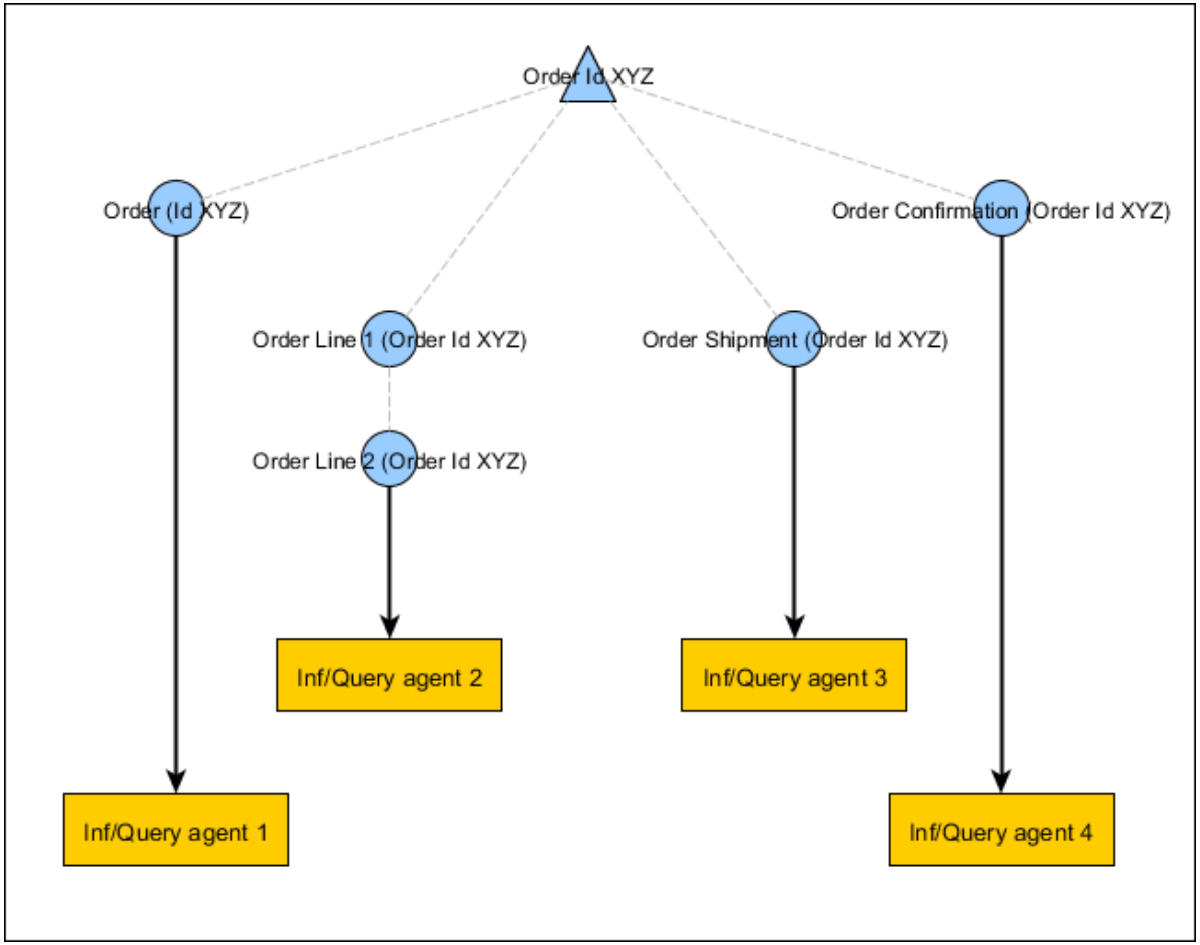


Load balancing - Method 1

Graph of related objects



Load balancing - Method 1



Problems with Method 1

- Random event distribution
  - Related events to different/random engines
- `C_Lock(orderId, global)`
  - Global lock to avoid race conditions
  - Threads stuck on global locks
  - Low cluster wide throughput
- Local cache useless
  - Graph modified on multiple agents
  - Always get from global cache
  - High latency

