

State Machines

Tobias Nyholm
@tobiasnyholm

<https://joind.in/talk/cb240>



What?

a way of thinking

Why state machines?

- Separates business logic from your models
- Makes the code more:
 - Reusable
 - Maintainable
 - Readable

Tobias Nyholm

- Full stack unicorn on Happyr.com
- Sound of Symfony podcast
- Certified Symfony developer
- PHP-Stockholm
- Open source

Open source

Neo4j

FriendsOfApi/boilerplate

Guzzle

Buzz

Swap

NSA

LinkedIn API client

PSR7

PHP-cache
CacheBundle

~~JMSTranslationBundle~~
PHP-Translation

KNP Github API

Mailgun

Stampie

HTTPPlug

SymfonyBundleTest

SimpleBus integrations

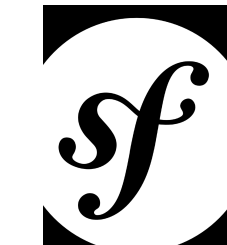
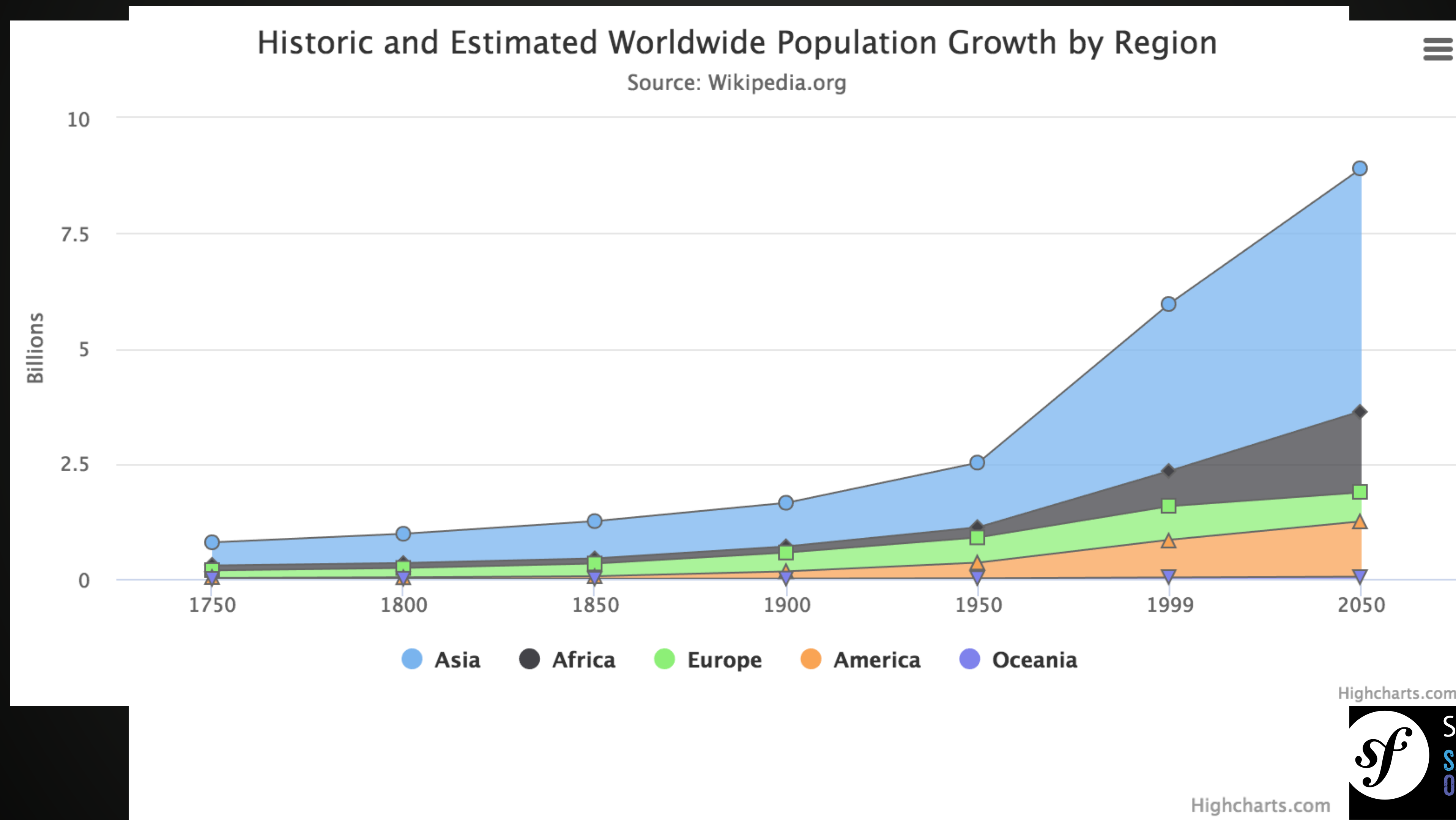
PHP-Geocoder

PSR HTTP clients

BazingaGeocoderBundle

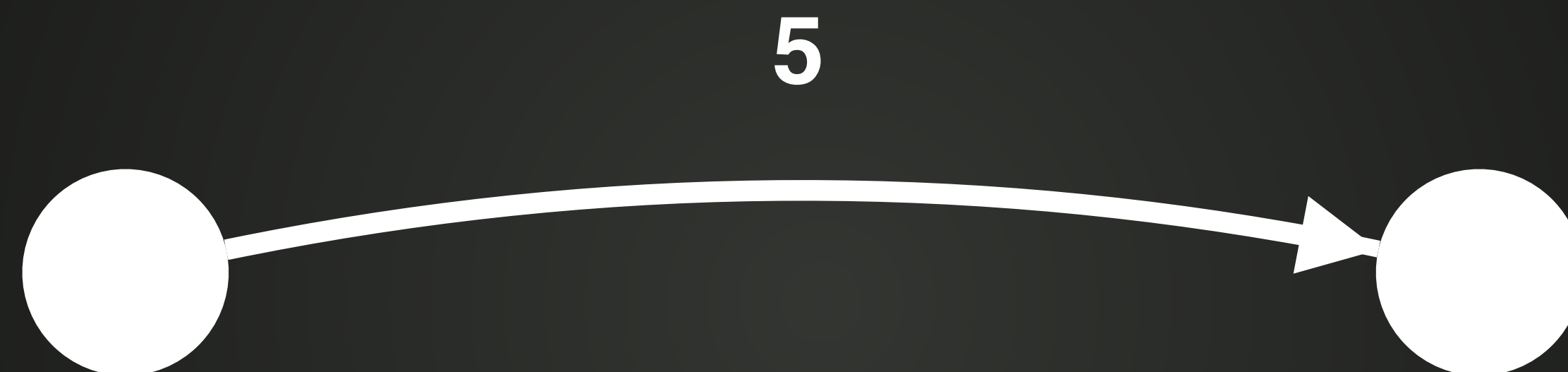
So how?

