redhat.

Configuring Condor with Wallaby

William C. Benton and Robert H. Rati
Red Hat, Inc.

http://www.redhat.com/mrg
http://getwallaby.com

redhat.

Forecast

- Background
- Getting started with Wallaby
- Using the Wallaby tools
- Using the Wallaby API
- What's next?

Background

Wallaby is a service that manages semantically meaningful, versioned configurations for even the largest Condor pools.

Wallaby is a service that manages semantically meaningful, versioned configurations for even the largest Condor pools.

Wallaby is a service that manages semantically meaningful, versioned configurations for even the largest Condor pools.

redhat

Wallaby is a service that manages semantically meaningful, versioned configurations for even the largest Condor pools.

Wallaby is a service that manages semantically meaningful, versioned configurations for even the largest Condor pools.

```
MASTER_HA_LIST = $(MASTER_HA_LIST), SCHEDD
HA_LOCK_URL = file:$(SPOOL)
VALID_SPOOL_FILES = $(VALID_SPOOL_FILES), SCHEDD.lock
SCHEDD_NAME = schedhost
SCHEDD.QMF_STOREFILE = $(SPOOL)/.schedd_storefile
HA_LOCK_HOLD_TIME = 300
HA_POLL_PERIOD = 60
```

I'D LIKE A HIGH-AVAILABILITY SCHEDD....