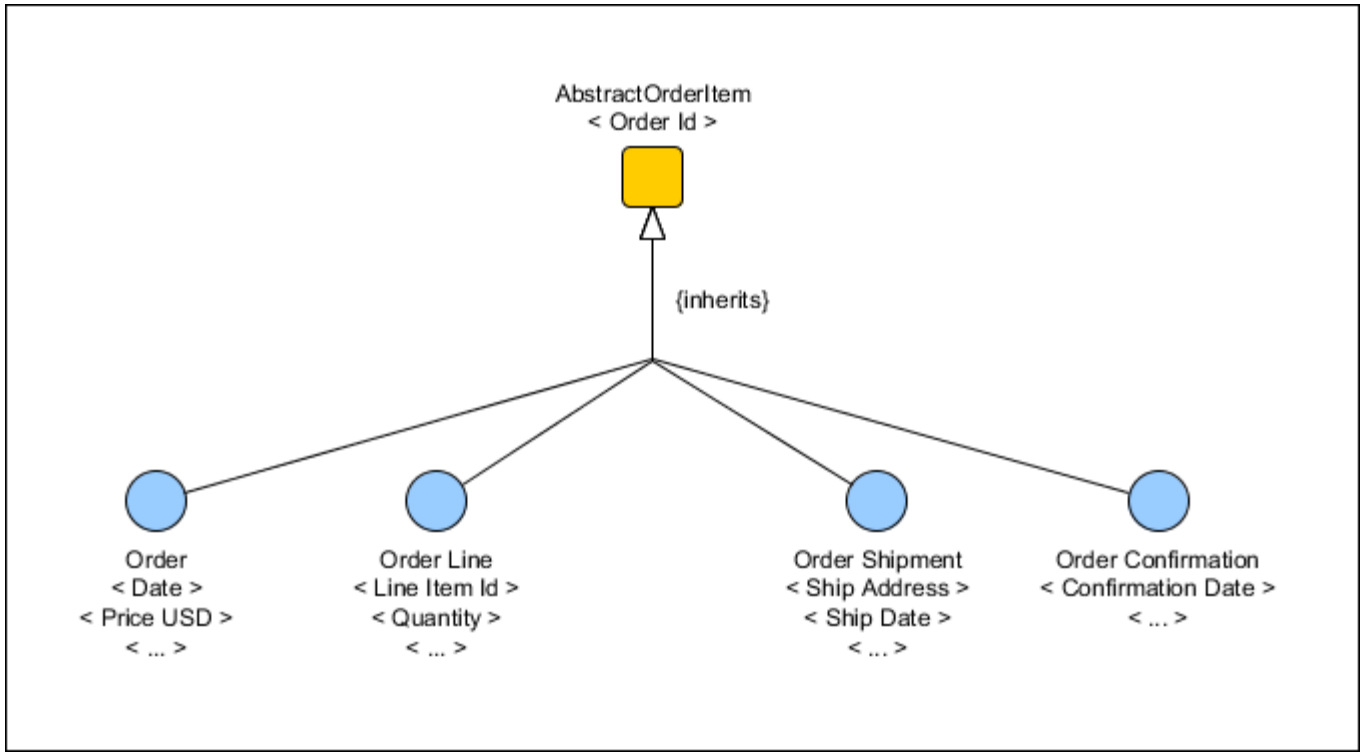
Load balancing - Method 2

- Content based load balancing
  - Peek into data