Graph theory

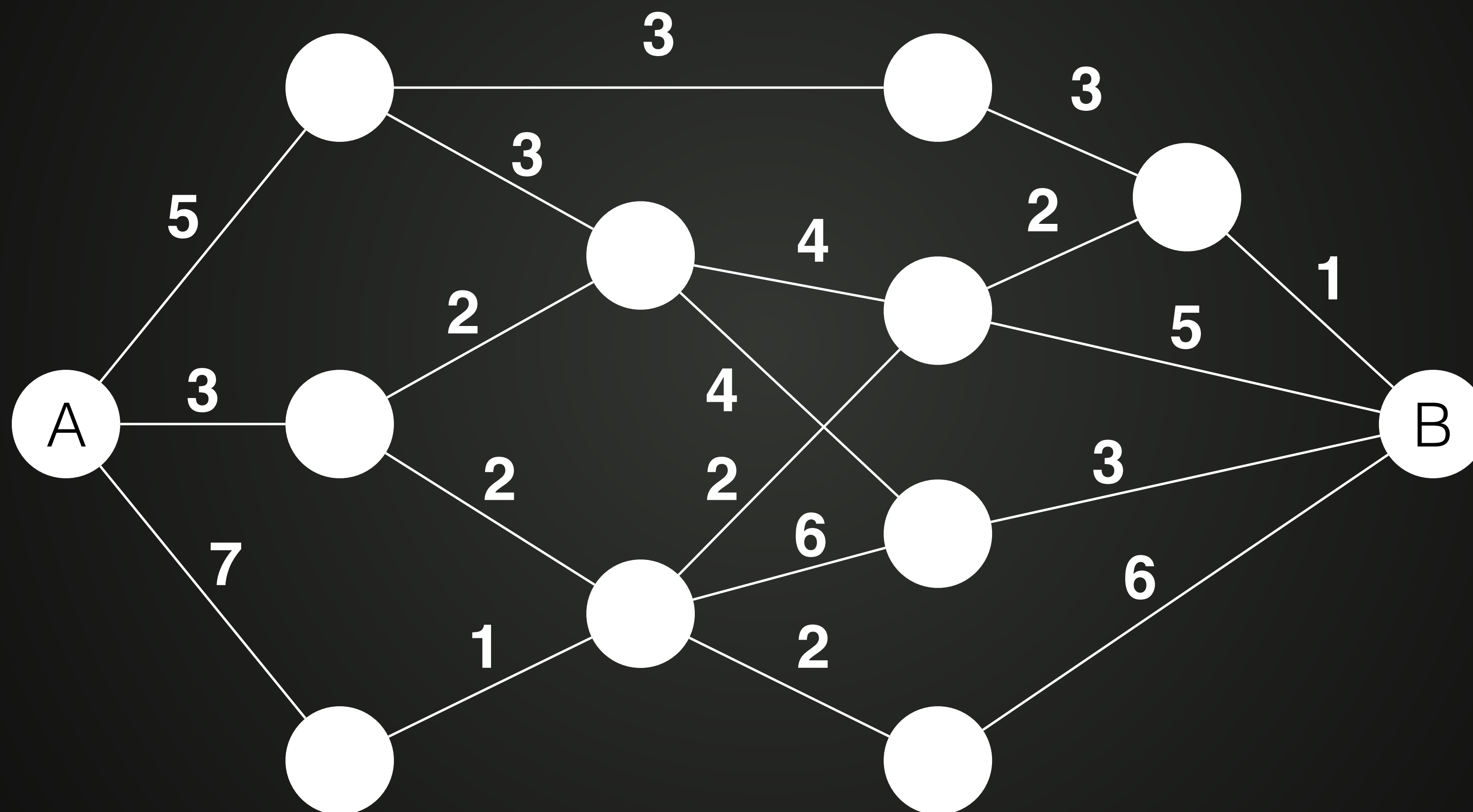


Symphony Live
SAN FRANCISCO 2017
OCTOBER 19-20

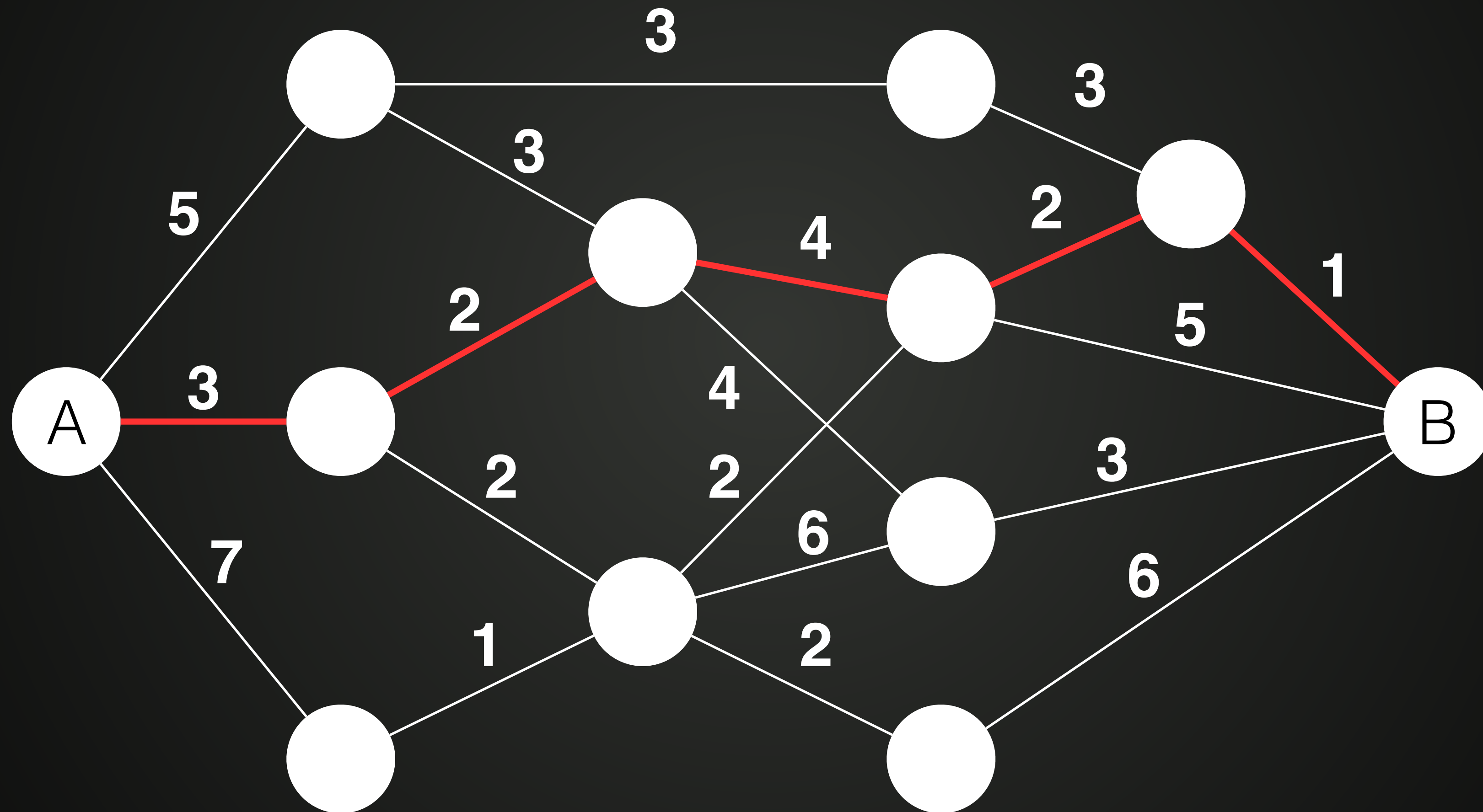
Graph theory



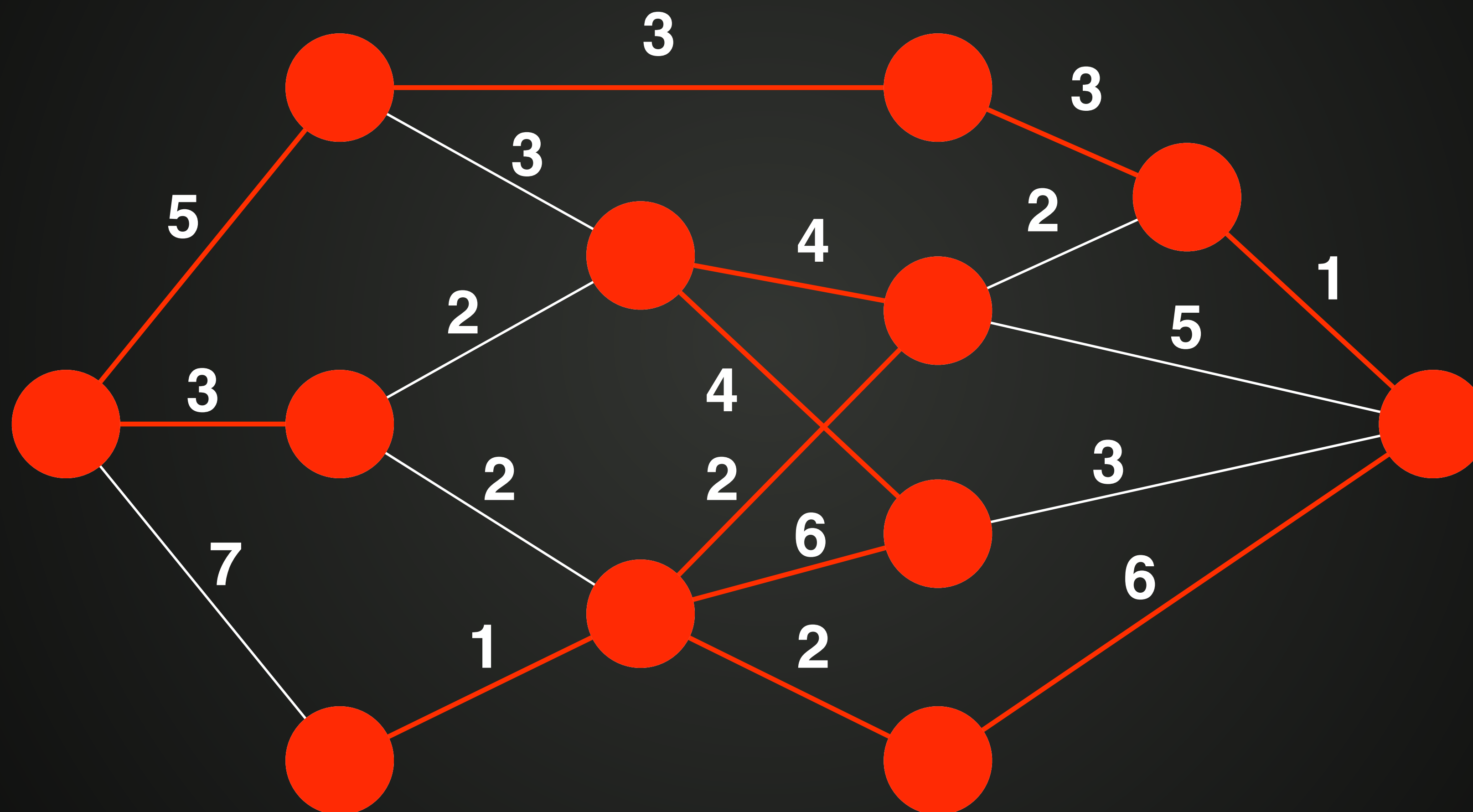
Graph theory



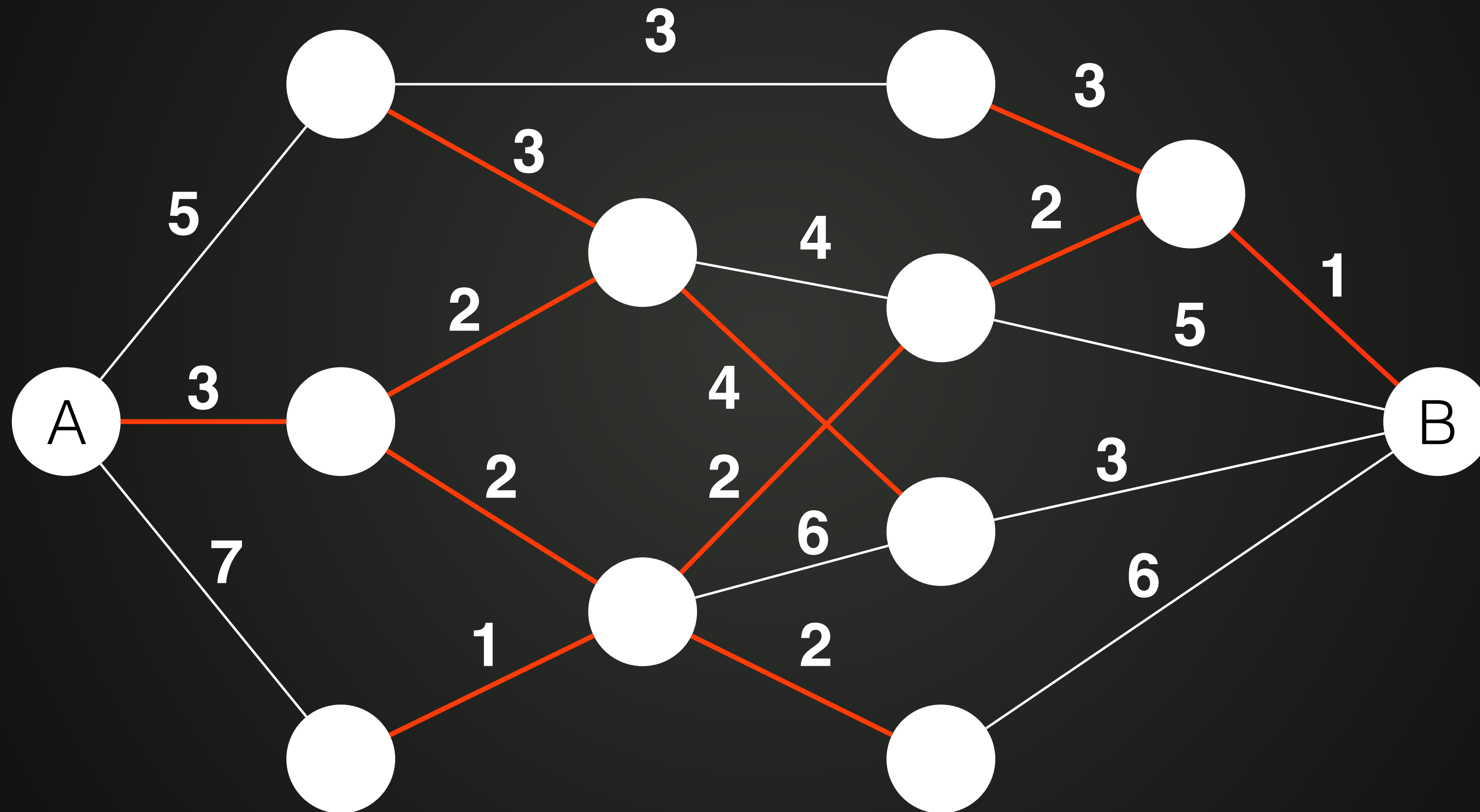
Graph theory



Graph theory

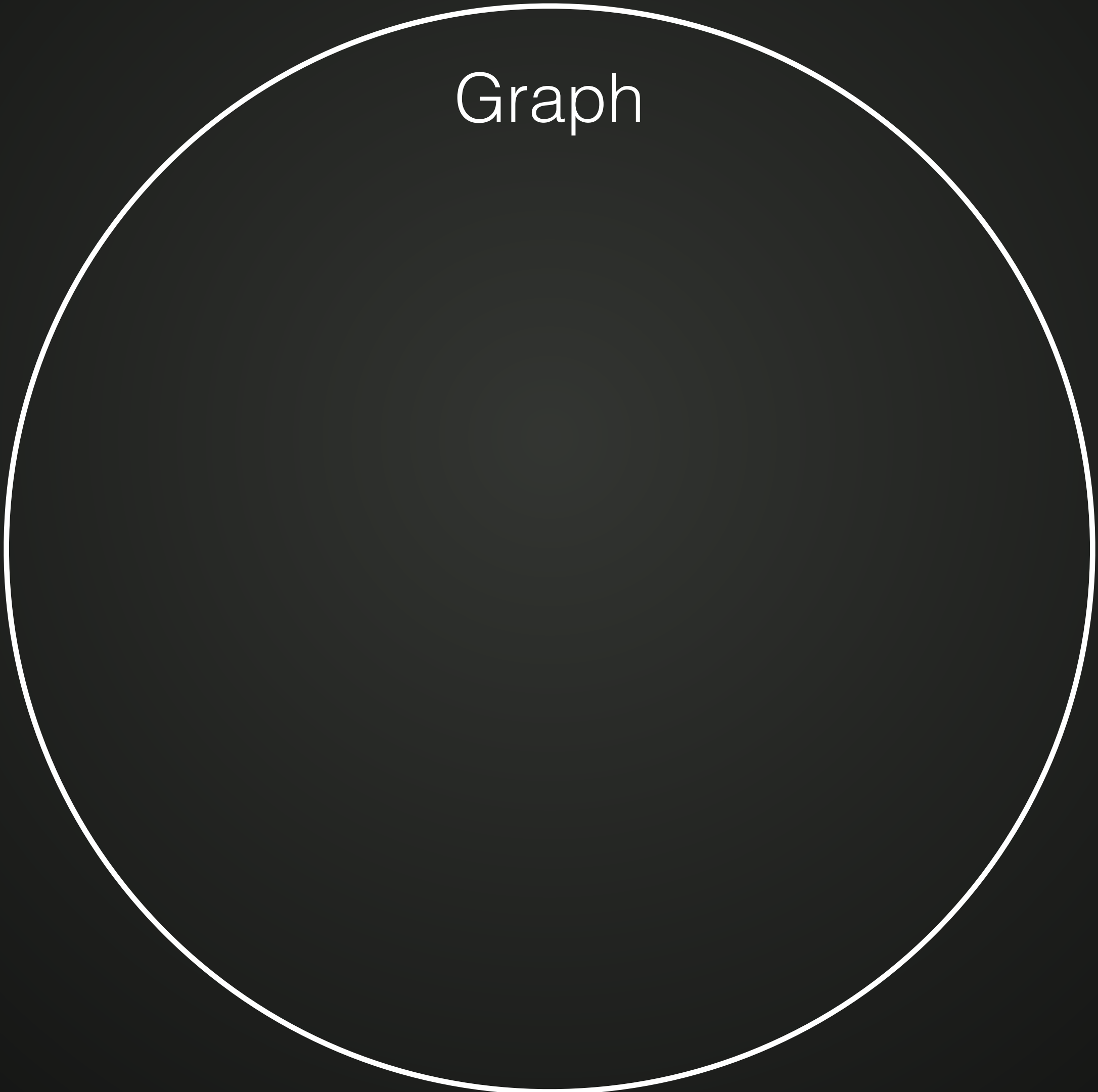


