APNIC

Network Automation and Orchestration

Network Automation and Orchestration

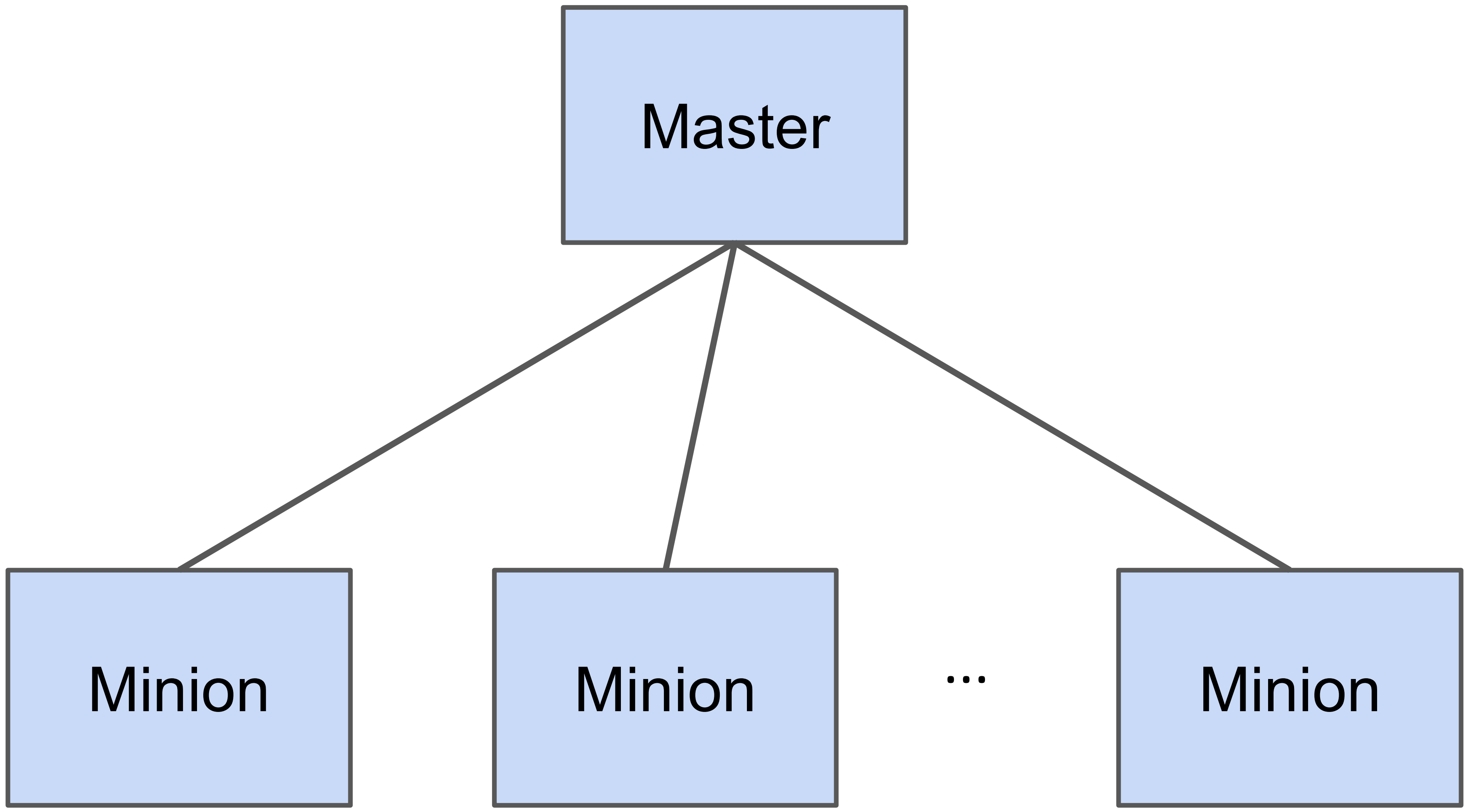
Module 2: Introduction to Salt

Salt

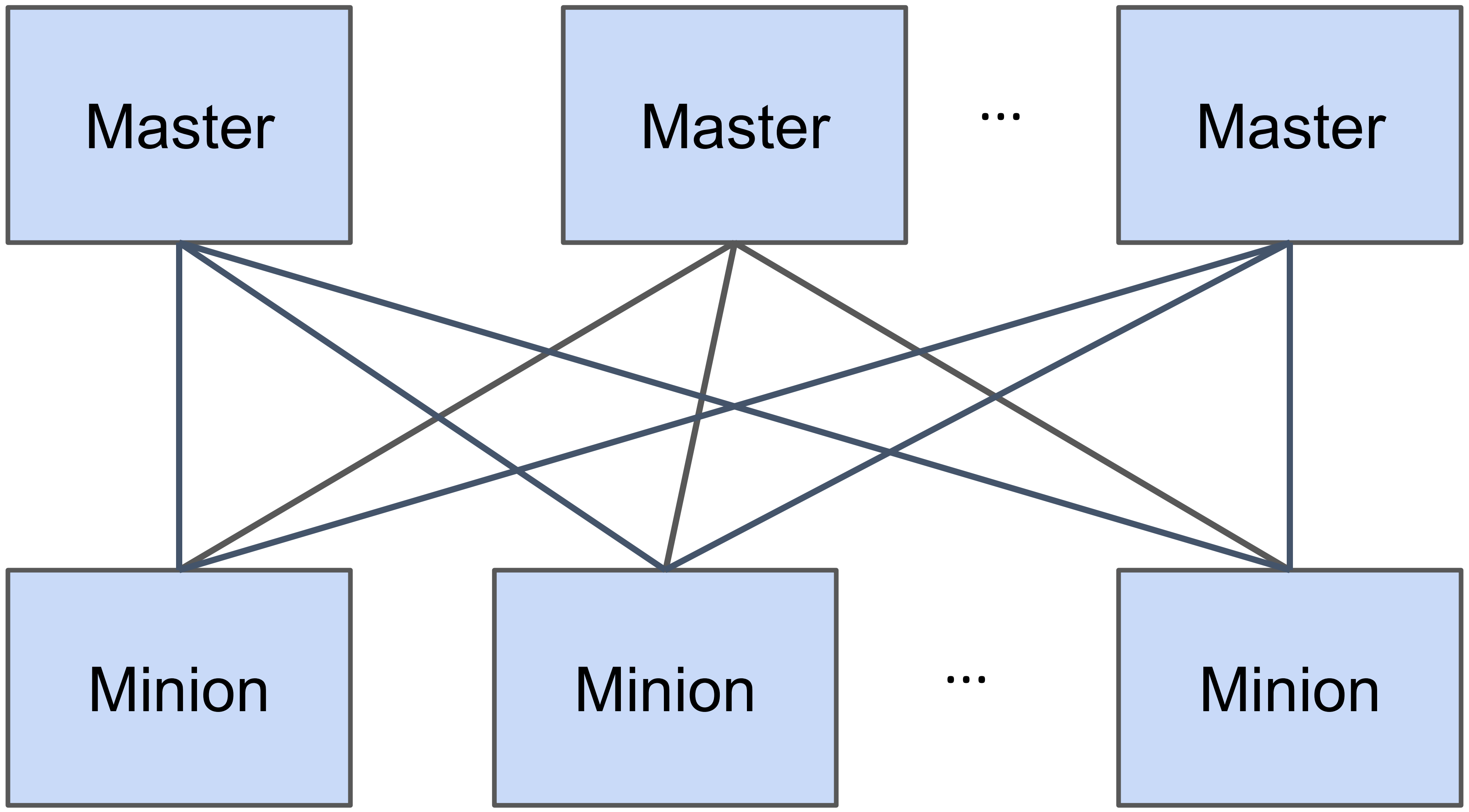
"In Salt, speed isn't a byproduct, it is a design goal. Salt was created as an extremely fast, lightweight communication bus to provide the foundation for a remote execution engine. Salt now provides orchestration, configuration management, event reactors, cloud provisioning, and more, all built around the Salt high-speed communication bus."

https://docs.saltstack.com/en/getstarted/speed.html

Salt Architecture: typical hub and spoke



Salt Architecture: multi-master



Salt Architecture

Minion: process or system service managing the machine it is installed on. Has the advantage to be executing everything locally: it receives command requests, executes locally, then returns the result.

Master: process or system service running on a dedicated machine, managing Minions. In typical large setups, one Master can manage around 35000-40000 Minions. This number is usually hardware limited.

Salt Architecture

The Minion-Master communication is established over two encrypted ZeroMQ ports (default 4505 and 4506), using a public-private key pair. The Minions listens to command requests over one port (4505) and sends executions results over 4506.

When the Minion starts up, it contacts the Master and provides it with its public key. If the Master recognizes the key signature, it accepts the Minion, otherwise rejects the affiliation.

Once the key has been accepted, the Minion can receive requests.

Salt Naming Conventions

Opts: the configuration options provided by the user in the Master or Minion configuration files. Usually, these refer strictly to the Master / Minion services themselves. Example: `ipv6: true` (whether should connect to the Master over IPv6).

Pillar: data introduced by the user, which is used to model automation logic, or it represents the data itself. Example: interfaces and their IP addresses, list of prefixes in a prefix-list, etc.

Salt Naming Conventions (2)

Grains: data collected dynamically by Salt. They represent attributes of the device being managed. Examples: model, vendor, serial number, CPU model and architecture, operating system name, etc.

In general, you don't need to do anything about it, but just be aware of their existence.

Grains can be used to model automation logic based on various attributes (e.g., deploy specific configuration only on a certain platform), or targeting groups of devices having the same characteristics (e.g., execute a command on Junos MX960 routers)

Salt Naming Conventions (3)

<u>Grains</u>: can also be provided statically or from external sources, as some attributes can’t be necessarily retrieved from the device. Example: device role, location (site, city, country, region), etc.