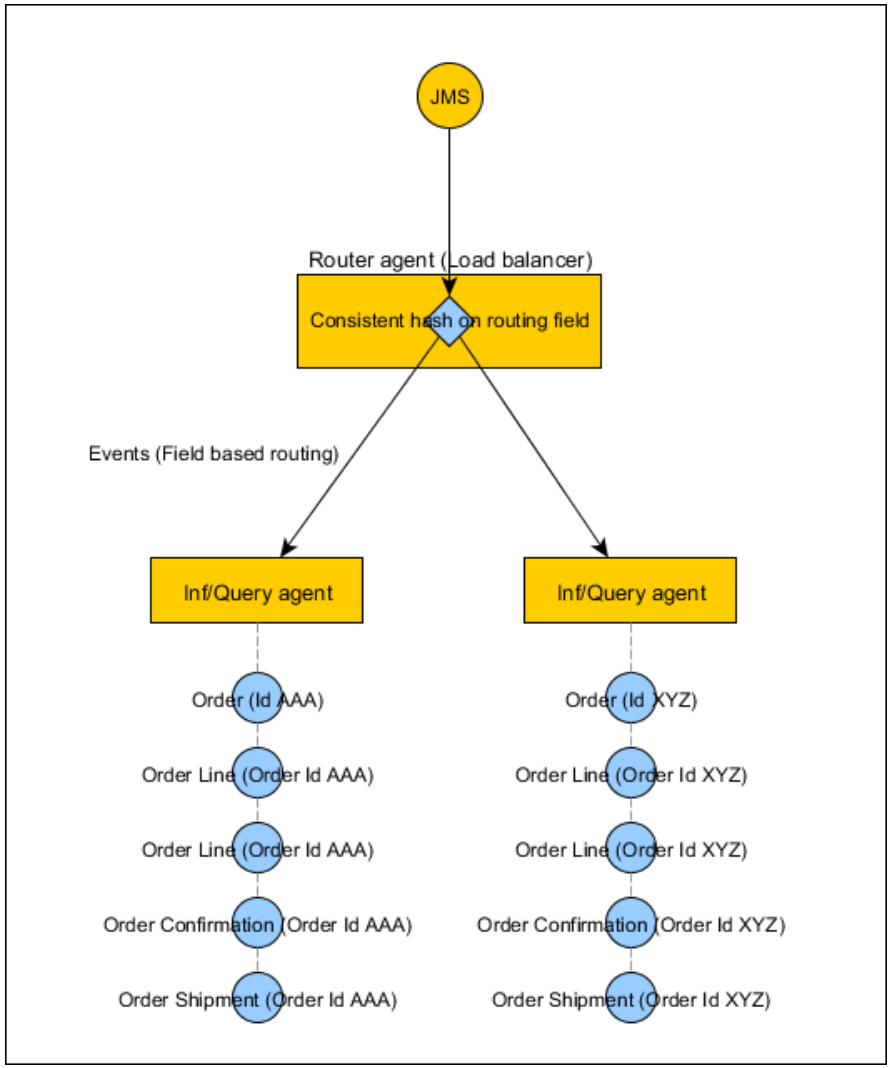
- BE 5.1 Standard
- 2 types
  - TIBCO EMS Channel based
  - Ad hoc with Local Channel