Graph theory

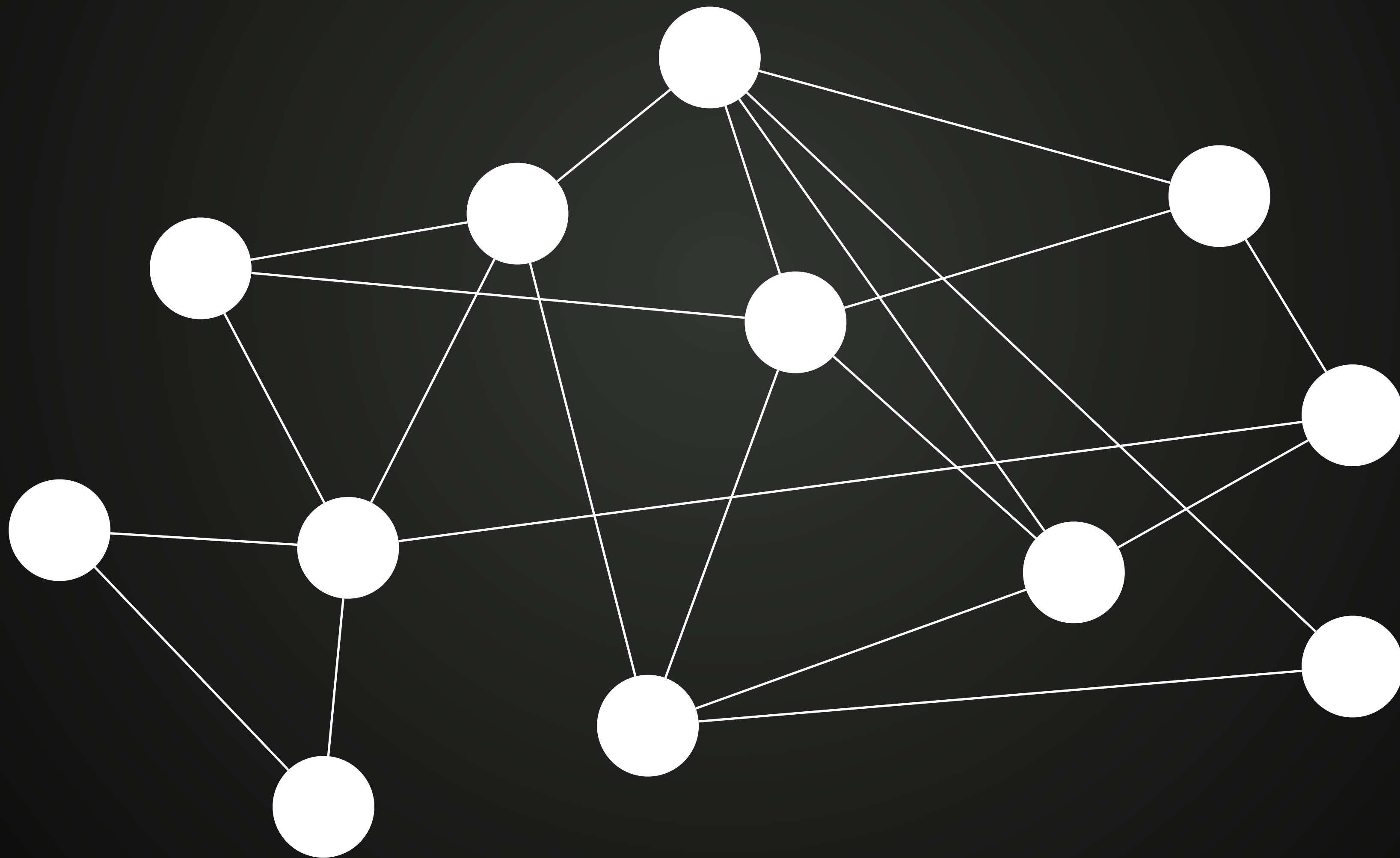


Theory

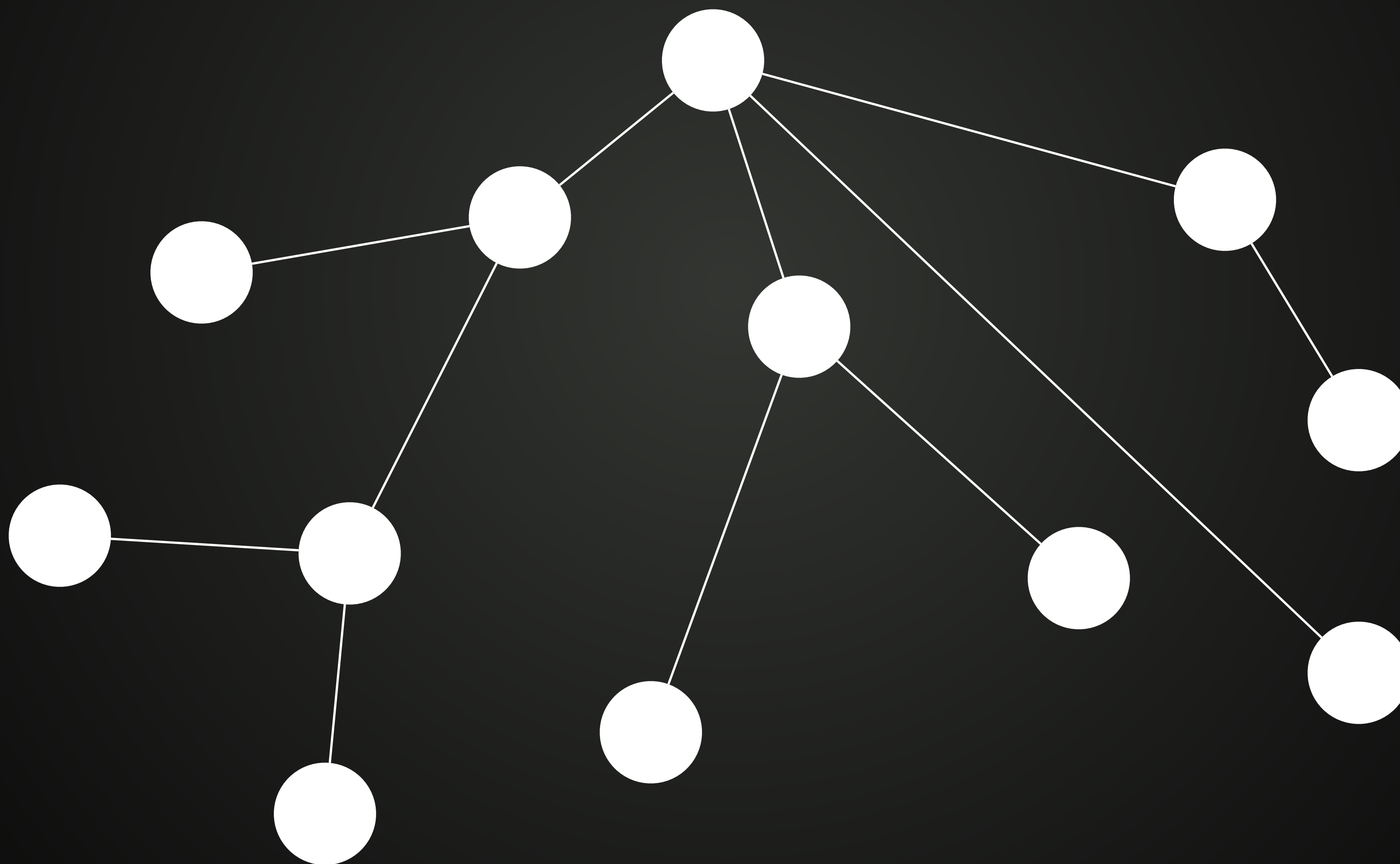
Graph

A large white circle outline is centered on the page, below the word 'Theory' and above the word 'Graph'. The circle is empty and serves as a visual element or a placeholder for a diagram.

