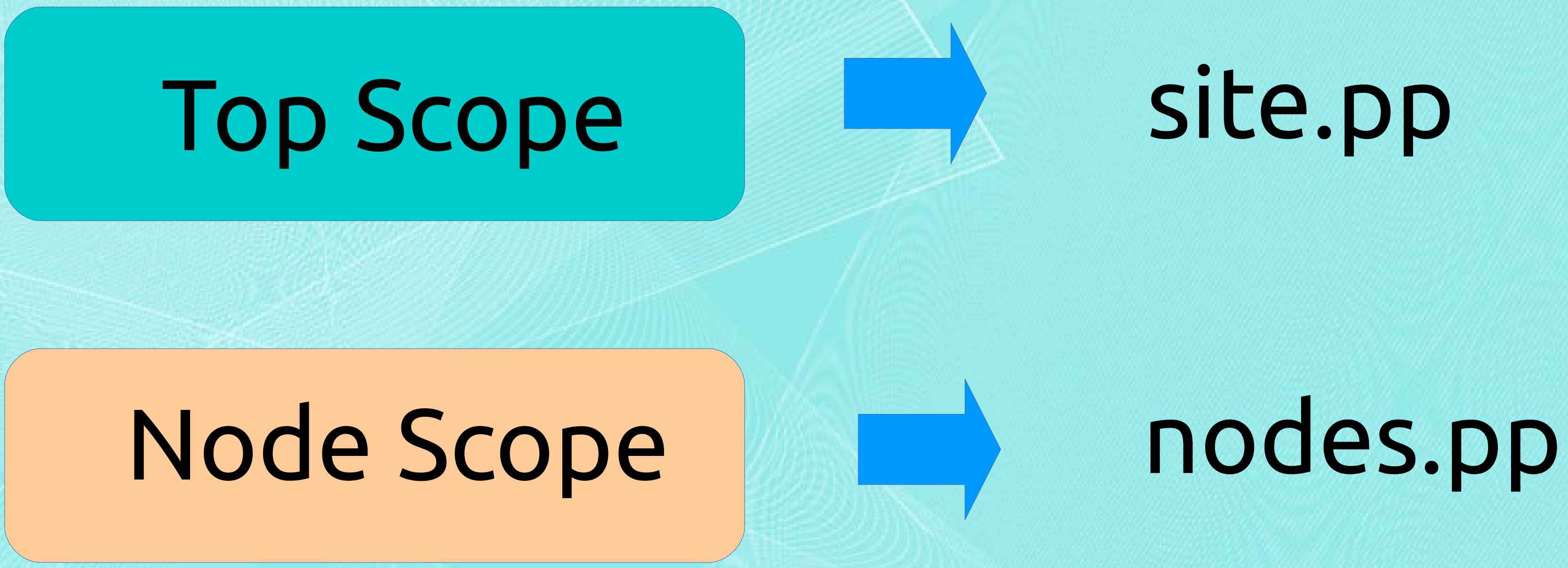
Variables and Facts

Scopes

A scope is a specific area of code, which is partially isolated from other areas of code. Scopes limit the reach of:

