The Myths*, Musts* and Migraines* of Migrations

Drupal Dev Days Lisbon
5 July 2018

Marc van Gend

@marcvangend

But enough about me –

Let's talk about you!

* Developers?
* Project managers?
* Site owners ("the client")?
* None of the above?

Quiz

The key to a successful migration is...

A. Good clean source data
B. A committed, cooperating client
C. Plenty of development time

You were right! You were right!

Everybody was right!

Non-Drupal to Drupal migrations

* Source - Process - Destination
* Setup
* My First Migration™
* Source data
* Process plugins
* Project context
* Tips

*Myth
Migrations are Complicated

Source.

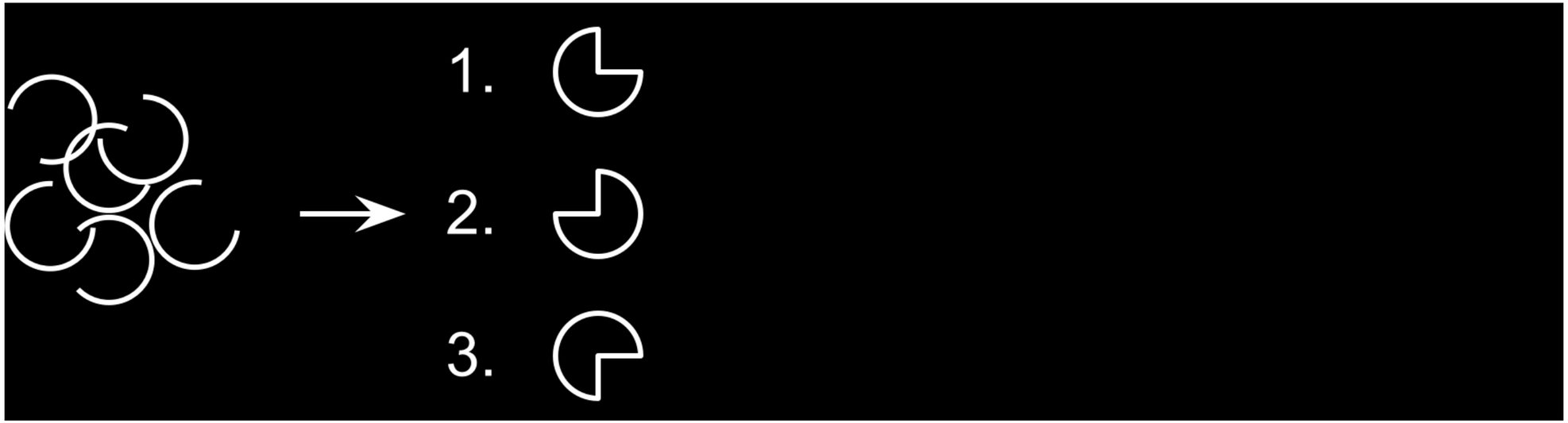
Process.

Destination.

Migrations are straight-forward.
Literally.