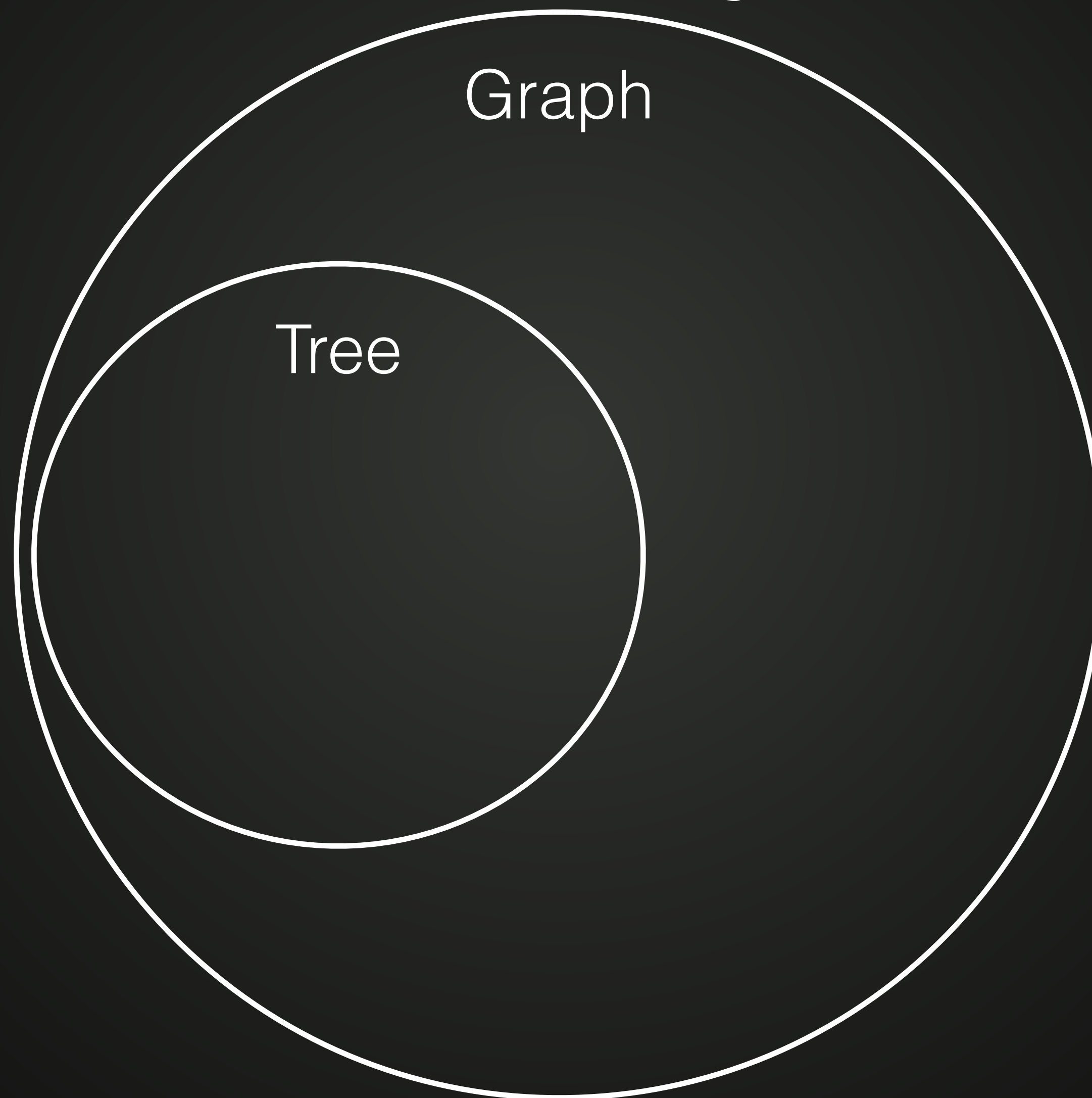
Graph theory



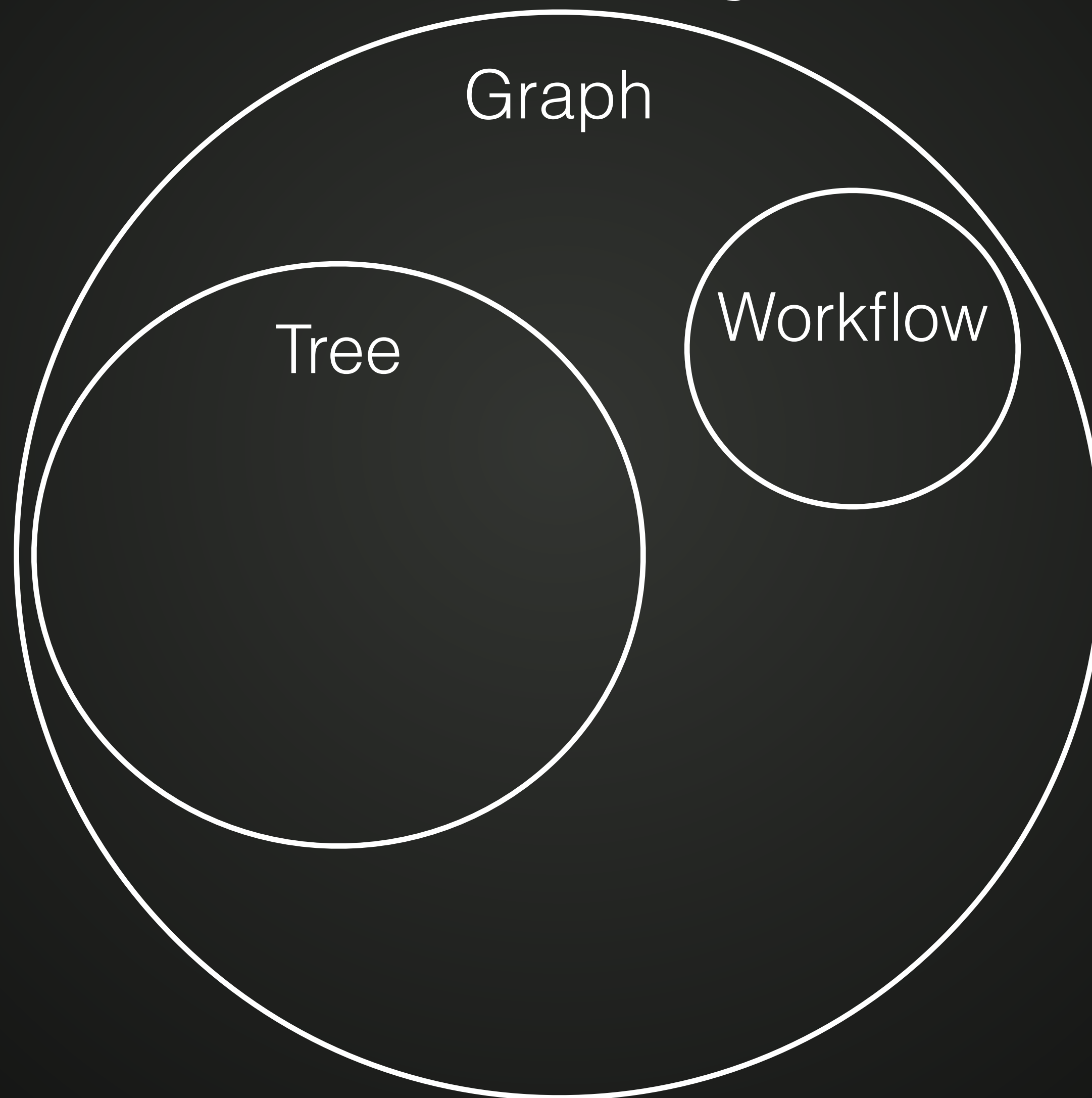
Graph theory



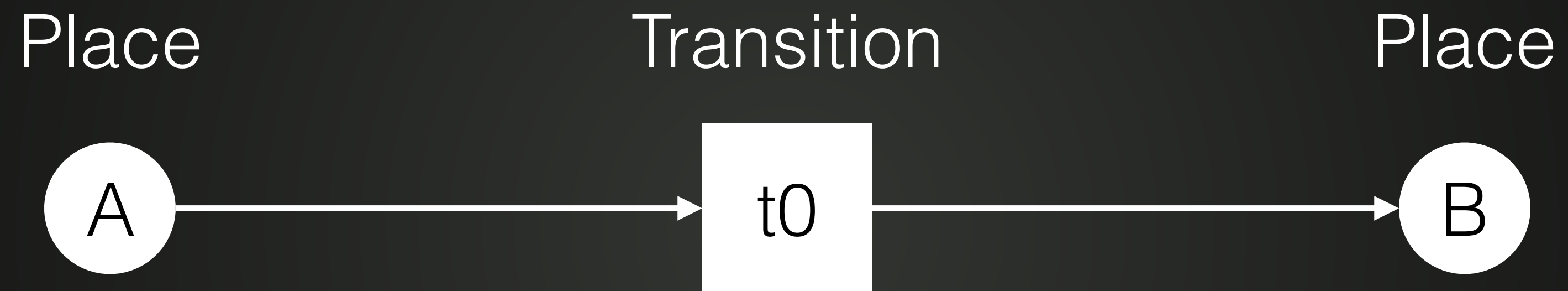
Theory



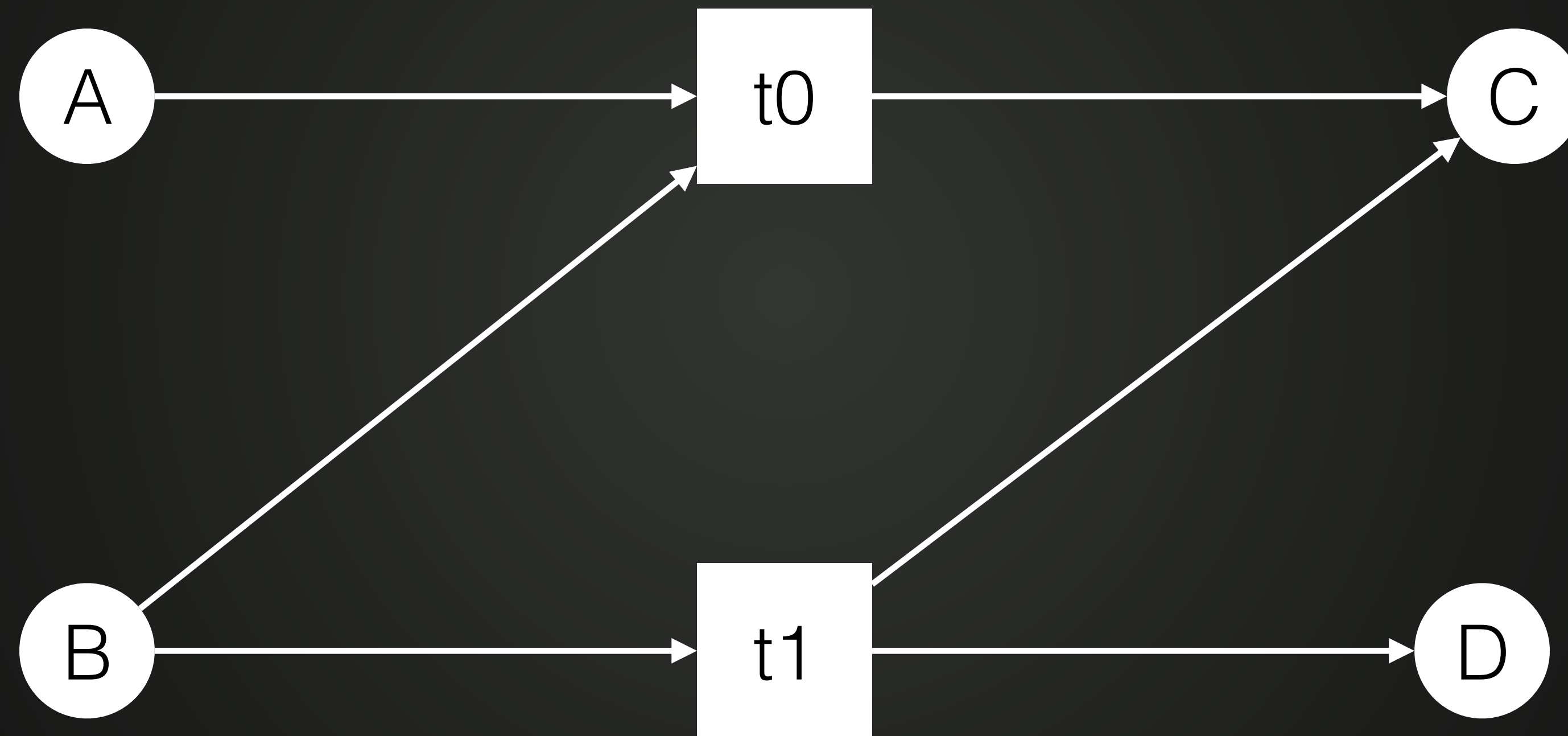
Theory



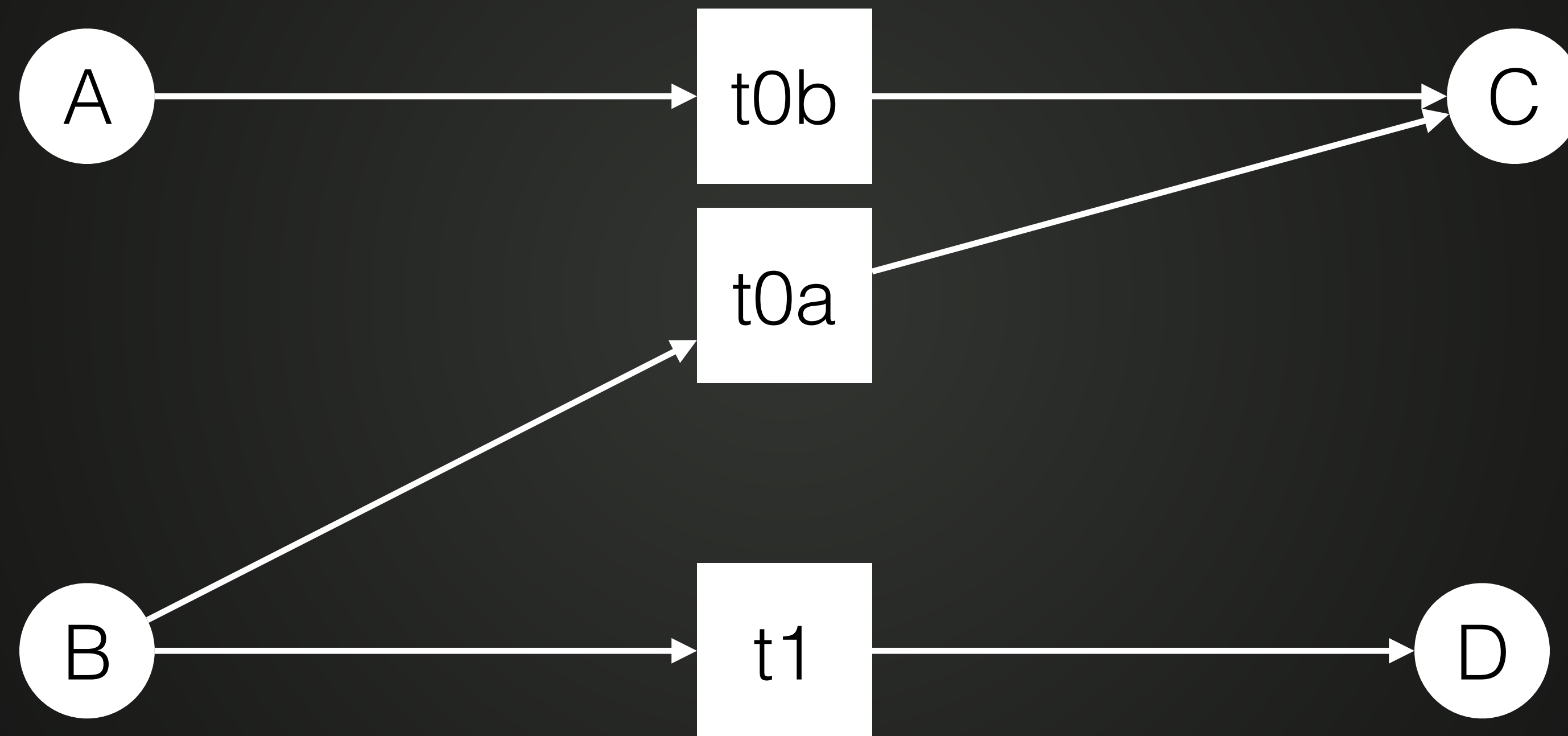
Workflow



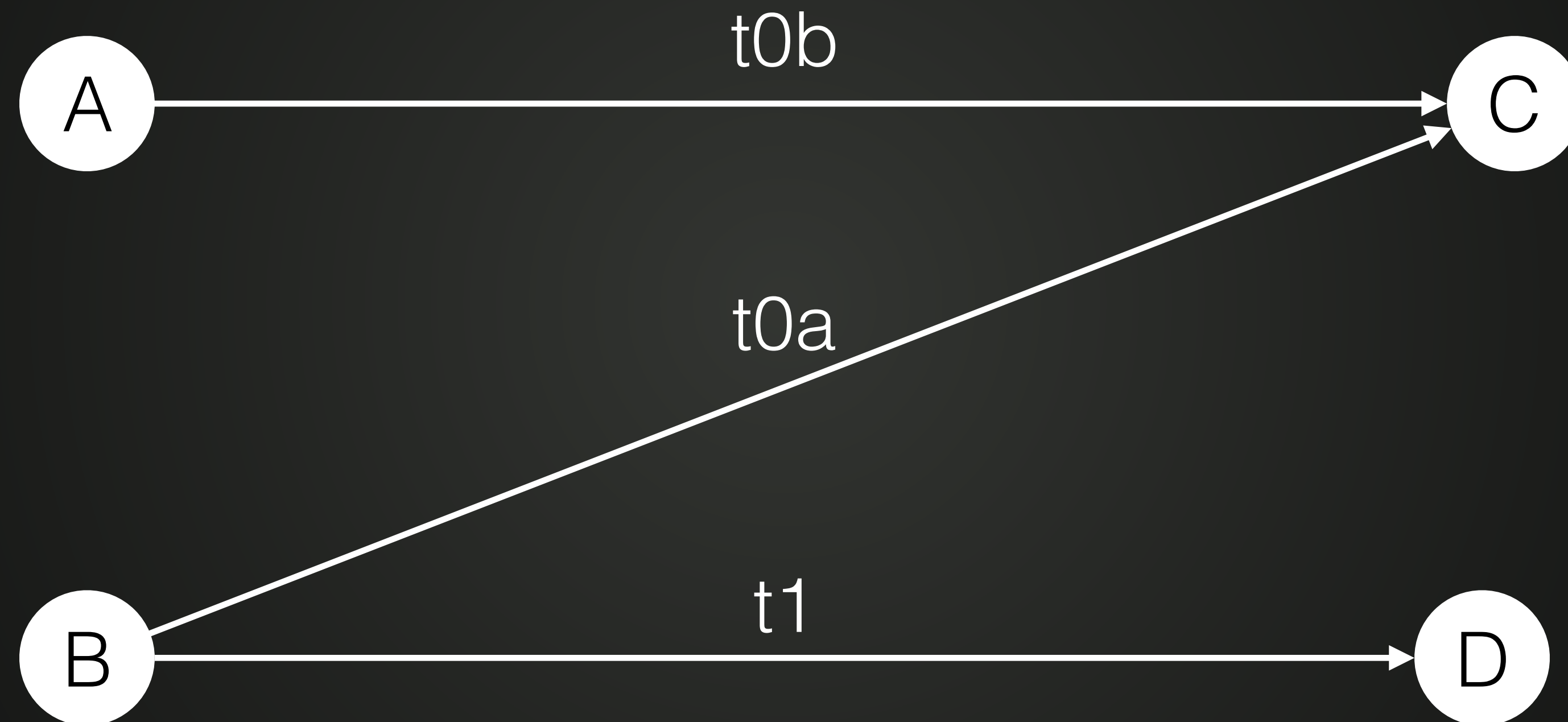
Workflow



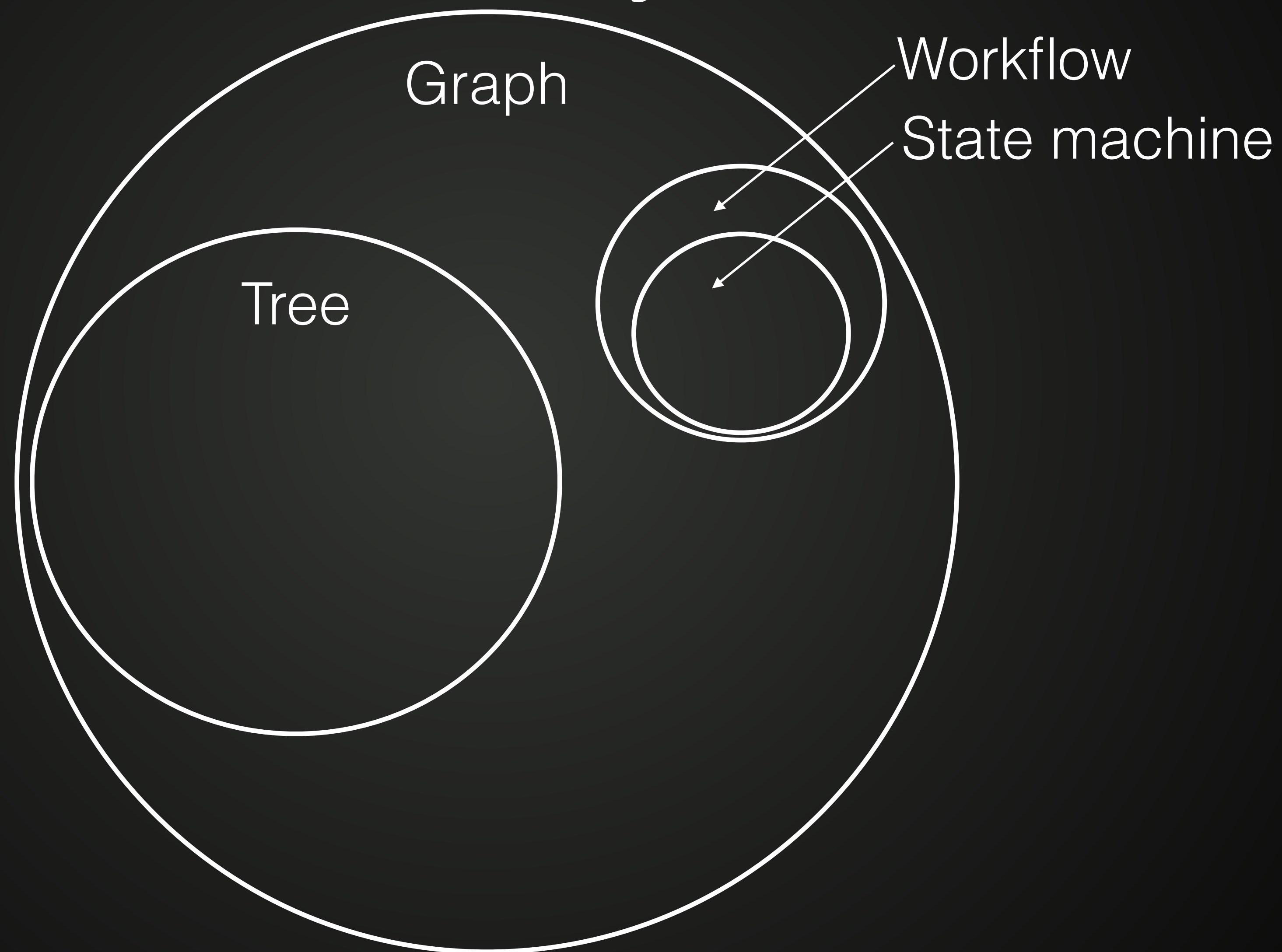
State Machine



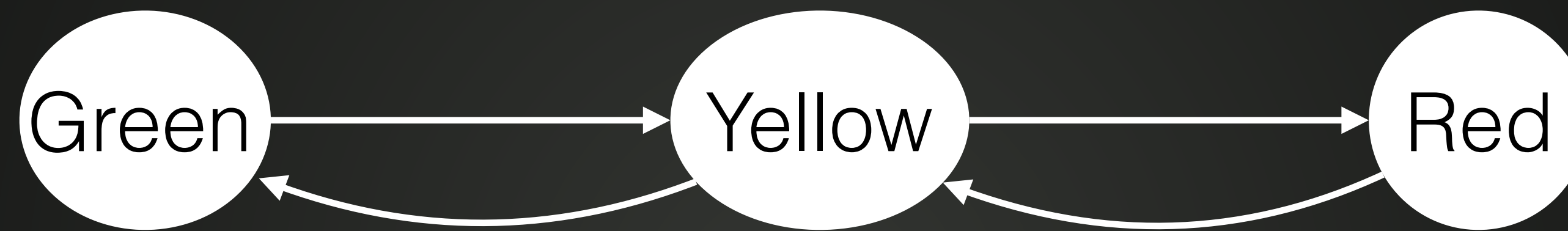
State Machine



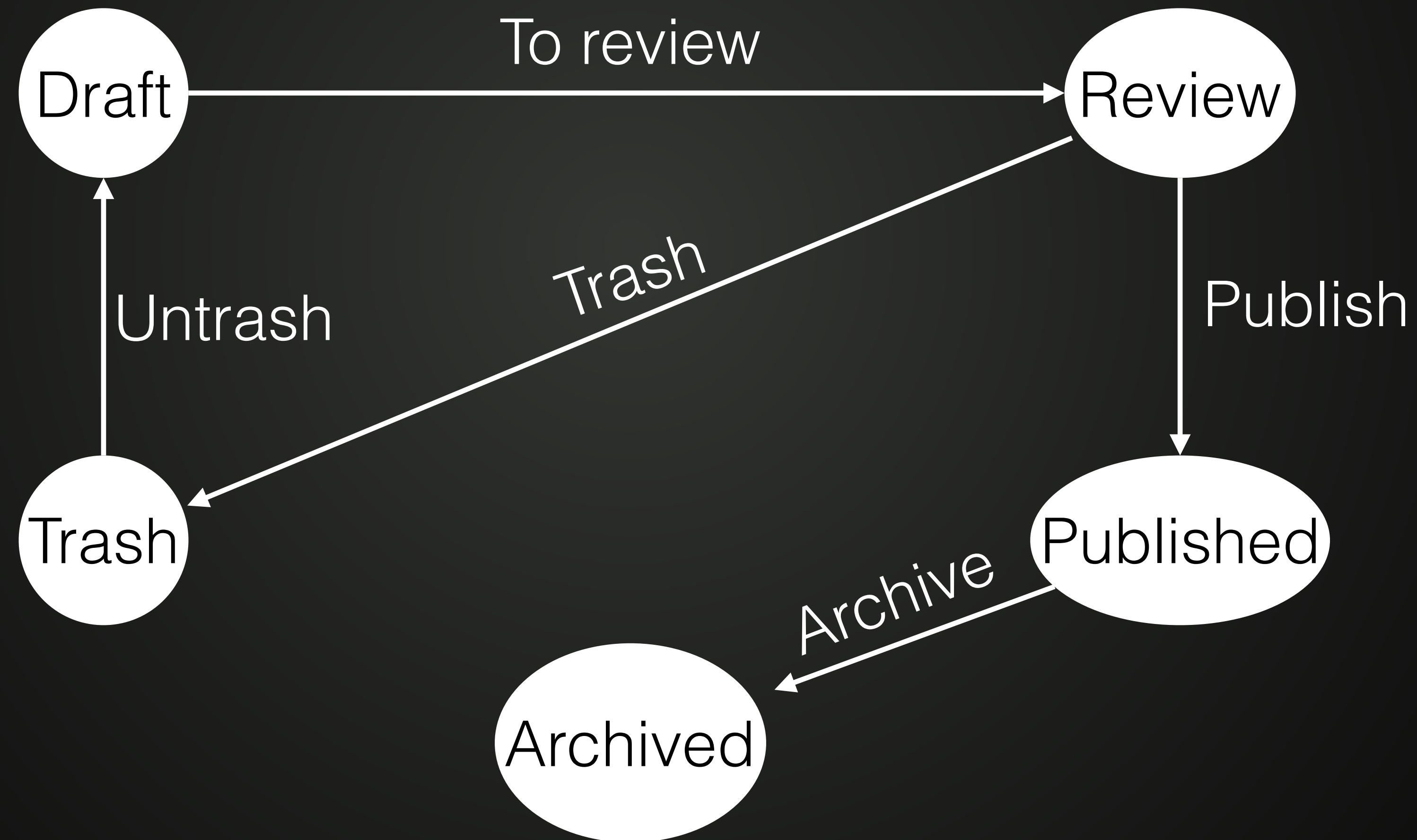
Theory



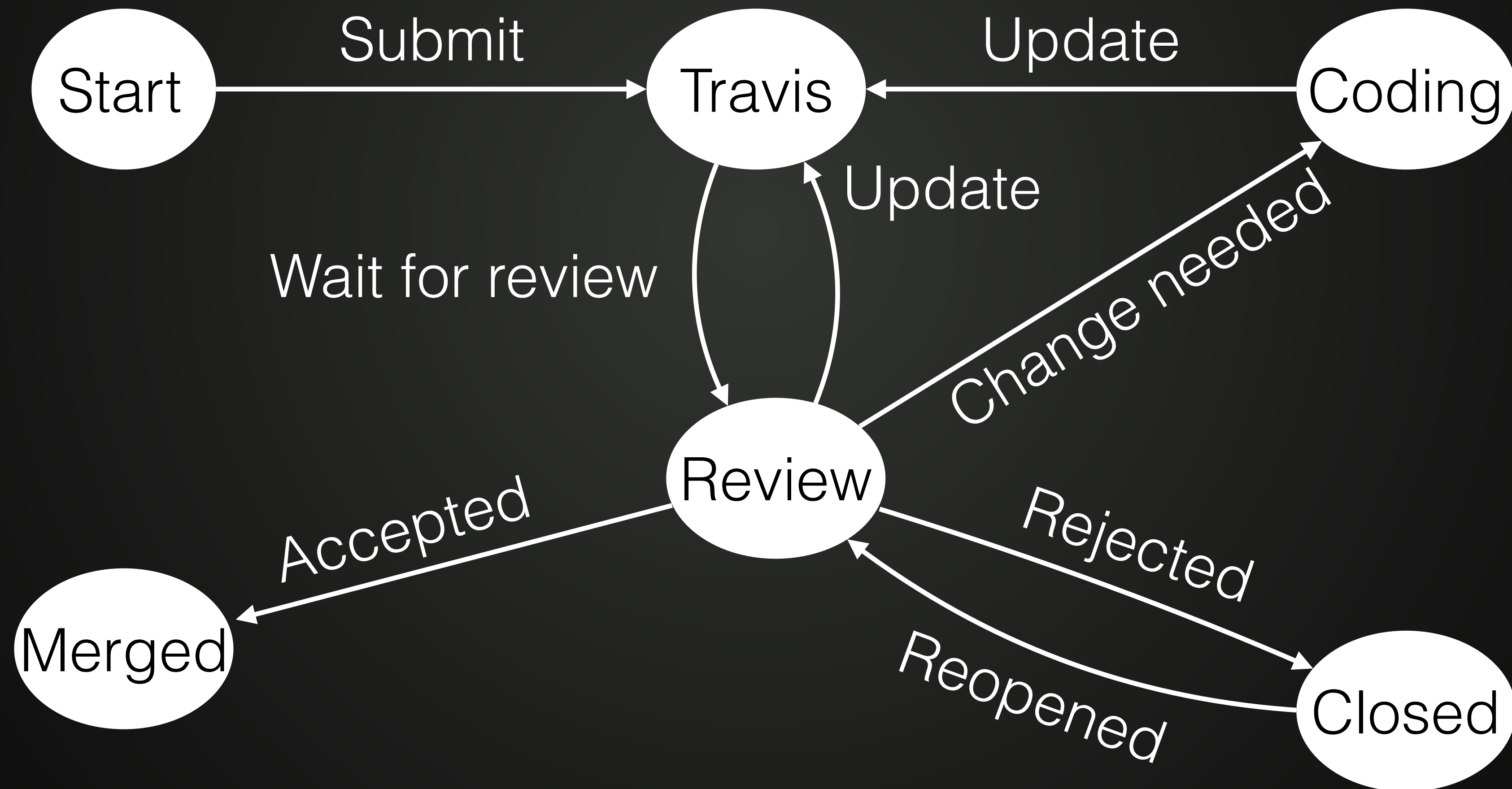
Traffic Light



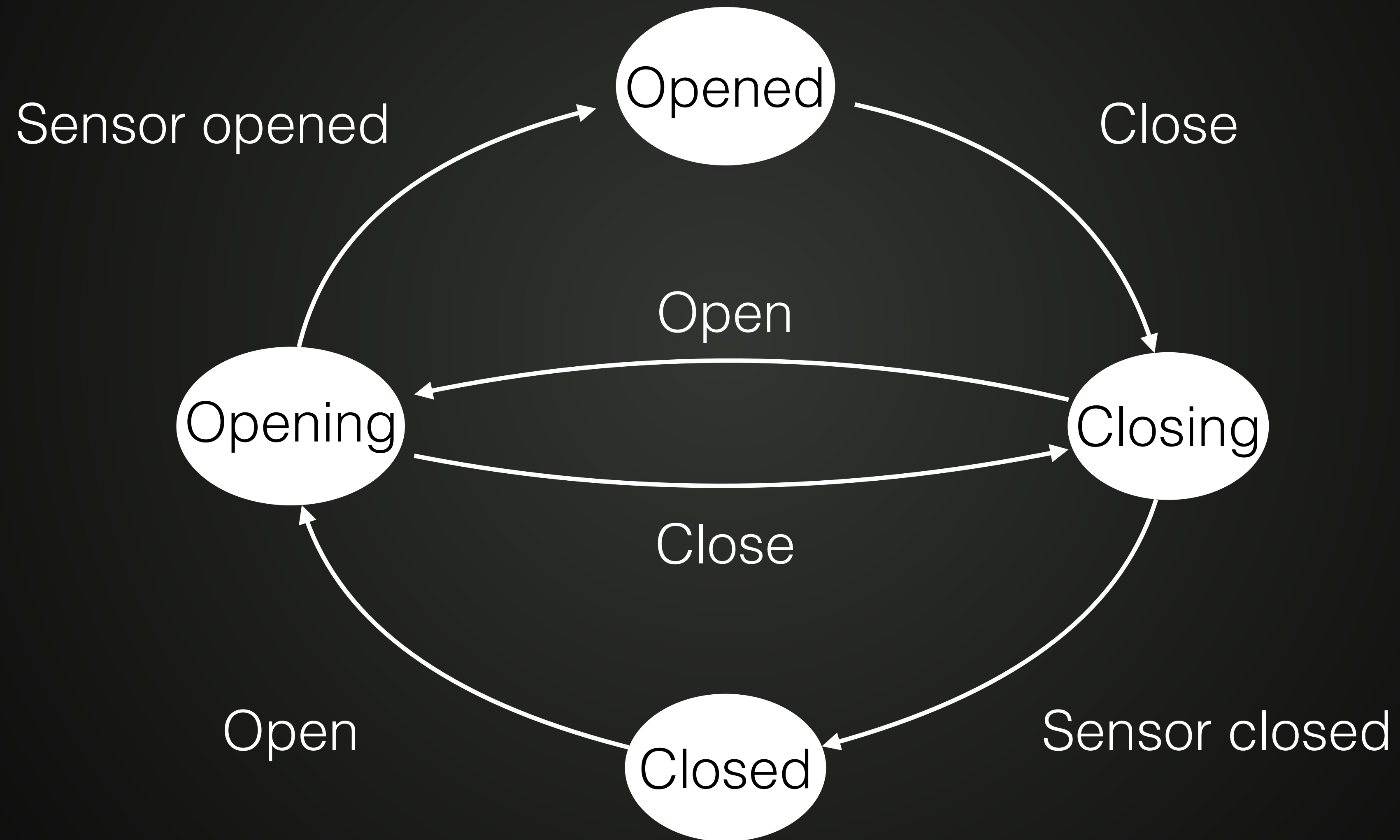
Job advert



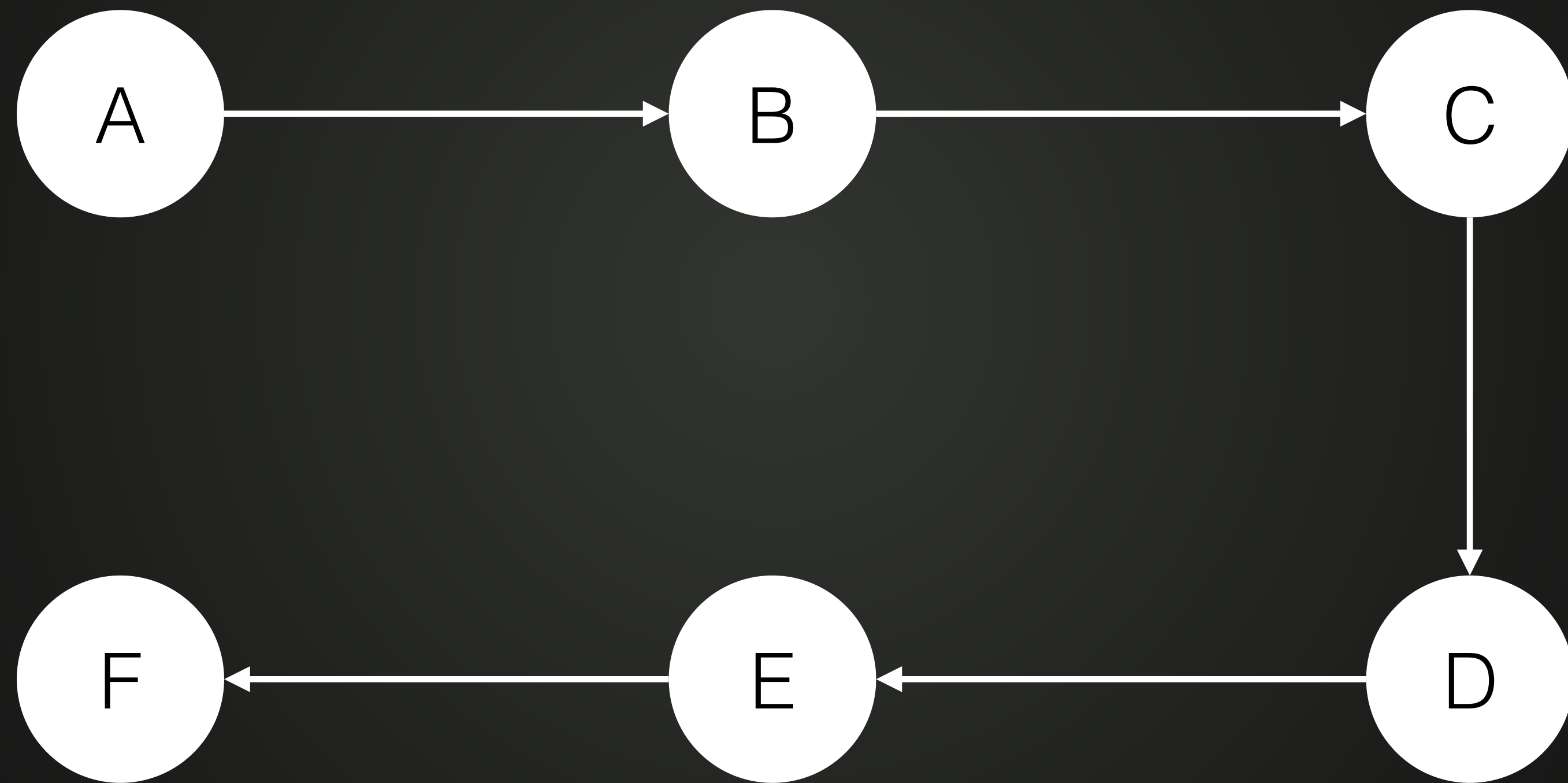
Pull Request



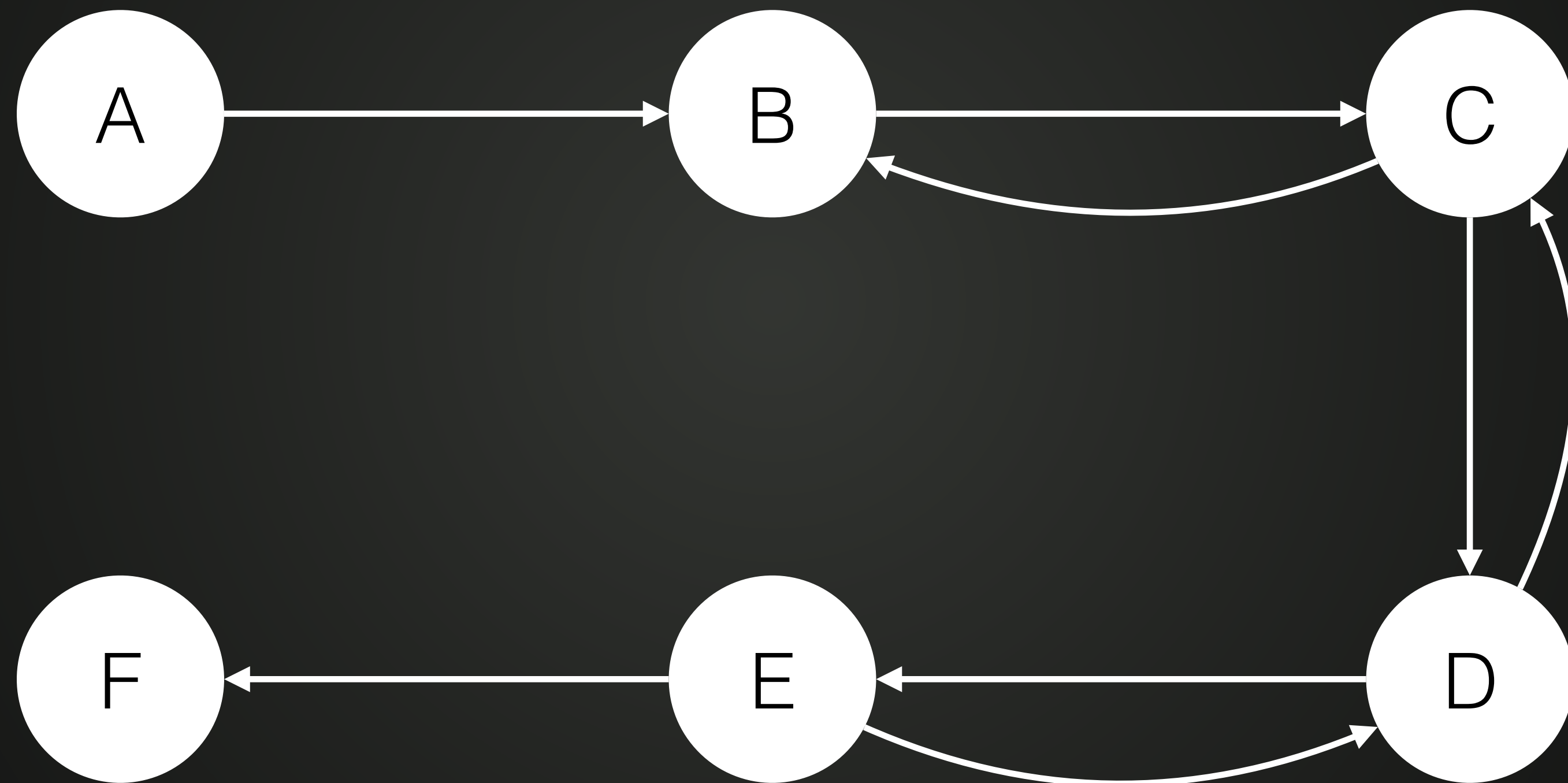
Elevator doors



Multi step form



Multi step form

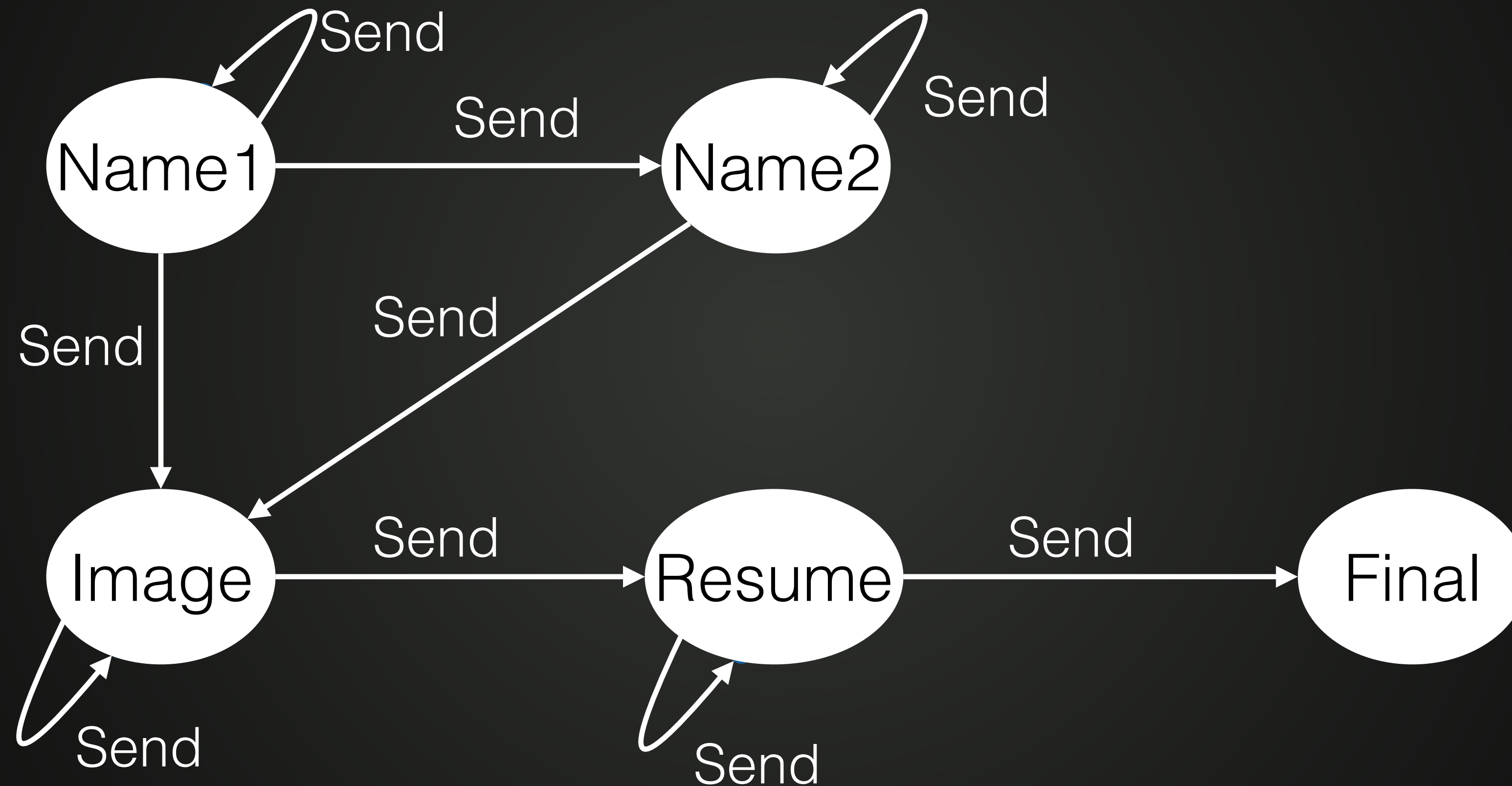


Two Implementations

State Pattern

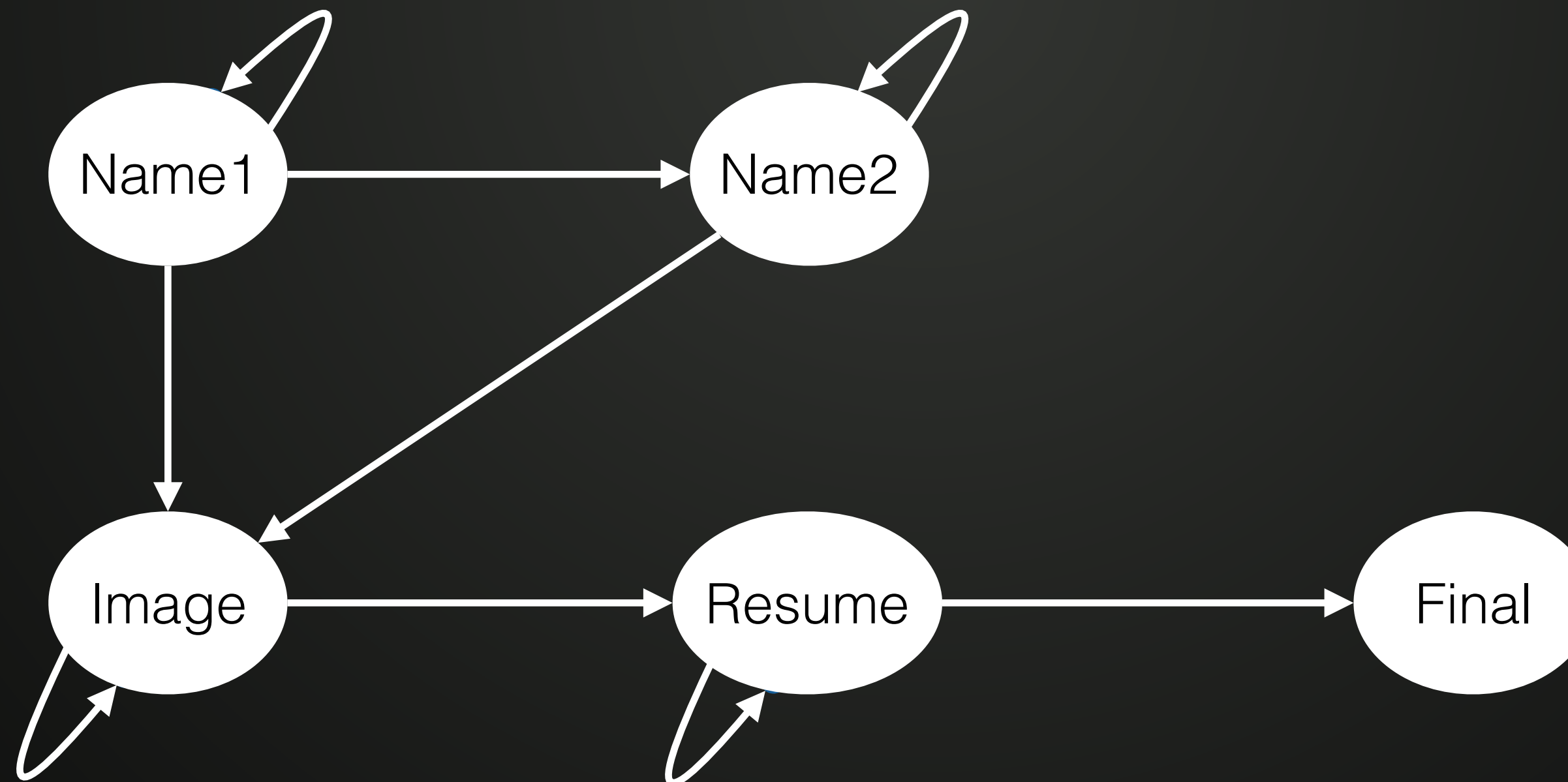
Moore

Complete Profile Reminder




```
class Worker
{
    // Call daily
    public function sendReminders()
    {
        $stateMachines = ...; // Fetch from database

        /** @var ProfileReminder $profileReminder */
        foreach ($stateMachines as $stateMachine) {
            $sended = $stateMachine->start($this->emailService);
            if ($sended === true) {
                // Remove $profileReminder from database
            }
        }
    }
}
```




```
class CompleteYourProfileReminderStateMachine {
    private $user; /** @var User */
    private $state; /** @var ProfileReminderState */

    public function __construct(User $user) {
        $this->user = $user;
        $this->state = new AddYourNameFirst();
    }

    /**
     * @param AnyEmailService $emailService
     *
     * @return bool true if this state machine has come to an end.
     */
    public function start(AnyEmailService $emailService) {
        while (true !== $this->state->send($this, $emailService)) {
            // Looping through states.
        }

        return $this->state instanceof FinalState;
    }

    public function getUser() {
        return $this->user;
    }

    public function setState(ProfileReminderState $state) {
        $this->state = $state;
    }
}
```

```
class AnyStep implements ProfileReminderState
{
    public function send(Compl...StateMachine $stateMachine, $emailService)
    {
        // Send email and set next state
        $emailService->send(
            'Mail\AddYourNameFirst.html.twig',
            ['user' => $stateMachine->getUser()]
        );

        $stateMachine->setState(new AddYourNameSecondReminder());

        return false;
    }
}
```

```
class AnyStep implements ProfileReminderState
{

    public function send(Compl...StateMachine $stateMachine, $emailService)
    {
        // If [precondition] send to other state.
        if ($stateMachine->getUser()->hasName()) {
            $stateMachine->setState(new AddYourImage());
            return false;
        }

        // Send email and set next state
        $emailService->send(
            'Mail\AddYourNameFirst.html.twig',
            ['user' => $stateMachine->getUser()]
        );

        $stateMachine->setState(new AddYourNameSecondReminder());