In short:

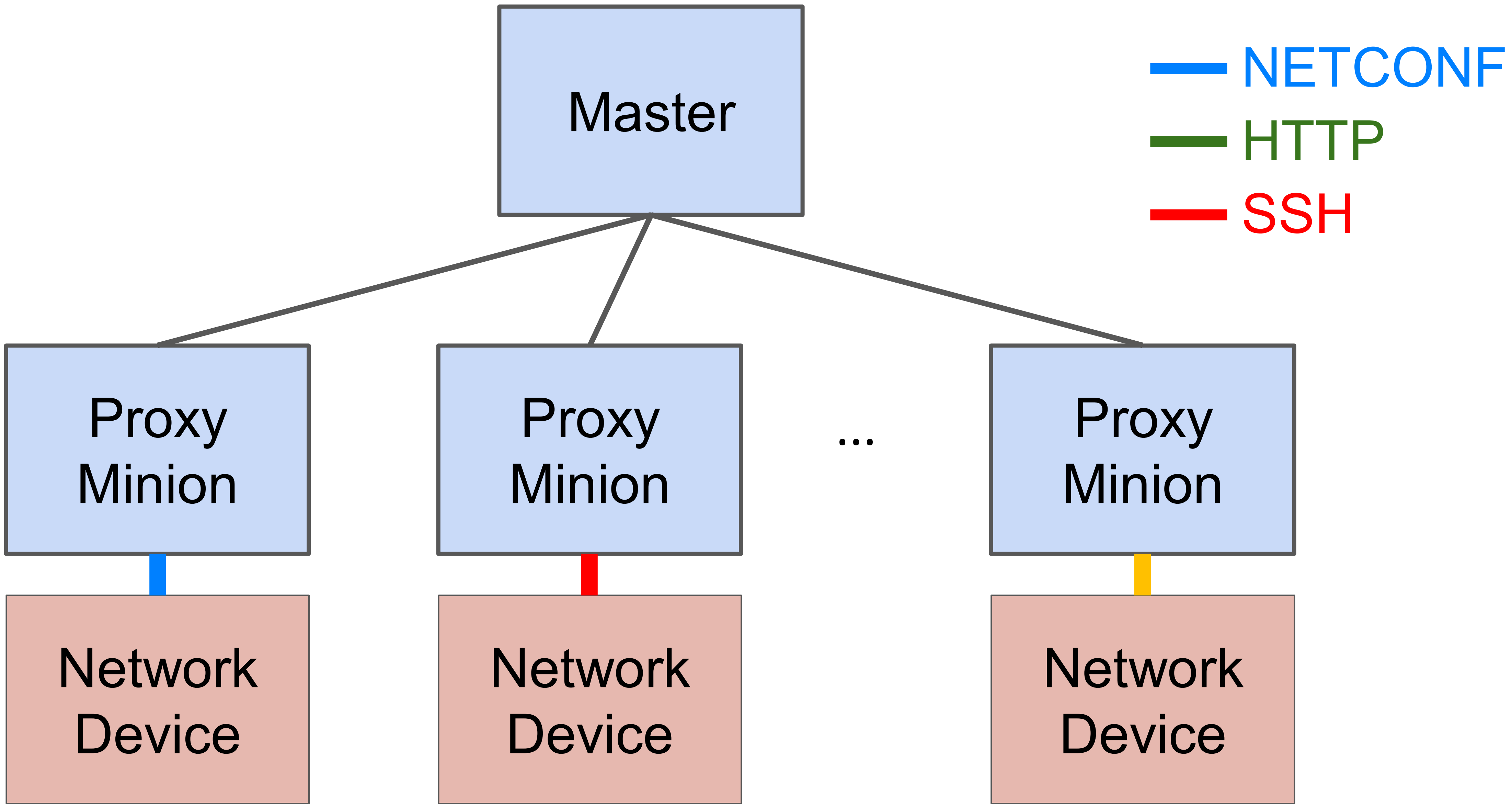
- Pillar = automation data
- Grains = attributes

Salt in networking

A Salt Minion implies a service running on the machine / system we want to manage. On typical network gear however, we can’t install custom software.

In 2015, SaltStack has introduced the *Proxy* Minion, which behaves just like the regular Minion, but doesn’t need to be running on the target device - instead it connects to the device over the channel or API of choice: HTTP, NETCONF, gRPC, SSH, etc.

Salt in networking (2)



Salt in networking (3)

Proxy Minions are simple processes able to run *anywhere*, as long as both conditions below are true:

1. Can connect to the Master.
2. Can connect to the network device (via the channel / API of choice - SSH / NETCONF / HTTP / gRPC, etc.)

For every network device, we have a Proxy Minion.

Salt in networking (4)

Deployment examples include:

- Running Proxy Minions as system services:
  - On a single server
  - Distributed over multiple servers
- Docker containers
  - E.g., managed by Kubernetes
- Services running in a cloud
  - See, for example, salt-cloud