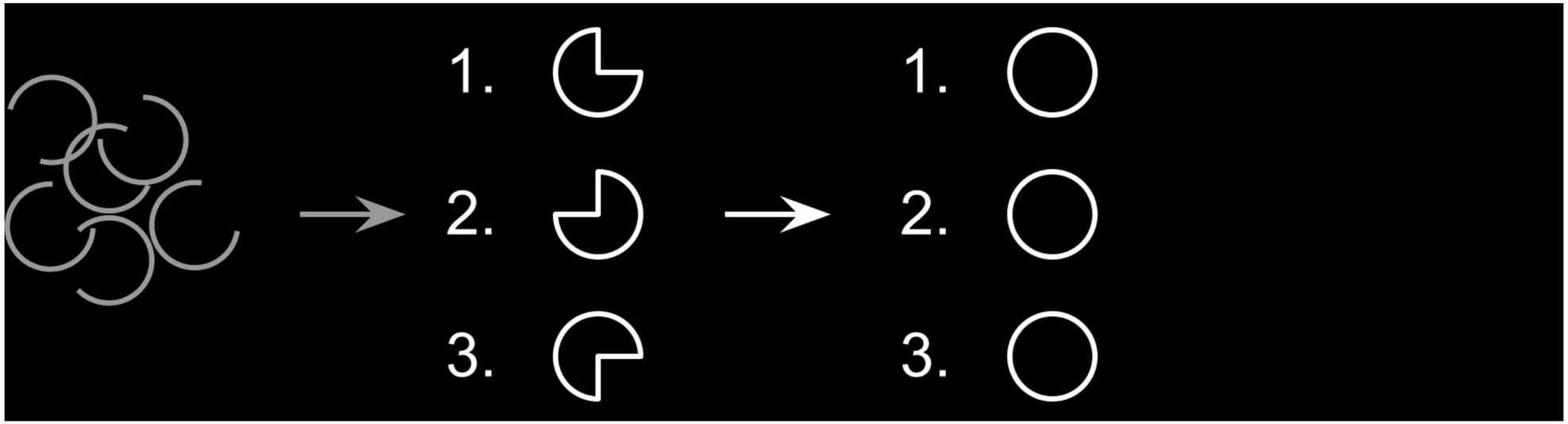
Source

* **Database**, File, URL...
* 1 result per migration item
* Defines unique Source ID



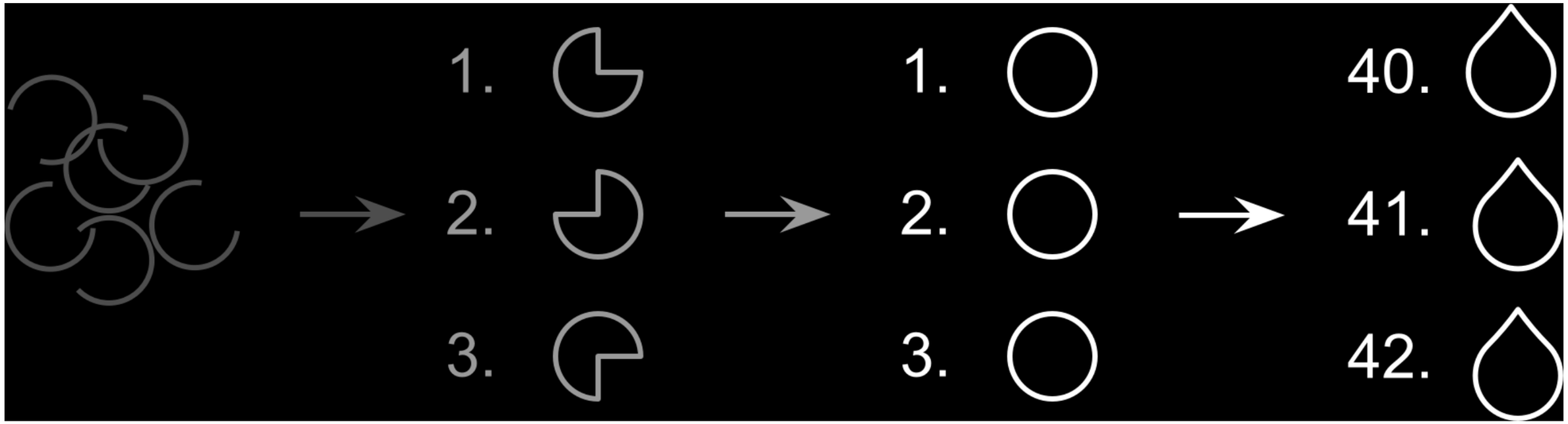
Process

- Maps source values to entity fields / properties
- Transforms data (uppercase, migration lookup, custom...)
- Chainable



Destination

- Node, Term, User, Redirect, Setting, custom...
- Writes collected data to Drupal DB
- Can support rollback
- Maps Source ID -> Destination ID



*Must

The right tools

* Modules
* Tools
* Custom module
* Test site

Modules

Migrate (core)

Migrate API and base functionality
Continuous migrations with ID mapping

Migrate Plus

Define migrations as config entities
Additional plugins and enhancements
Data parsers and authentication
Examples

Migrate Tools

Drush commands and UI

Tools

Drush

Import migration config
Run migrations
Rollback, reset, status, etc.