        return false;
    }
}
```



```
class AnyStep implements ProfileReminderState
{
    private $createdAt;
    public function __construct() {
        $this->createdAt = new \DateTime();
    }

    public function send(Compl...StateMachine $stateMachine, $emailService)
    {
        // If [precondition] send to other state.
        if ($stateMachine->getUser()->hasName()) {
            $stateMachine->setState(new AddYourImage());
            return false;
        }

        // If we have been at this step for less than seven days. Do nothing
        if ($this->createdAt > new \DateTime('-7days')) {
            return true;
        }

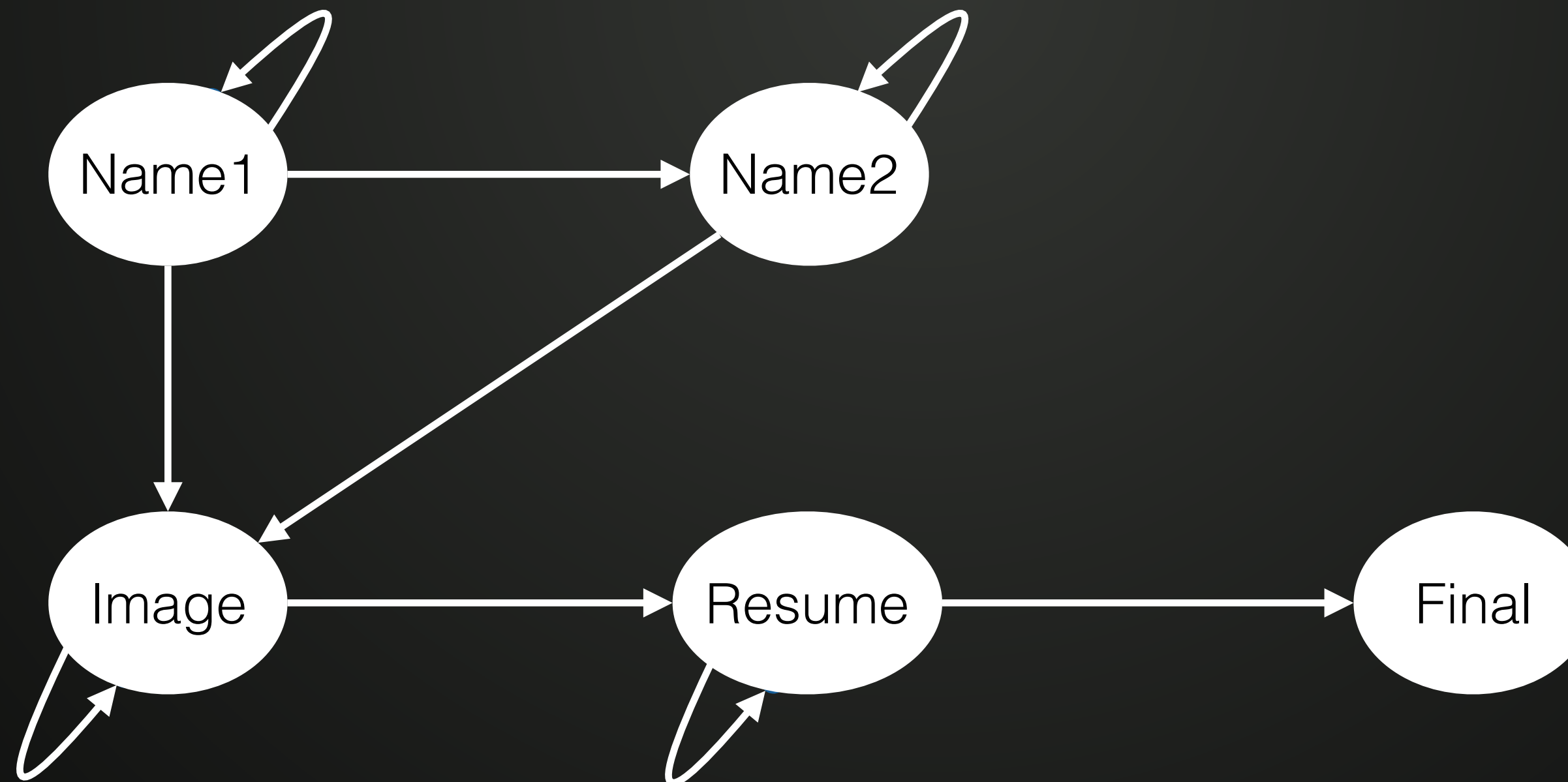
        // Send email and set next state
        $emailService->send(
            'Mail\AddYourNameFirst.html.twig',
            ['user' => $stateMachine->getUser()]
        );
        $stateMachine->setState(new AddYourNameSecondReminder());
        return false;
    }
}
```


src/AppBundle/StatePattern/

- Worker.php
- Email/
 - AddYourImage.php
 - AddYourNameFirstReminder.php
 - AddYourNameSecondReminder.php
 - AddYourResume.php
 - CompleteYourProfileReminderStateMachine.php
 - FinalState.php
 - ProfileReminderState.php

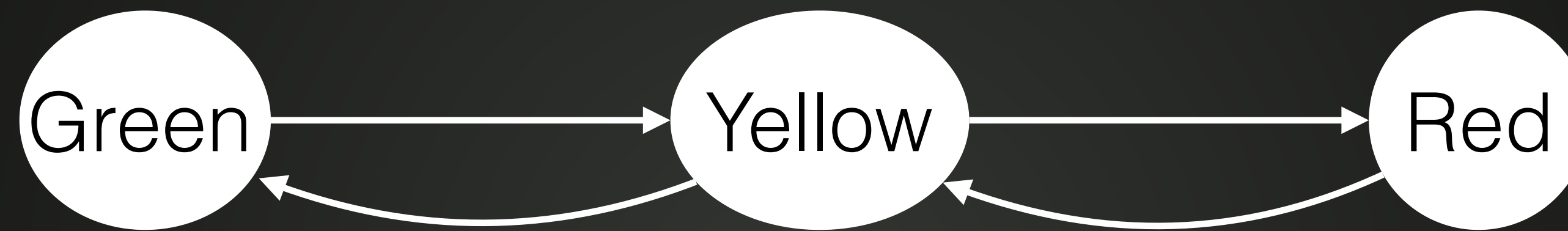
```
class Worker
{
    // Call daily
    public function sendReminders()
    {
        $stateMachines = ...; // Fetch from database

        /** @var ProfileReminder $profileReminder */
        foreach ($stateMachines as $stateMachine) {
            $sended = $stateMachine->start($this->emailService);
            if ($sended === true) {
                // Remove $profileReminder from database
            }
        }
    }
}
```



Mealy State Machine

Traffic Light



<?php

@tobiasnyholm

```
namespace App;
```

```
class StateMachine
```

```
{
```

```
    const STATE_GREEN = 0;
```

```
    const STATE_YELLOW = 1;
```

```
    const STATE_RED = 2;
```

```
    private $state;
```

```
    public function can($transition) {
```

```
        switch ($this->state) {
```

```
            case self::STATE_GREEN:
```

```
                return ($transition === 'to_yellow');
```

```
            case self::STATE_YELLOW:
```

```
                return ($transition === 'to_green' || $transition === 'to_red');
```

```
            case self::STATE_RED:
```

```
                return ($transition === 'to_yellow');
```

```
            default:
```

```
                return false;
```

```
        }
```

```
    }
```

```
    // ..
```

```
}
```

<?php

@tobiasnyholm

```
namespace App;
```

```
class StateMachine
```

```
{
```

```
    // ..
```

```
    public function apply($transition) {
```

```
        if (!$this->can($transition)) {
```

```
            throw new \InvalidArgumentException('Invalid transition');
```

```
        }
```

```
        switch ($this->state) {
```

```
            case self::STATE_GREEN && ($transition === 'to_yellow'):
```

```
                $this->state = self::STATE_YELLOW;
```

```
                break;
```

```
            case self::STATE_YELLOW && ($transition === 'to_green'):
```