Content based load balancer

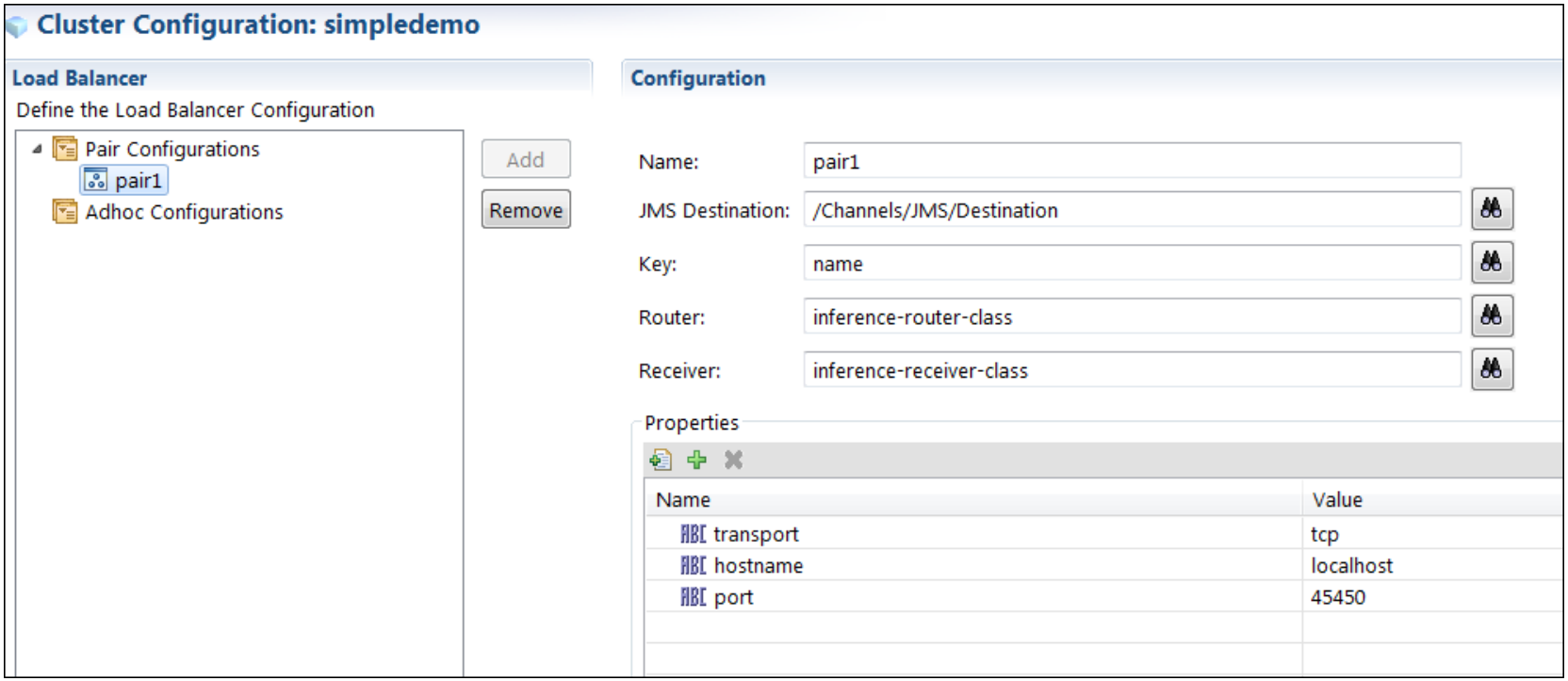
Pick a common routing field



Type 1 - EMS Channel load balancer



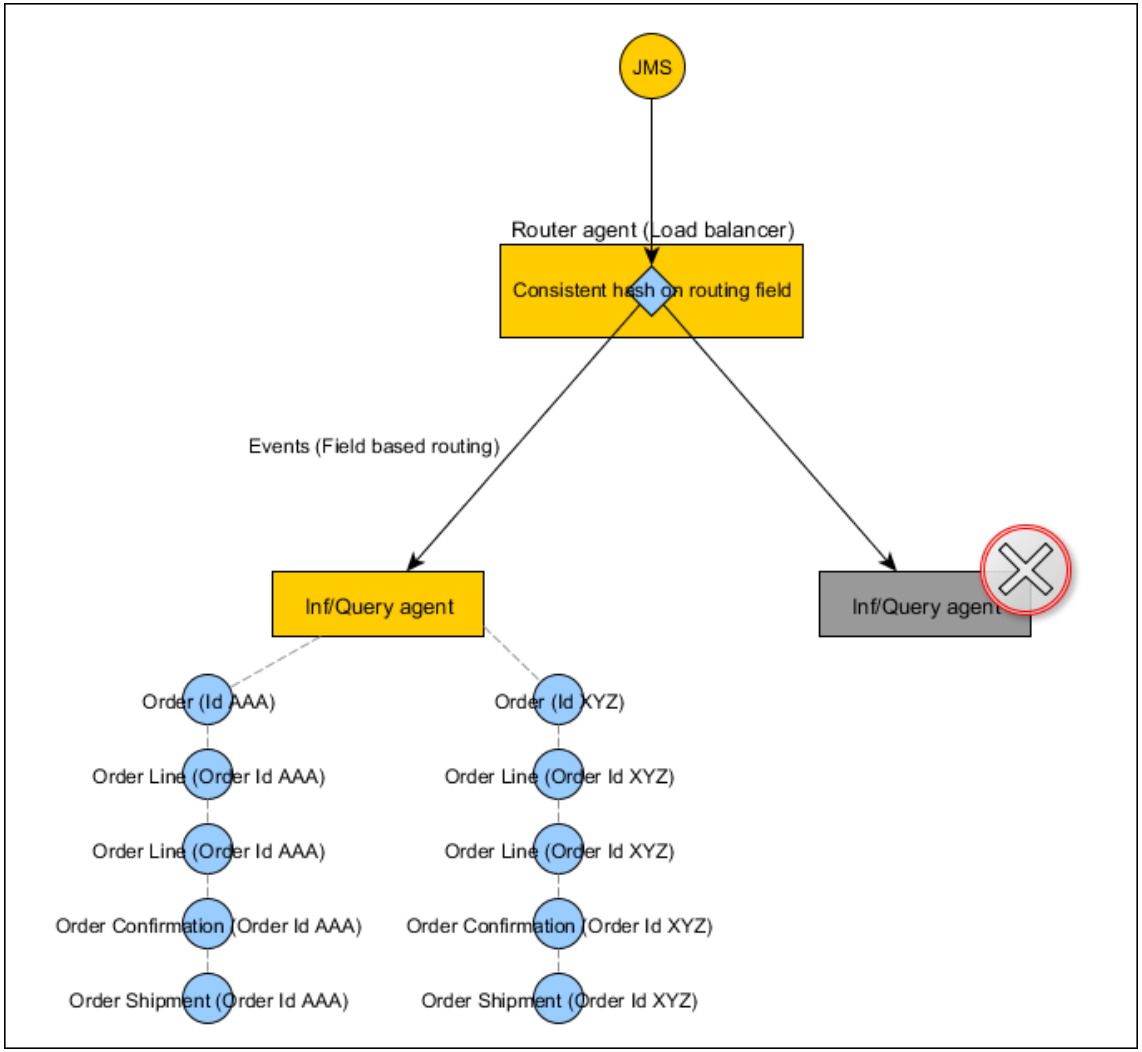
EMS Channel load balancer



EMS Channel load balancer

- All events with same routing field value
  - Routed to same receiver agent
- If receiver fails
  - New events go to another agent
- If new receiver starts
  - Evenly re-distributes key ownership

EMS Channel load balancer



EMS Channel load balancer

Image - Matthew Hartwick, from The Noun Project

EMS Channel load balancer

- Router agent
  - No rules
  - multiple copies for HA (like any other agent)
- Receiver agent
  - Normal agent (rules/queries)

- Same channel deployed on both agents
- Only router “really” connects to EMS
- Receiver’s EMS channel is a dummy
- Router talks to receiver over TCP
- Events “arrive” at receiver on dummy EMS channel

- Receiver can ack (EXPLICIT_CLIENT_ACK)
- Router relays ack (EMS ← Router ← Receiver)
- Router holds event in memory until receiver acks

EMS Channel load balancer

- Pros
  - Almost avoid global locks
  - No other receiver is working on “Order Id AAA”
  - Higher local cache hit ratio

- Cons
  - Extra send & ack hop
    - EMS → router → receiver
    - EMS ← router ← receiver