Drupal Console

Create plugins

Database manager

PhpStorm, phpMyAdmin

Custom module

```
migrate_demo
├── migrate_demo.info.yml
├── config
│   └── install
│       ├── migrate_plus.migration.demo_node_article.yml
│       └── migrate_plus.migration_group.demo.yml
└── src
    └── Plugin
        └── migrate
            ├── process
            │   └── TitleCleanup.php
            └── source
                └── DemoNodeArticle.php
```

My First Migration™

* Migrate Group
* Migration
* Go!

*Must Handwritten YAML

```yaml
# Enough about you, let's talk about me!
name: Marc
favorites:
  music: dEUS
  beer:
    - IPA
    - Triple
colleagues:
  -
    name: Dirk
    role: Support engineer
  -
    name: Joyce
    role: Drupal developer
```

Migration groups

* Groups migrations (duh!)
* Import all migrations in group
* Shared configuration

Home » Administration » Structure

MIGRATION GROUP	MACHINE NAME	DESCRIPTION	SOURCE TYPE	OPERATIONS
Demo Imports	demo	A few demo imports, to demonstrate how to implement migrations.	SQL database	List migrations
Some other group	some_other_group	Just for fun	dev/null	List migrations

Migration group config

migrate_demo/config/install/migrate_plus.migration_group.demo.yml

```
id: demo
label: Demo Imports
description: A few demo imports, to demonstrate how to implement migrations.
source_type: SQL database
shared_configuration:
  source:
    key: migrate
dependencies:
  enforced:
    module:
      - migrate_demo
```

Migration source: DB settings

settings.php

```
$databases['migrate']['default'] = array (
  'database' => 'migrate_demo_source',
  'username' => 'migrate_demo_source',
  'password' => 'Secret!',
  'prefix' => '',
  'host' => '127.0.0.1',
  'port' => '3306',
  'namespace' => 'Drupal\\Core\\Database\\Driver\\mysql',
  'driver' => 'mysql',
);
```

Migration source: creating a source plugin

PHP:

```
class MySourcePlugin extends SourcePluginBase
```

Drupal Console:

```
drupal generate:plugin:migrate:source {}
```

Drush 9:

```
(nope)
```

Migration source: source plugin

migrate_demo/src/Plugin/migrate/source/DemoNodeArtist.php

```
/**
 * Source plugin for artist content.
 *
 * @MigrateSource(
 *   id = "demo_node_artist"
 * )
 */
class DemoNodeArtist extends SqlBase {
```

migrate_demo/src/Plugin/migrate/source/DemoNodeArtist.php

```
  /**
   * {@inheritdoc}
   */
  public function query() {
    $query = $this->select('Artist', 'a')
      ->fields('a', [
        'ArtistId',
        'Name',
      ]);
    return $query;
  }
```

migrate_demo/src/Plugin/migrate/source/DemoNodeArtist.php

```
  /**
   * {@inheritdoc}
   */
  public function fields() {
    $fields = [
      'ArtistId' => $this->t('Artist ID'),
      'Name' => $this->t('Artist name'),
    ];

    return $fields;
  }
```

migrate_demo/src/Plugin/migrate/source/DemoNodeArtist.php

Migration config

migrate_demo/config/install/migrate_plus.migration.demo_node_artist.yml

```yaml
id: demo_node_artist          # Migration ID
label: Artists                # Human name
migration_group: demo
source:
  plugin: demo_node_artist    # Data source
process:
  title: Name                 # Use the 'Name' value as title
  type:
    plugin: default_value
    default_value: artist     # Node type is 'artist'
destination:
  plugin: entity:node         # Save as a node
```

*Must

Importing your config

```
$ drush config-import \
  --partial \
  --source=modules/custom/migrate_demo/config/install
```

Go!