```
                $this->state = self::STATE_GREEN;
```

```
                break;
```

```
            case self::STATE_YELLOW && ($transition === 'to_red'):
```

```
                $this->state = self::STATE_RED;
```

```
                break;
```

```
            case self::STATE_RED && ($transition === 'to_yellow'):
```

```
                $this->state = self::STATE_YELLOW;
```

```
                break;
```

```
        }
```

```
    }
```

```
}
```

<?php

@tobiasnyholm

```
namespace App;
```

```
class StateMachine
```

```
{
```

```
    const STATE_GREEN = 0;
```

```
    const STATE_YELLOW = 1;
```

```
    const STATE_RED = 2;
```

```
    public function can(TrafficLight $trafficLight, $transition) {  
        $state = $trafficLight->getCurrentState();
```

```
        switch ($state) {
```

```
            case self::STATE_GREEN:
```

```
                return ($transition === 'to_yellow');
```

```
            case self::STATE_YELLOW:
```

```
                return ($transition === 'to_green' || $transition === 'to_red');
```

```
            case self::STATE_RED:
```

```
                return ($transition === 'to_yellow');
```

```
            default:
```

```
                return false;
```

```
        }
```

```
    }
```

```
    // ..
```

```
}
```


<?php

@tobiasnyholm

```
namespace App;
```

```
class StateMachine
```

```
{
```

```
    // ..
```

```
    public function apply(TrafficLight $trafficLight, $transition) {  
        if (!$this->can($trafficLight, $transition)) {  
            throw new \InvalidArgumentException('Invalid transition');  
        }  
  
        $state = $trafficLight->getCurrentState();  
  
        switch ($state) {  
            case self::STATE_GREEN && ($transition === 'to_yellow'):  
                $trafficLight->setState(self::STATE_YELLOW);  
                break;  
            case self::STATE_YELLOW && ($transition === 'to_green'):  
                $trafficLight->setState(self::STATE_GREEN);  
                break;  
            case self::STATE_YELLOW && ($transition === 'to_red'):  
                $trafficLight->setState(self::STATE_RED);  
                break;  
            case self::STATE_RED && ($transition === 'to_yellow'):  
                $trafficLight->setState(self::STATE_YELLOW);  
                break;  
        }  
    }  
}
```



```
<?php

namespace App;

class StateMachine
{
    // ...

    public function allowTraffic(TrafficLight $trafficLight)
    {
        if ($this->can($trafficLight, 'to_green')) {
            $this->apply($trafficLight, 'to_green');
        } elseif ($this->can($trafficLight, 'to_yellow')) {
            $this->apply($trafficLight, 'to_yellow');
        }
    }

    public function stopTraffic(TrafficLight $trafficLight)
    {
        if ($this->can($trafficLight, 'to_red')) {
            $this->apply($trafficLight, 'to_red');
        } elseif ($this->can($trafficLight, 'to_yellow')) {
            $this->apply($trafficLight, 'to_yellow');
        }
    }
}
```

```
<?php

namespace App;

class StateMachine
{
    // ...

    public function allowTraffic(TrafficLight $trafficLight)