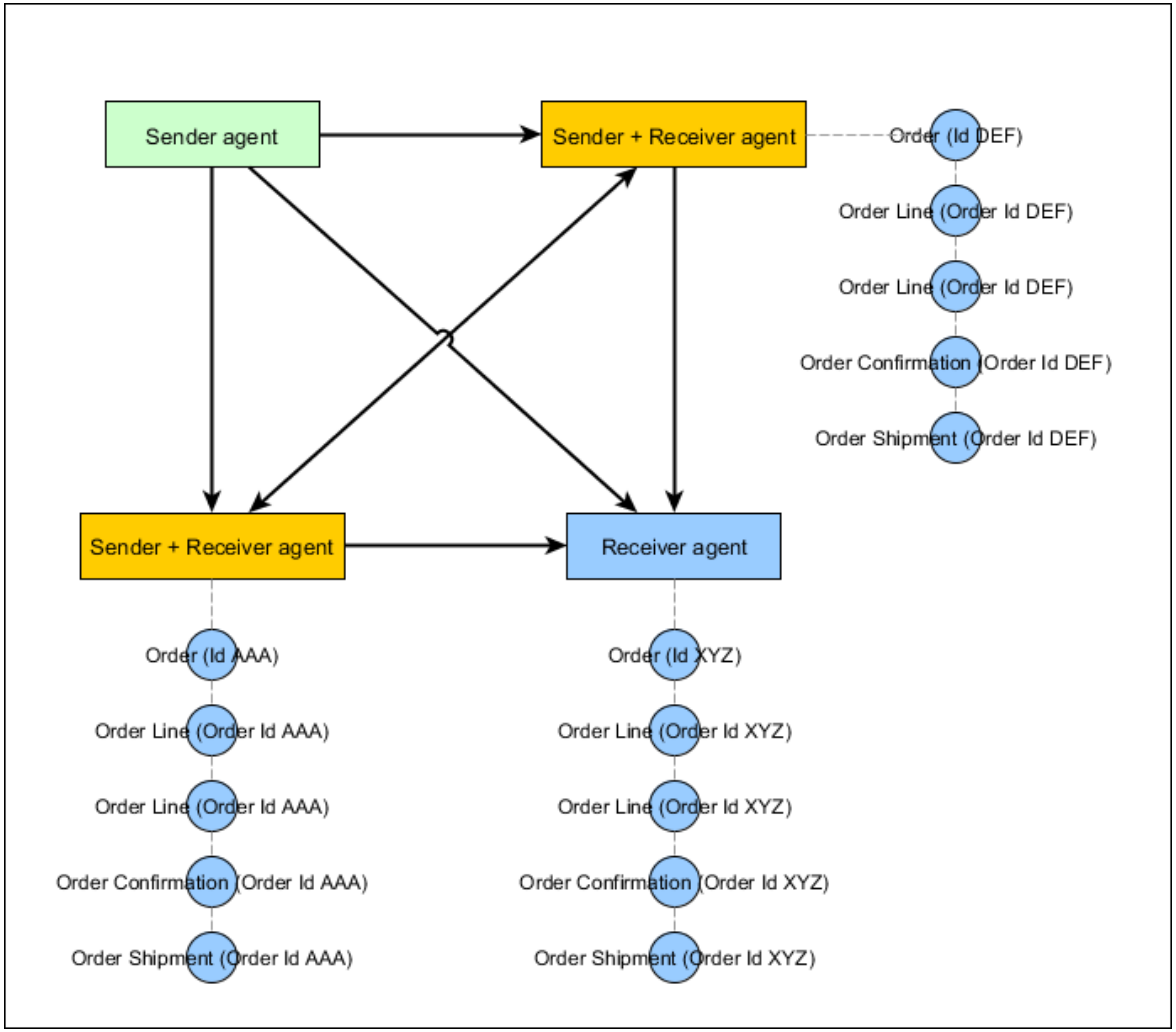
Type 2 - Ad hoc load balancer

- No separate router
- Any agent can send and/or receive

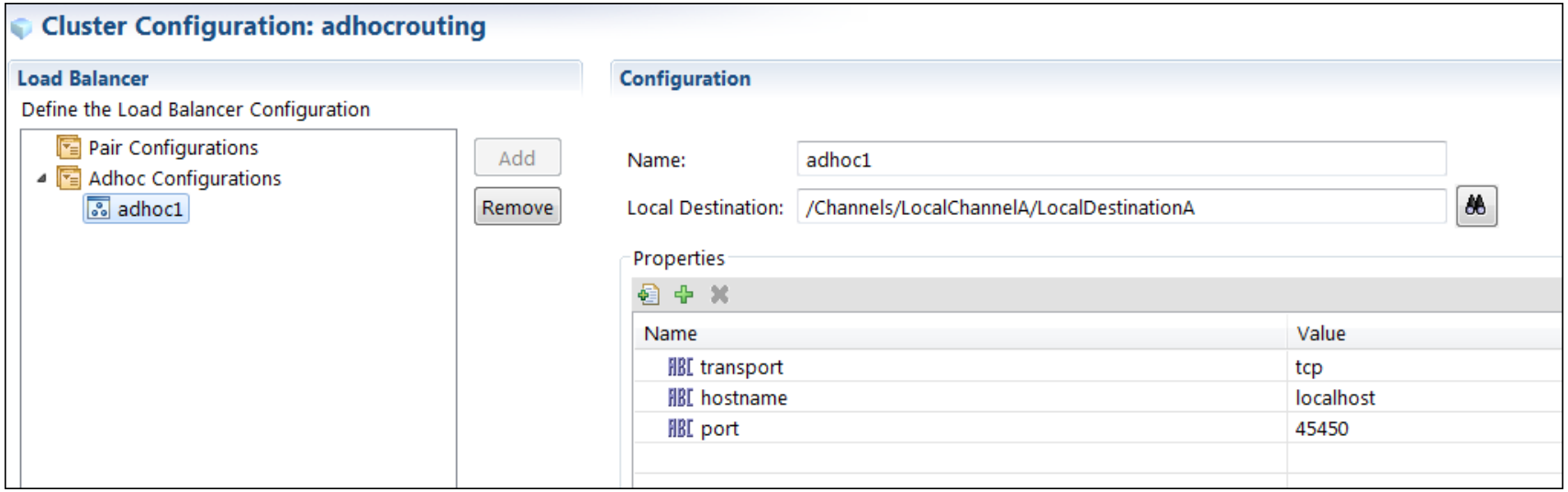
- Sender and receiver started with catalog fns.