Semantic configuration

- Parameters
- Features
- Groups
- Nodes
- Subsystems

Semantic configuration

- Parameters
- Features
- Groups
- Nodes
- Subsystems

type, documentation, conflict and dependency relationships

Semantic configuration

- Parameters
- Features
- Groups
- Nodes
- Subsystems

configuration params, inclusion, conflict and dependency relationships

Semantic configuration

- Parameters
- Features
- Groups
- Nodes
- Subsystems

group-specific installed features, configuration parameters

Semantic configuration

- Parameters
- Features
- Groups
- Nodes
- Subsystems

sequence of explicit group memberships, liveness metadata

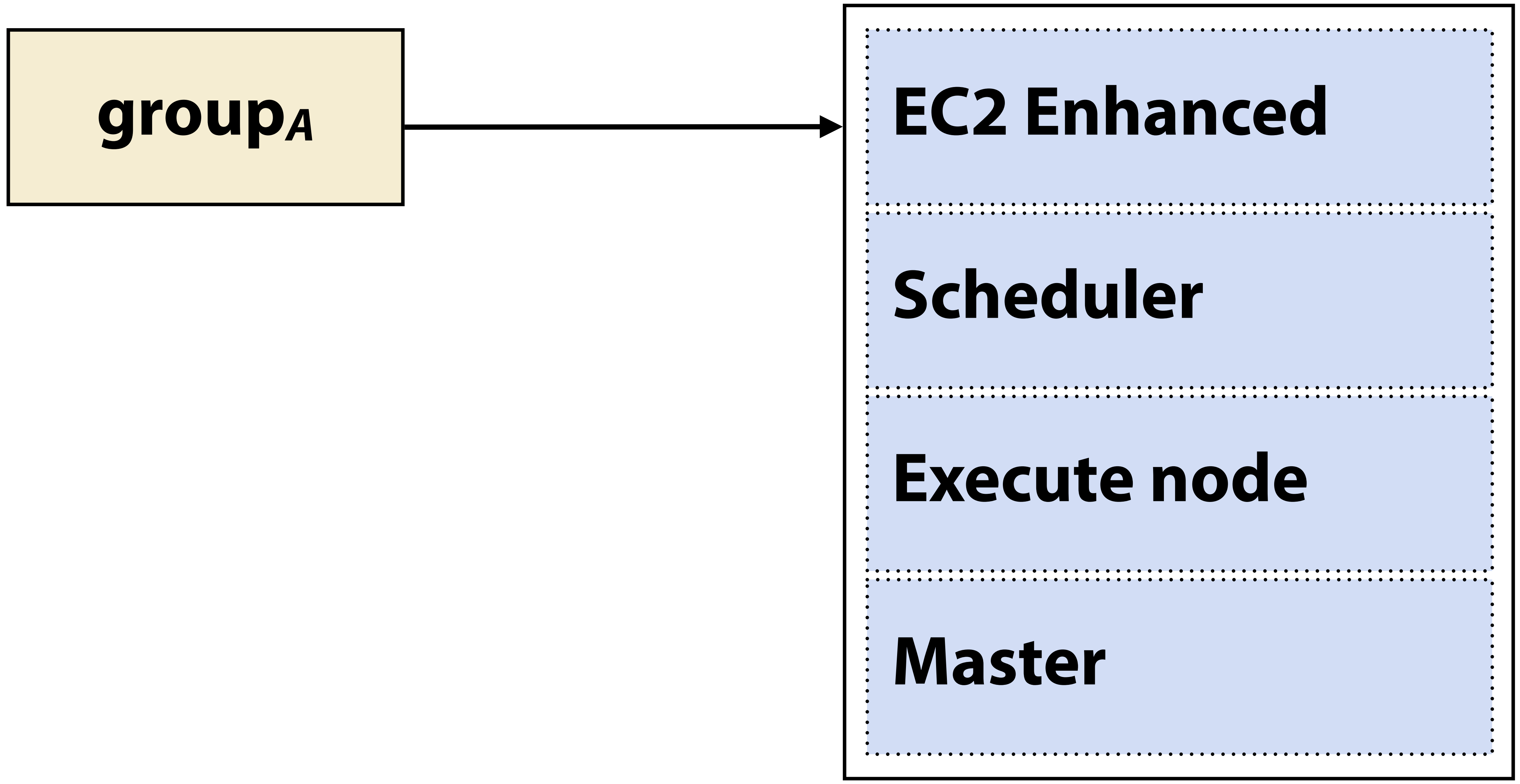
redhat

Semantic configuration

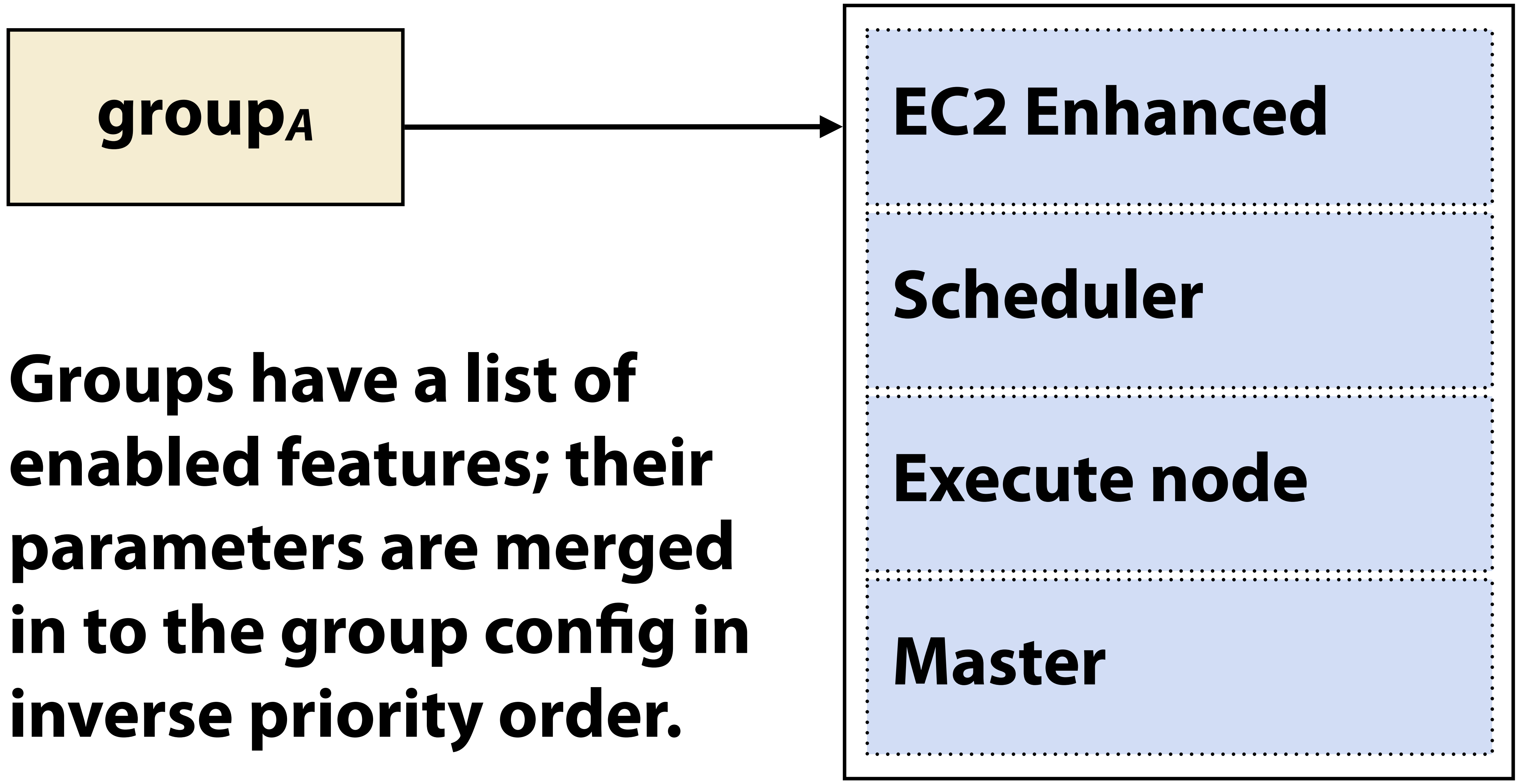
- Parameters
- Features
- Groups
- Nodes
- Subsystems