    {
        if ($this->can($trafficLight, 'to_green')) {
            $this->apply($trafficLight, 'to_green');
        } elseif ($this->can($trafficLight, 'to_yellow')) {
            $this->apply($trafficLight, 'to_yellow');
        }
    }

    public function stopTraffic(TrafficLight $trafficLight)
    {
        if ($this->can($trafficLight, 'to_red')) {
            $this->apply($trafficLight, 'to_red');
        } elseif ($this->can($trafficLight, 'to_yellow')) {
            $this->apply($trafficLight, 'to_yellow');
        }
    }
}
```



```
class StateMachine{
    const STATE_GREEN = 0;
    const STATE_YELLOW = 1;
    const STATE_RED = 2;
    private $states = [
        self::STATE_GREEN => [
            'to_yellow' => self::STATE_YELLOW,
        ],
        self::STATE_YELLOW => [
            'to_green' => self::STATE_GREEN,
            'to_red' => self::STATE_RED,
        ],
        self::STATE_RED => [
            'to_yellow' => self::STATE_YELLOW,
        ],
    ];

    public function can(TrafficLight $trafficLight, $transition) {
        $state = $trafficLight->getCurrentState();

        return isset($this->states[$state][$transition]);
    }

    public function apply(TrafficLight $trafficLight, $transition) {
        if (!$this->can($trafficLight, $transition)) {
            throw new \InvalidArgumentException('Invalid transition');
        }

        $state = $trafficLight->getCurrentState();
        $newState = $this->states[$state][$transition];
        $trafficLight->setState($newState);
    }
}
```



```
class StateMachine{
```

@tobiasnyholm

```
    private $states;
```

```
    public function __construct(array $states) {  
        $this->states = $states;  
    }
```

```
    public function can(TrafficLight $trafficLight, $transition) {  
        $state = $trafficLight->getCurrentState();  
  
        return isset($this->states[$state][$transition]);  
    }
```

```
    public function apply(TrafficLight $trafficLight, $transition) {  
        if (!$this->can($trafficLight, $transition)) {  
            throw new \InvalidArgumentException('Invalid transition');  
        }  
  
        $state = $trafficLight->getCurrentState();  
        $newState = $this->states[$state][$transition];  
        $trafficLight->setState($newState);  
    }
```

```
    // ...
```

```
}
```



```
class StateMachine{
```

@tobiasnyholm

```
    private $states;
```

```
    public function __construct(array $states) {  
        $this->states = $states;  
    }
```

```
    public function can(StateAwareInterface $object, $transition) {  
        $state = $object->getCurrentState();  
  
        return isset($this->states[$state][$transition]);  
    }
```

```
    public function apply(StateAwareInterface $object, $transition) {  
        if (!$this->can($object, $transition)) {  
            throw new \InvalidArgumentException('Invalid transition');  
        }  
  
        $state = $object->getCurrentState();  
        $newState = $this->states[$state][$transition];  
        $object->setState($newState);  
    }
```

```
    // ...
```