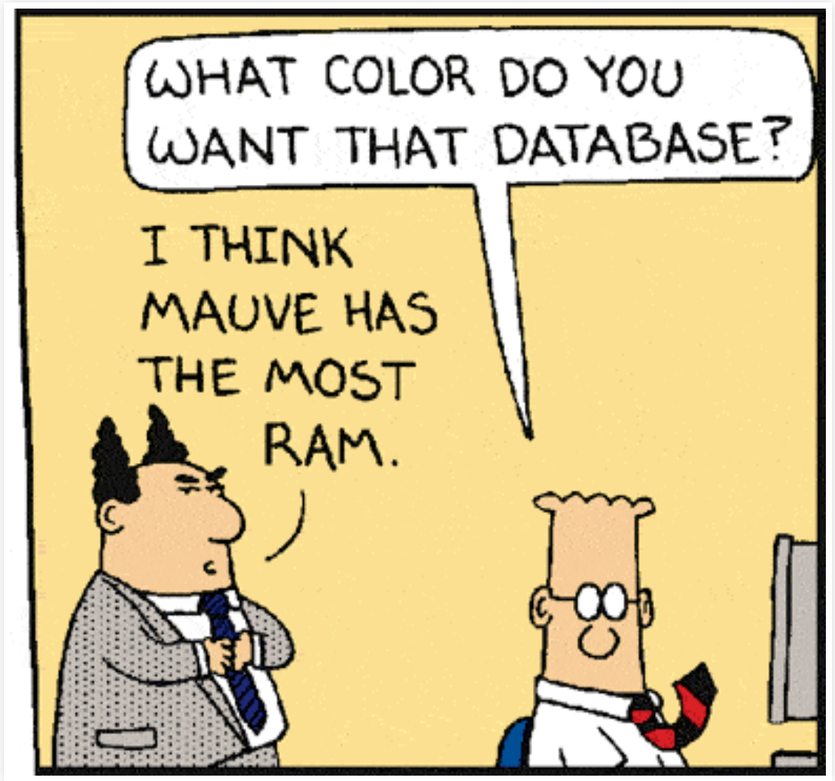
```
$ drush migrate-import demo_node_artist
```

Or use the UI:

Home » Administration » Structure » Migrations » Edit migration group

MIGRATION	MACHINE NAME	STATUS	TOTAL	IMPORTED	UNPROCESSED	MESSAGES	LAST IMPORTED	OPERATIONS
Albums	demo_node_album	Idle	347	347	0	0	2018-03-28 01:25:12	Execute
Artists	demo_node_artist	Idle	275	275	0	0	2018-03-24 21:39:31	Execute

Source data



Human interaction. Where things get messy.

*Myth

“The data is clean and complete”

*Migraine

Believe me. It's not.

Getting the data

- How can we access the data?
  - Direct access? Export? API?
- How do we get updates?
  - How often?
  - Incremental?
  - With timestamps?
- How about assets like PDF's and images?
- What size are we talking about?
  - Number of items, GB's of files

Make some friends at the supplier's side.

Analyze the provided data

Ask the most stupid questions you can think of.

- What does it all mean?
- Does everything have a unique, unchanging ID?
  - Do users have unique email addresses?
  - Do all articles have titles?
- Are records being added and deleted?
  - Yes ⇒ Are unique ID's reused?
- Does the data contain duplicates?
  - No ⇒ Really? Did you check?

Do not assume people know their own data.

Start planning

- Make choices
  - Don't spend 4h automating what takes 8h manually
  - Agree what you will (not) do
- Have the result tested
  - Functional
  - Content
- Write a plan for the go-live
  - Content freeze
  - Pick dates
  - Instruct editors

*Myth

Migrations are Simple

(I know. I said migrations are straight-forward.)

Prepare to write custom processors.

Processors are Awesome

Default process plugin: `Get`

```
process:
  title:
    plugin: get
    source: Name
```

“Map the 'Name' property to the 'title' field”

`Shorthand:`

```
process:
  title: Name
```

Minimal required process config.

Processors can have configuration

```
process:
  type:                          # Node type
    plugin: default_value        # Plugin ID
    default_value: album         # Configuration value
```

In the plugin:

```
$this->configuration['default_value']; // Returns "album".
```

Linking migrations together

The migration_lookup process plugin

```
process:
  field_artist:                       # Entity reference field
    plugin: migration_lookup
    migration: demo_node_artist       # Linked migration ID
    source: ArtistId                  # Source ID
```

“Get the ID of the entity which was created
when demo_node_artist imported this ArtistID.”