parameters of interest,
daemon process name

Configuring groups

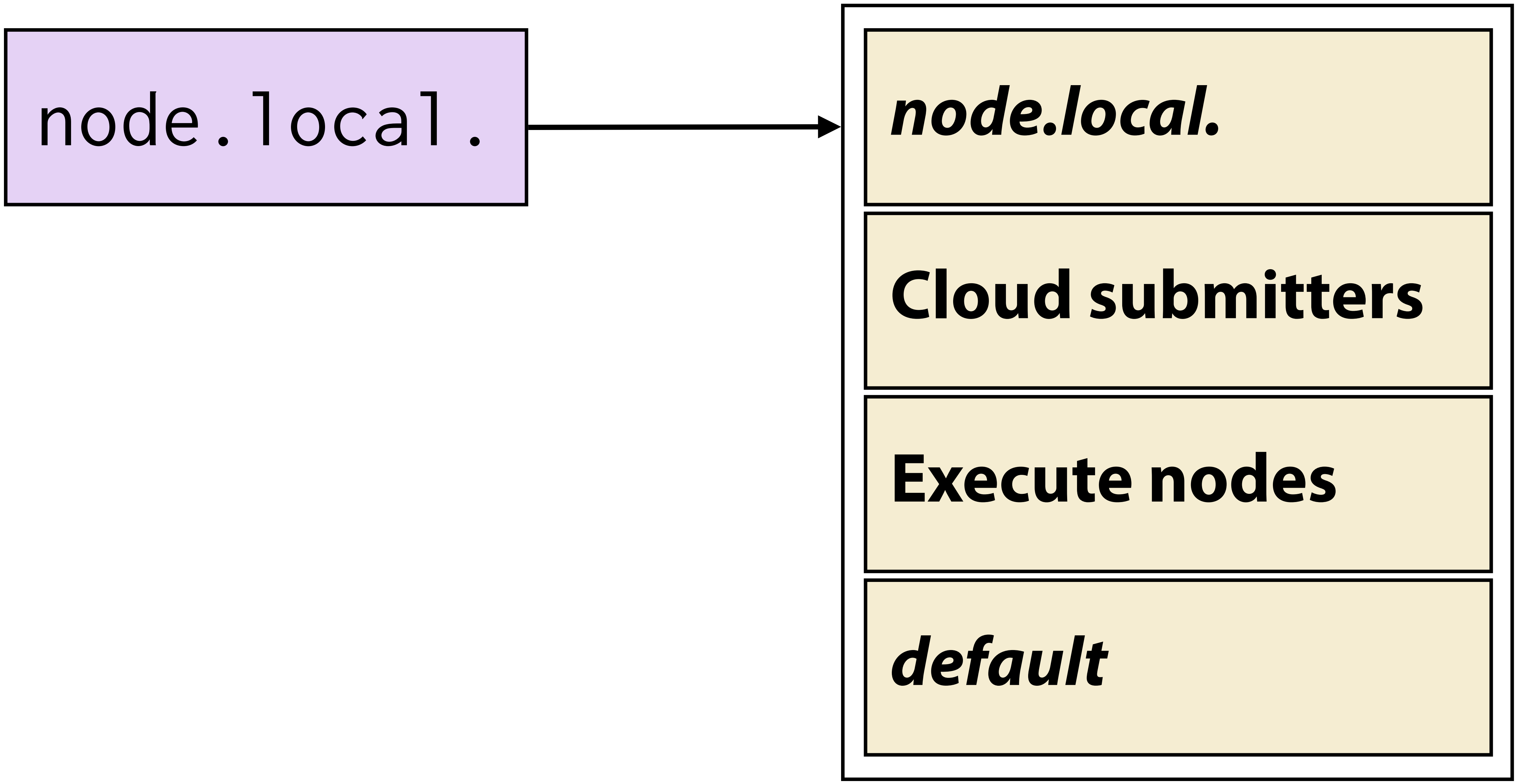


Configuring groups