- Receiving needs local channel
- Actual communication via TCP

Ad hoc load balancer



Ad hoc load balancer



Ad hoc load balancer

Startup.rulefunction

```
/**
 * @description
 */
void rulefunction Router.RuleFunctions.Startup {
    attribute {
        validity = ACTION;
    }
    scope {

    }
    body {
        Router.Entities.Counter c = Router.Entities.Counter.Counter("the-only-counte

        System.debugOut("Creating a load balancer to: adhoc1");
        Object loadBalancer = LoadBalancer.Router.createLoadBalancerTo("adhoc1");
        System.debugOut("Created a load balancer to: adhoc1");

        Util.HashMap.createMap("AllLoadBalancers");
        Util.HashMap.putObject("AllLoadBalancers", "demoLB", loadBalancer);
    }
}
```

Startup.rulefunction

```
/**
 * @description
 */
void rulefunction Receiver.RuleFunctions.Startup {
    attribute {
        validity = ACTION;
    }
    scope {

    }
    body {
        System.debugOut("Creating a load balanced receiver for: adhoc1");
        Object receiver = LoadBalancer.Receiver.createTcpReceiverFor("adhoc1");
        System.debugOut("Created a load balanced receiver for: adhoc1");

        Util.HashMap.createMap("AllReceivers");
        Util.HashMap.putObject("AllReceivers", "demoReceiver", receiver);
    }
}
```

Ad hoc load balancer

OnTimeEventA.rule

```
 * @author ajayapra-W510
 */
rule Router.Rules.OnTimeEventA {
    attribute {
        priority = 5;
        forwardChain = true;
    }
    declare {
        Router.Entities.TimeEventA a;
    }
    when {

    }
    then {
        String senderId = Cluster.getAgentName() + ":" + Cluster.getAgentId();
        String uniqueId = senderId + ":" + System.nanoTime();

        Router.Entities.Counter c = Instance.getByExtId("the-only-counter");

        String routingKey = "from-timer-at [" + senderId + "] " + c.testEventACtr;
        c.testEventACtr = (c.testEventACtr + 1) % 20;

        Entities.TestEventA tea = Entities.TestEventA.TestEventA(uniqueId, null, routi

        Object loadBalancer = Util.HashMap.getObject("AllLoadBalancers", "demoLB");

        System.debugOut("Sending event to remote receiver with name/routing key=" + te
        LoadBalancer.Router.send(loadBalancer, tea, tea.name);
        System.debugOut("Sent event to remote receiver");
    }
}
```

OnTestEventA.rule

```
/**
 * @description
 * @author ajayapra-W510
 */
rule Receiver.Rules.OnTestEventA {
    attribute {
        priority = 5;
        forwardChain = true;
    }
    declare {
        Entities.TestEventA t;
    }
    when {

    }
    then {
        System.debugOut("Processing event with name: " + t.name + " in rule");

        Event.consumeEvent(t);
    }
}
```

Ad hoc load balancer

Image - Matthew Hartwick, from The Noun Project

Ad hoc load balancer

- No ack relays to sender
- “Send and forget”
- Suitable for less frequent, less imp events
- Receivers can “unsubscribe” any time

Load balancer summary

- Reduce random event distribution
  - Cache thrashing
  - Global locks

- Use content based stickiness (Data locality)
  - Use local locks
  - Maximize local cache
  - Profit !

Q & A