Chaining process plugins

```
process:
  uid:
    -
      plugin: author_deduplicate  # Custom deduplication processor
      source: author_id
    -
      plugin: migration_lookup    # Migration Lookup returns a UID
      migration: users
      # No 'source' property, because chaining!
    -
      plugin: default_value       # Result is passed to the next processor
      default_value: 44           # If empty, use UID 44
```

Creating a process plugin

PHP:

```
class MyProcessPlugin extends ProcessPluginBase {}
```

Drupal Console:

```
drupal generate:plugin:migrate:process
```

Drush 9:

```
drush generate plugin-migrate-process
```

A custom processor: Idea

When importing an album...

* Connect to Spotify API
* Enrich with release date and URL

A custom processor

migrate_demo/src/Plugin/migrate/process/SpotifyInfo.php

```
/**
 * Retrieves album info through the Spotify Web API.
 *
 * @MigrateProcessPlugin(
 *   id = "spotify_info",
 * )
 */
class SpotifyInfo extends ProcessPluginBase {
```

migrate_demo/src/Plugin/migrate/process/SpotifyInfo.php

```php
  /**
   * {@inheritdoc}
   */
  public function transform($value, MigrateExecutableInterface $migrate_executabl
    $type = $this->configuration['type'];
    $query_info = $this->getSpotifyQueryInfo($value, $type);
    $spotify_result = $this->spotifyQuery($query_info['query'], $type);

    $property = $this->configuration['property'];
    $data_path = array_merge($query_info['parents'], explode('][', $property));

    return NestedArray::getValue($spotify_result, $data_path);
  }
```

migrate_demo/src/Plugin/migrate/process/SpotifyInfo.php

```
protected function getSpotifyQueryInfo($value, $type) {
  switch ($type) {
    case 'album':
      // Expect $value to be an array: [album title, artist name].
      list($title, $artist) = $value;
      $query = "album:$title artist:$artist";
      $parents = ['albums', 'items', 0];
      break;
  }

  if (empty($query)) {
    throw new MigrateSkipProcessException('Could not build a query.');
  }
  return ['query' => $query, 'parents' => $parents];
}
```

migrate_demo/src/Plugin/migrate/process/SpotifyInfo.php

A custom processor: config

```
process:
  field_release_date:
    plugin: spotify_info
    source:
      - Title
      - ArtistName
    type: album
    property: release_date
  field_spotify_url/uri:    # Link fields are composed of multiple values
    plugin: spotify_info
    source:
      - Title
      - ArtistName
    type: album
    property: external_urls][spotify          # Will be split into an array
```

*Migraine

A migration never comes alone

If everything was perfect, there wouldn't be a migration, right?

Client goals

What they say	What it means
New site	New URL's
New design	An image on every node
New navigation	Revised categories
New workflow	Different input filters

...and the existing content will magically fit in.

Complexity = changes²

Reduce the number of changes introduced with the migration.

Spotify process plugin = A really bad idea

*Must

Start early

Adapt your site to old content *and* future needs.

Migrate early, keep importing.

Estimate generously. Double it.

Tips & tricks

(If we have time)

Save time: performance

* Disable search indexing
* Run migrations with Drush

Save time: reduced dataset

settings.local.php

```
$settings['my_migration_reduce_factor'] = 100;
```

mySourcePlugin::query()

```
$reduce_factor = Settings::get('my_migration_reduce_factor');
if ($reduce_factor && is_int($reduce_factor)) {
  $query->where('MOD(a.id, :reduce_factor) = 0',
    [':reduce_factor' => $reduce_factor]);
}
```

Imports approximately 1 in every 100 items.

Manual edits... now what?

Run a migration on selected fields:

```
destination:
  plugin: 'entity:node'
  overwrite_properties:
    - category
```

`overwrite_properties` ignores all values except explicitly listed.

Shortcut: entity_generate

```
process:
  tags:
    -
      plugin: explode
      source: keywords          # Eg. "Foo,Bar,Baz"
      delimiter: ','            # Explode comma separated string to array
    -
      plugin: entity_generate  # Generate taxonomy terms that don't exist
```

No rollbacks, no mappings, no nothing.

Great for things that don't have a source ID.

Come for the software, stay for the community

Before we get to the questions...

Thank you!

Diamond Sponsor

Platinum Sponsors

Gold Sponsors

You – Drupal Community

LimoenGroen

Questions?