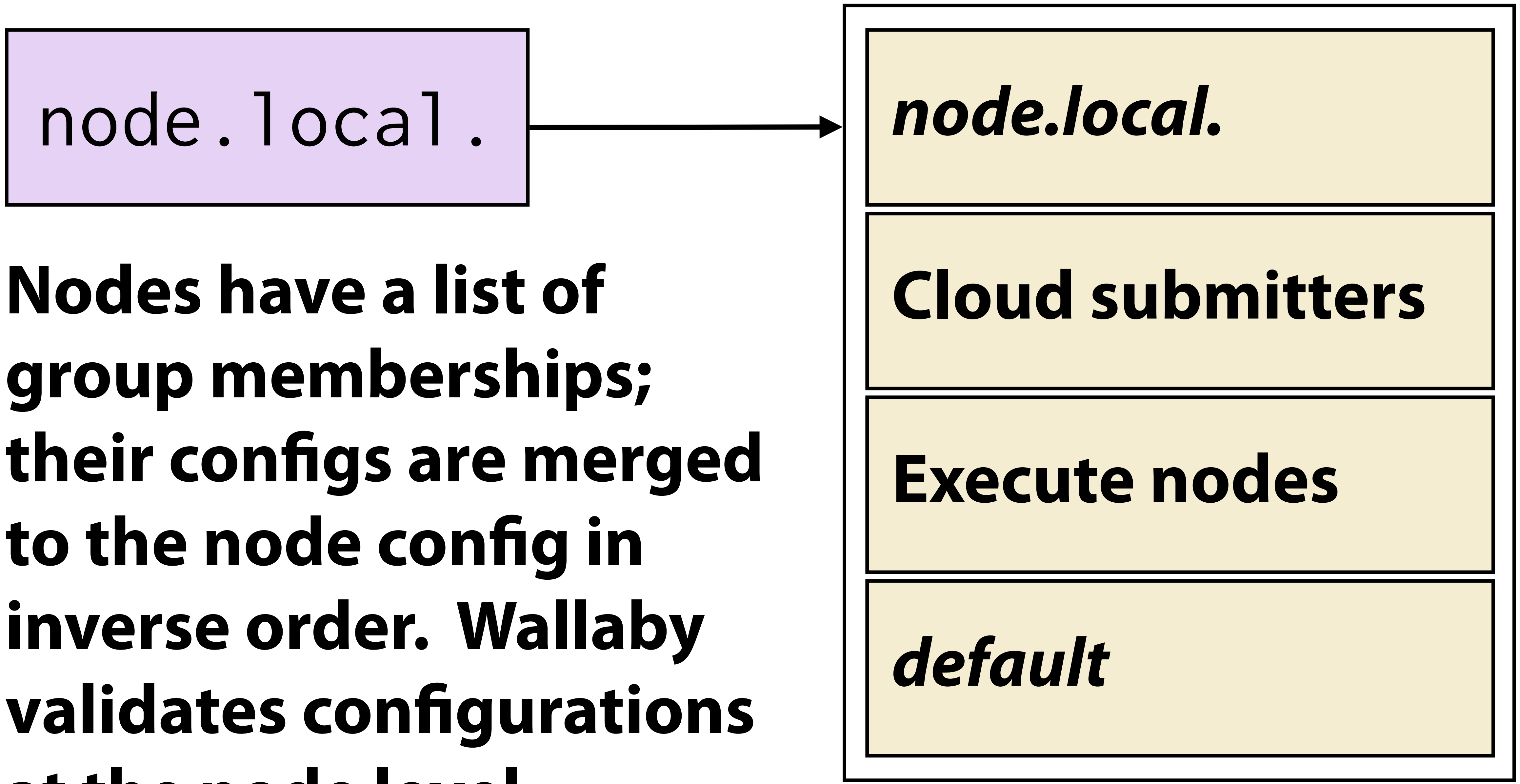


Groups have a list of enabled features; their parameters are merged in to the group config in inverse priority order.