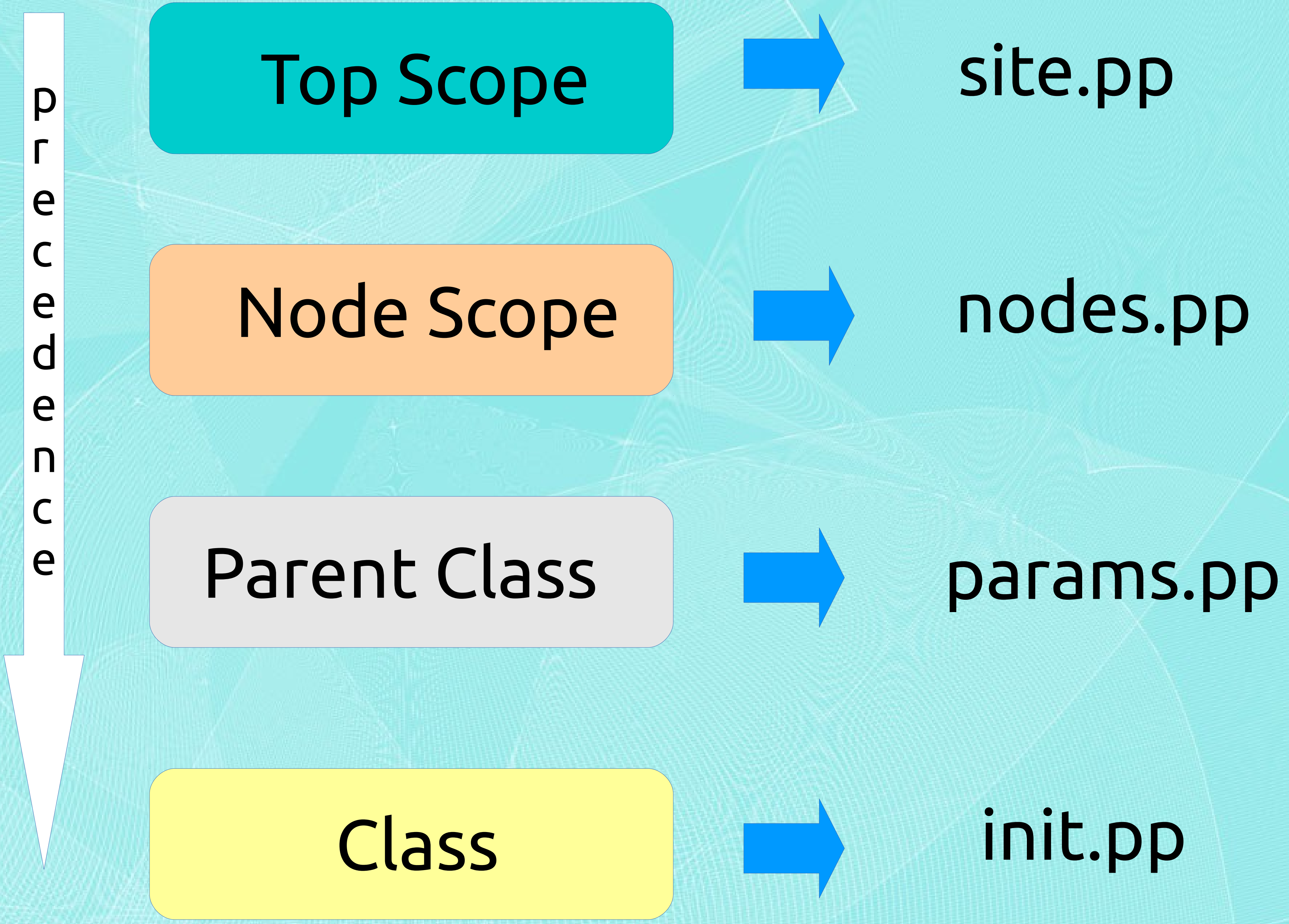
- Variables
- Resource defaults

$$\begin{tikzcd}\boxed{\text{Top Scope}} \arrow[r] & \text{site.pp}\end{tikzcd}$$


Top Scope
site.pp
Node Scope
nodes.pp

Top Scope	➡	site.pp
Node Scope	➡	nodes.pp
Parent Class	➡	params.pp

Top Scope	→	site.pp
Node Scope	→	nodes.pp
Parent Class	→	params.pp
Class	→	init.pp


precedence
Top Scope
site.pp
Node Scope
nodes.pp
Parent Class
params.pp
Class
init.pp

variable

```
$content = "some content\n"
```

Assigning Variables

- $variables start with a dollar sign. You assign to variables with the = operator.
- can hold strings, numbers, booleans, arrays, hashes, and the special undef value
- Variables can be assigned once in each scope
- Variables can be used even without assigning a value, in which case its set to undef

Using Variables

Variables are used to set

- Resource Titles
- Attribute Values

```
$content = "hello bollywood"

file{“${bollywood_conf}”:
       ensure    => present,
       mode      => 0644,
       content   => “${content}”,
}
```

Best Practices

```
${var}
```

Its a good practice to surround variable with curly braces.

MUST

“${var}”

It's essential to enclose variables in double quotes in order for it to be interpolated.

Variable Names

$var1

$$\uparrow$$

Short
name

params.pp

Variable Names

$$\begin{tikzcd} \$var1 \arrow[r] & \$nginx::params::var1 \\ \text{Short name} \arrow[u, dashed] & \text{Qualified name} \arrow[u, dashed] \end{tikzcd}$$

params.pp

Constants

- Variables can be assigned only once per scope
- Behave more like constants

Automatic variables : ipaddress, hostname, osfamily, is_virtual, operatingsystem, architecture, fqdn

Automatic variables == Facts

Facts

```
file {“${index_file}”:
      ensure  => file,
      mode    => 0644,
      content => "This is ${hostname} : ${osfamily}\n",
}
```

Facts

Facts are top scope variables

e.g. $::ipaddress

Finding Facts

```
$  facter | less
$ facter osfamily
```

Conditionals

Conditionals

Puppet has almost complete set of conditionals. We are going to overview the most common ones.

If - else

This is the most used and simplest of all the conditional statements available.

It's syntax is pretty much straight forward like most programming languages:

```
if condition {
  block of code
}
elsif condition {
  block of code
}
else {
  block of code
}
```

```
if $operatingsystem='centos' {
   package {“httpd”:
      ensure => installed
   }
}
elsif $operatingsystem='ubuntu' {
   package {“apache2”:
      ensure => installed
   }
}
```

Unless

- Revered if statements
- Block is applied only when the condition is false
- Can not use else or elsif with unless

```
unless EXPRESSION {
        OPTIONAL_SOMETHING
}
```

Case

Case

Widely used to choose values of variables based on osfamily.

e.g.:

```
case $osfamily: {
    RedHat: { $apache = 'httpd' }
    Debian: { $apache = 'apache2' }
    default: { fail("Not Supported") }
}

package {'apache':
    name => $apache,
    ensure => installed
}
```

Good Practice

default{}

Its a good practice to have a default case. In case a case must match to something, default can be used with a fail function to error out and exit.

Selectors

Select value from multiple options, based on a condition

```
$apache = $operatingsystem ? {
      centos          => 'httpd',
      ubuntu          => "apache2",
      default         => undef,
}
```

Case vs Selector

```
case $::osfamily {
     'Debian': {
       $pkg = 'apache2'
     }
     'RedHat': {
       $pkg = 'httpd'
} 'Darwin': {
       $pkg = 'apache'
     }
     default: {
       $pkg = 'UNKNOWN'
} }
```

```
$pkg = $::osfamily ? {
     'Debian' => 'apache2',
     'RedHat' => 'httpd',
     'Darwin' => 'apache',
     default  => 'UNKNOWN',
  }
```

Expressions

- Comparison Operators
- Boolean
- Arithmatic Operators