Configuring nodes

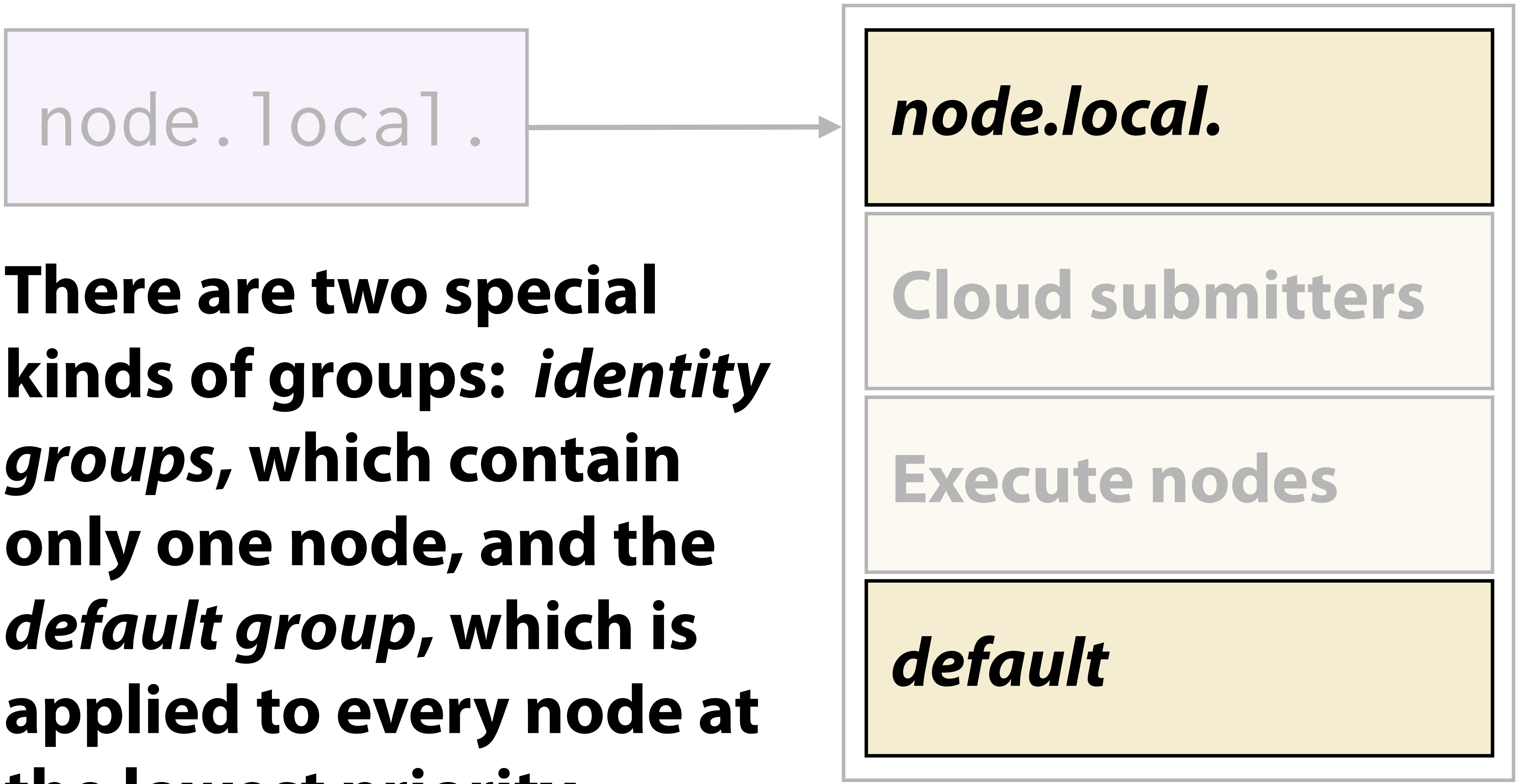


Configuring nodes



Nodes have a list of group memberships; their configs are merged to the node config in inverse order. Wallaby validates configurations at the node level.

Configuring nodes



There are two special kinds of groups: *identity groups*, which contain only one node, and the *default group*, which is applied to every node at the lowest priority.

Other details

- Import legacy Condor configurations
- Version control and differencing
- Guided and non-interactive tools
- Programmable API (access from Ruby, Python, or C++)
- HTTP REST interface

Other details

- Import legacy Condor configurations
- Version control and differencing
- Guided and non-interactive tools
- Programmable API (access from Ruby, Python, or C++)
- HTTP REST interface

redhat

Other details

- Import legacy Condor configurations
- Version control and differencing
- Guided and non-interactive tools
- Programmable API (access from Ruby, Python, or C++)
- HTTP REST interface

Guided command-line configuration

Live demo

Extending Wallaby

Basic concepts

- Wallaby entities
- Access the Wallaby service via QMF or via user-friendly client libraries
- Use the *Wallaby console* for quick shebang scripts or prototyping
- Extend Wallaby with *shell commands*

Using the Wallaby shell

- `wallaby help` **shows options**
- `wallaby help commands` **shows a list of available commands**
- **Try it out: get help on params with** `wallaby apropos -i start`
- **We can extend the shell by creating new commands – stay tuned!**

Exploring the Wallaby API

- API docs at http://getwallaby.com
- `wallaby console` gives you an interactive Ruby environment with a connection to your Wallaby agent
- The Wallaby agent is available as an object: `Wallaby::store`

Try it out!

```
% wallaby console
irb(main):001:0> Wallaby::store.features

=> [<Mrg::Grid::ConfigClient::Feature:
ExecuteNodeTriggerData>,
<Mrg::Grid::ConfigClient::Feature:
BaseJobExecuter>,
<Mrg::Grid::ConfigClient::Feature:
AviaryScheduler>, ... ]
```

Try it out!

```
irb(main):002:0> f = Wallaby::store.features[0]
irb(main):003:0> f.params

=> {"TRIGGER_DATA_GETDATA_PERIOD"=>"5m",
"STARTD_CRON_AUTOPUBLISH"=>"If_Changed",
"TRIGGER_DATA_JOBLIST"=>"GetData", ... }

irb(main):004:0> f.included_features

=> ["ExecuteNode"]
```

redhat

Try it out!

```
irb(main):005:0> Wallaby::store.addParam("FOO")
irb(main):006:0> f.modifyParams("ADD", {"FOO"=>
"BAR"}, {})
irb(main):007:0> f.update
irb(main):008:0> f.params["FOO"]

=> "BAR"
```

Extending the Wallaby shell

- The Wallaby shell provides a convenient environment for making commands that access Wallaby
- Simply create a class that interacts with Wallaby entities; don’t worry about application boilerplate

redhat.

Extending the Wallaby shell

Extending the Wallaby shell

```
% export WALLABY_COMMAND_DIR=${HOME}/.wallaby
% mkdir $WALLABY_COMMAND_DIR
% wallaby new-command -d "Lists nodes that \
  haven’t checked in for at least a week." -D \
  $WALLABY_COMMAND_DIR slacker-nodes
```

Extending the Wallaby shell

```
% export WALLABY_COMMAND_DIR=${HOME}/.wallaby
% mkdir $WALLABY_COMMAND_DIR
% wallaby new-command -d "Lists nodes that \
  haven’t checked in for at least a week." -D \
  $WALLABY_COMMAND_DIR slacker-nodes

% wallaby help commands | grep slacker
```

Extending the Wallaby shell

```
% export WALLABY_COMMAND_DIR=${HOME}/.wallaby
% mkdir $WALLABY_COMMAND_DIR
% wallaby new-command -d "Lists nodes that \
  haven’t checked in for at least a week." -D \
  $WALLABY_COMMAND_DIR slacker-nodes

% wallaby help commands | grep slacker

% cd $WALLABY_COMMAND_DIR
% $EDITOR cmd_slacker_nodes.rb
```

Interesting parts

```ruby
def self.opname
  "slacker-nodes"
end

def self.description
  "Lists nodes that haven't checked in " +
  "for at least a week."
end
```

Interesting parts

```ruby
def init_option_parser
  OptionParser.new do |opts|
    opts.banner = "Usage:  wallaby #{self.class.opname}\n" +
                  "#{self.class.description}"

    opts.on("-h", "--help", "displays this message") do
      puts @oparser
      exit
    end
  end
end
```

redhat

Interesting parts

```
def act




  return 0
end

private

def one_week_ago


end
```

Interesting parts

```
def act
  slackers = store.nodes.select
    {|n| n.last_checkin < one_week_ago}
  slackers.each {|s| puts s.name}

  return 0
end

private

def one_week_ago
  tm ||= Time.now.utc - (60 * 60 * 24 * 7)
  (tm.tv_sec * 1000000) + tm.tv_usec
end
```

What now?

redhat

Find some Wallaby API apps

- `wallaby http-server` – **download a node's configuration over HTTP**
- `wallaby feature-import` – **migrate old configuration file snippets**
- **Albatross (Erik Erlandson) – automatic testing of pool functionality, scale, throughput**

redhat

Keep up with development

- We're working on a full REST API, more templating features, more sophisticated versioning, and more!
- Visit us at http://getwallaby.com – we'd love to hear how *you're* using and extending Wallaby