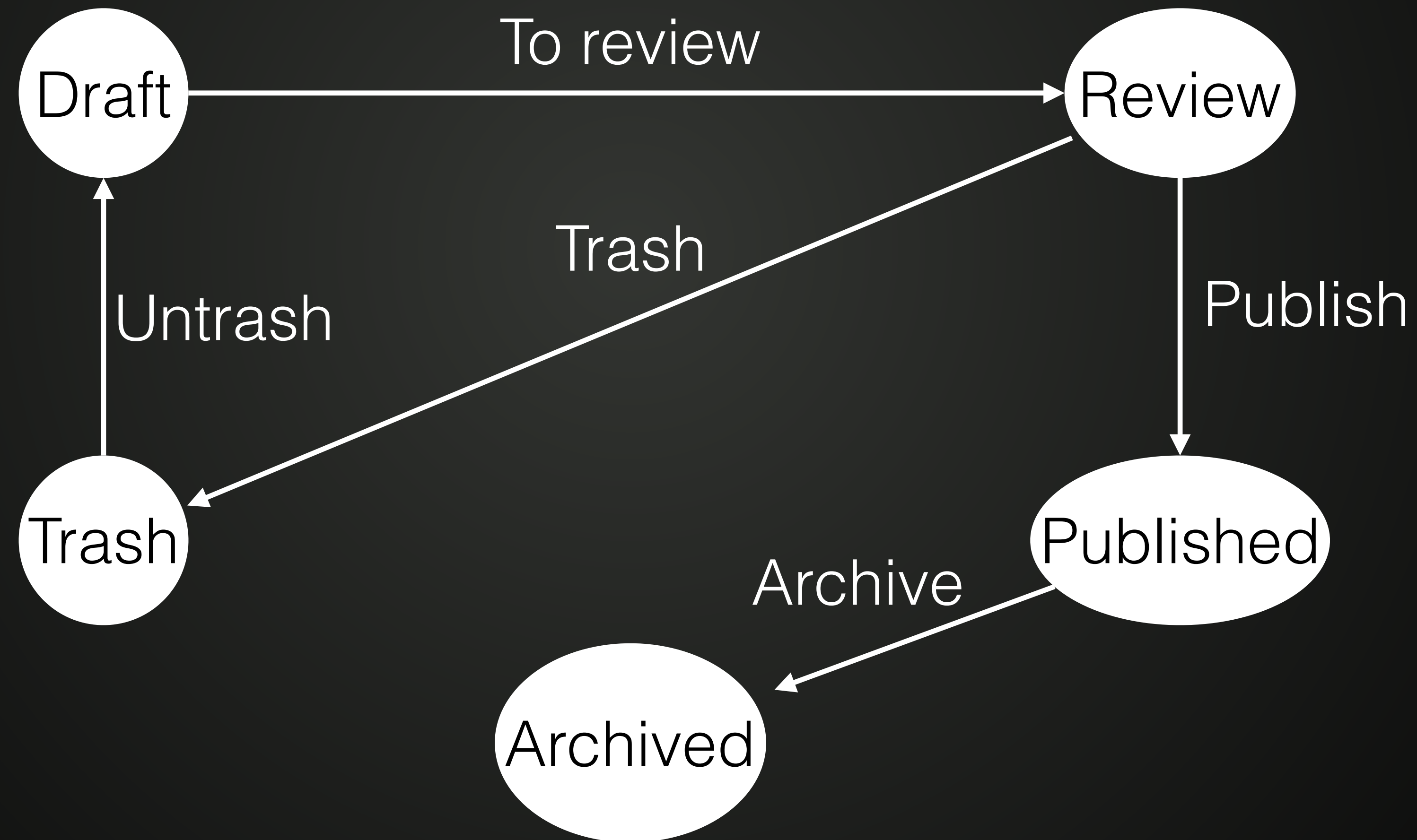
```
}
```

Workflow component

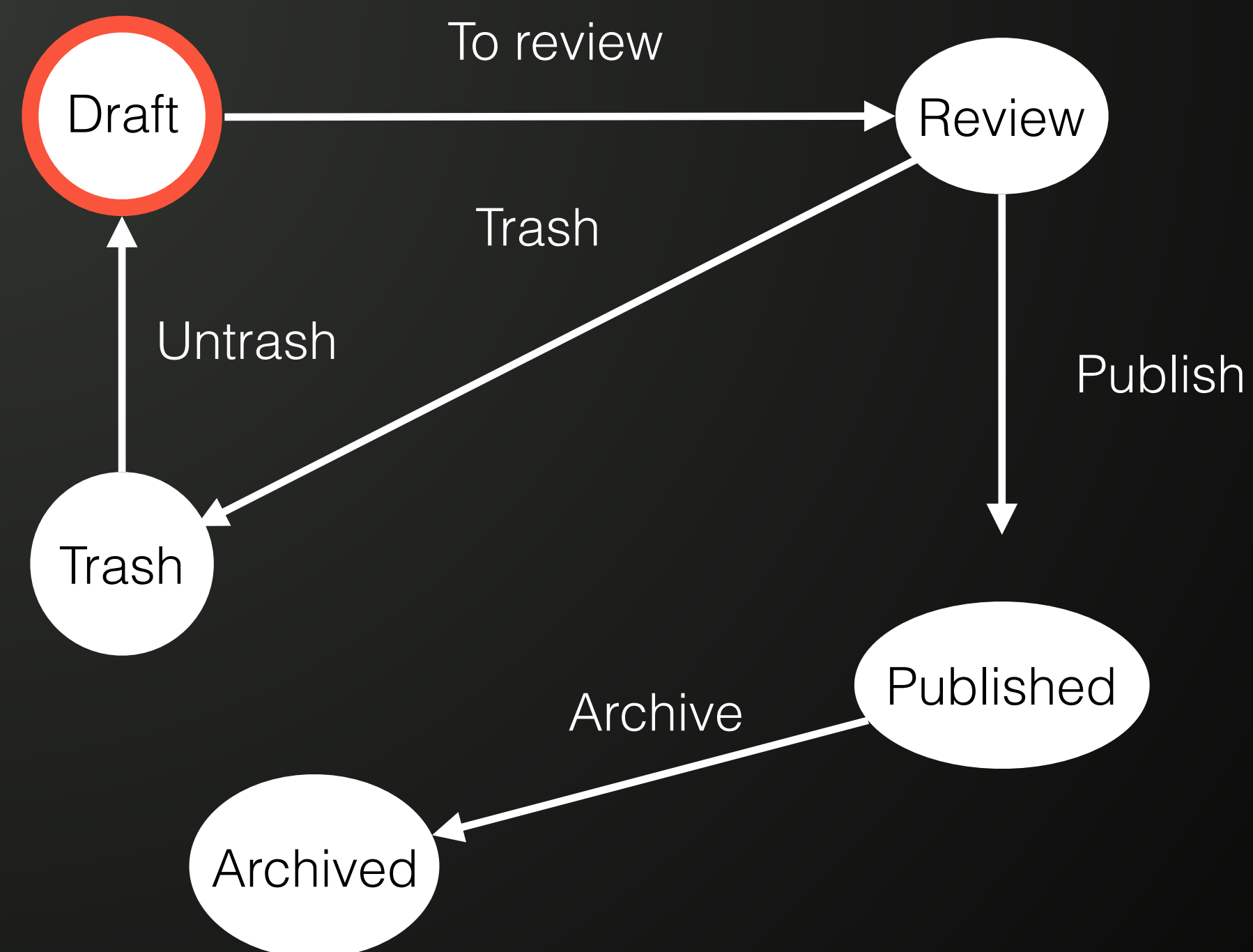
Functions

- Can we do [transition]?
- Apply [transition]
- What state are we in?
- Which are my valid transitions?

Job advert



Examples

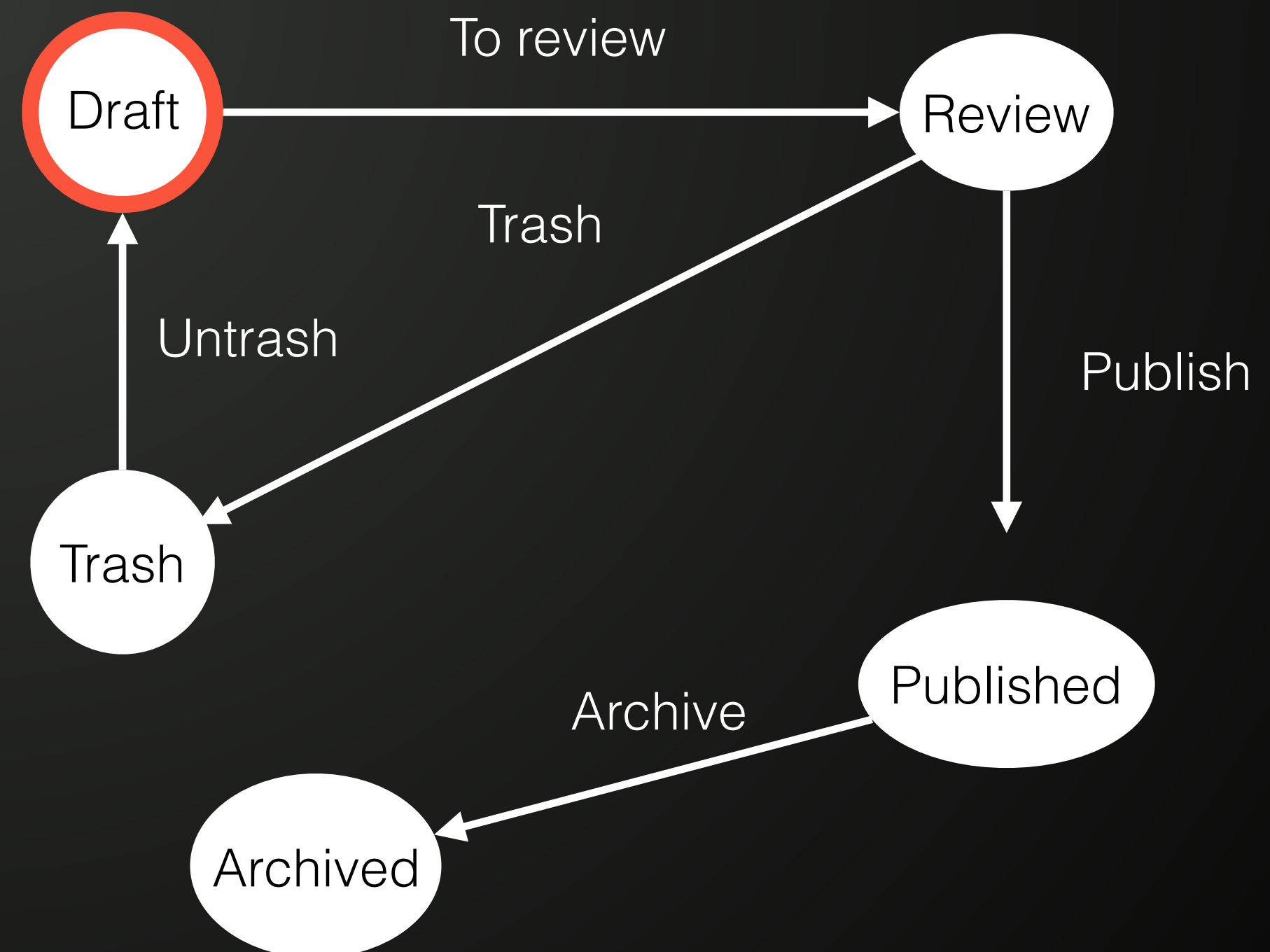


Examples

```
$stateMachine = $this->get('state_machine.job_advert');  
$places = $stateMachine->getMarking($advert)->getPlaces();  
var_dump($places);
```

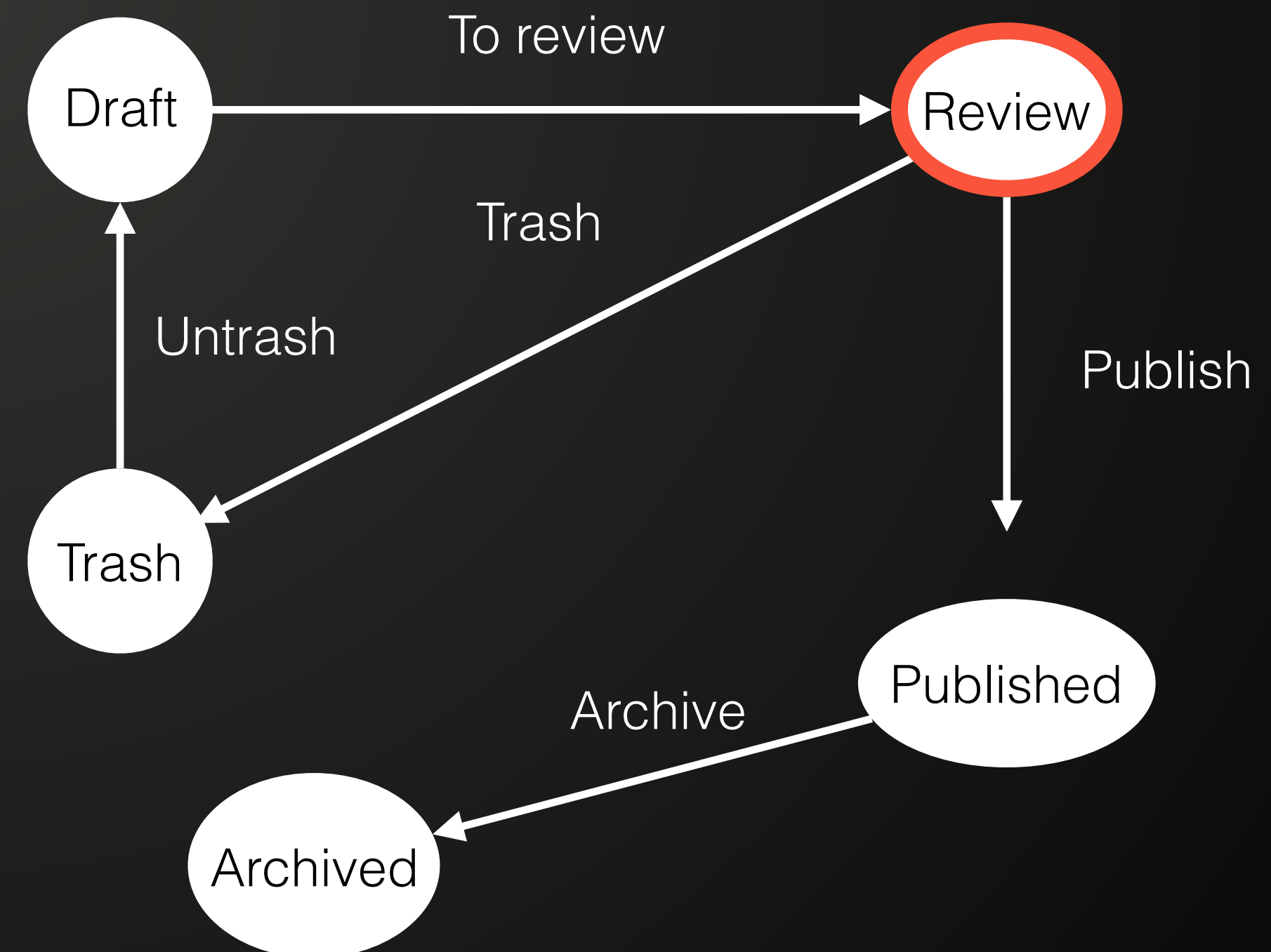
```
// array (size=1)  
// 'draft' => int 1
```

```
$stateMachine->can($advert, 'publish'); // false  
$stateMachine->can($advert, 'to_review'); // true  
$stateMachine->apply($advert, 'to_review');
```



Examples

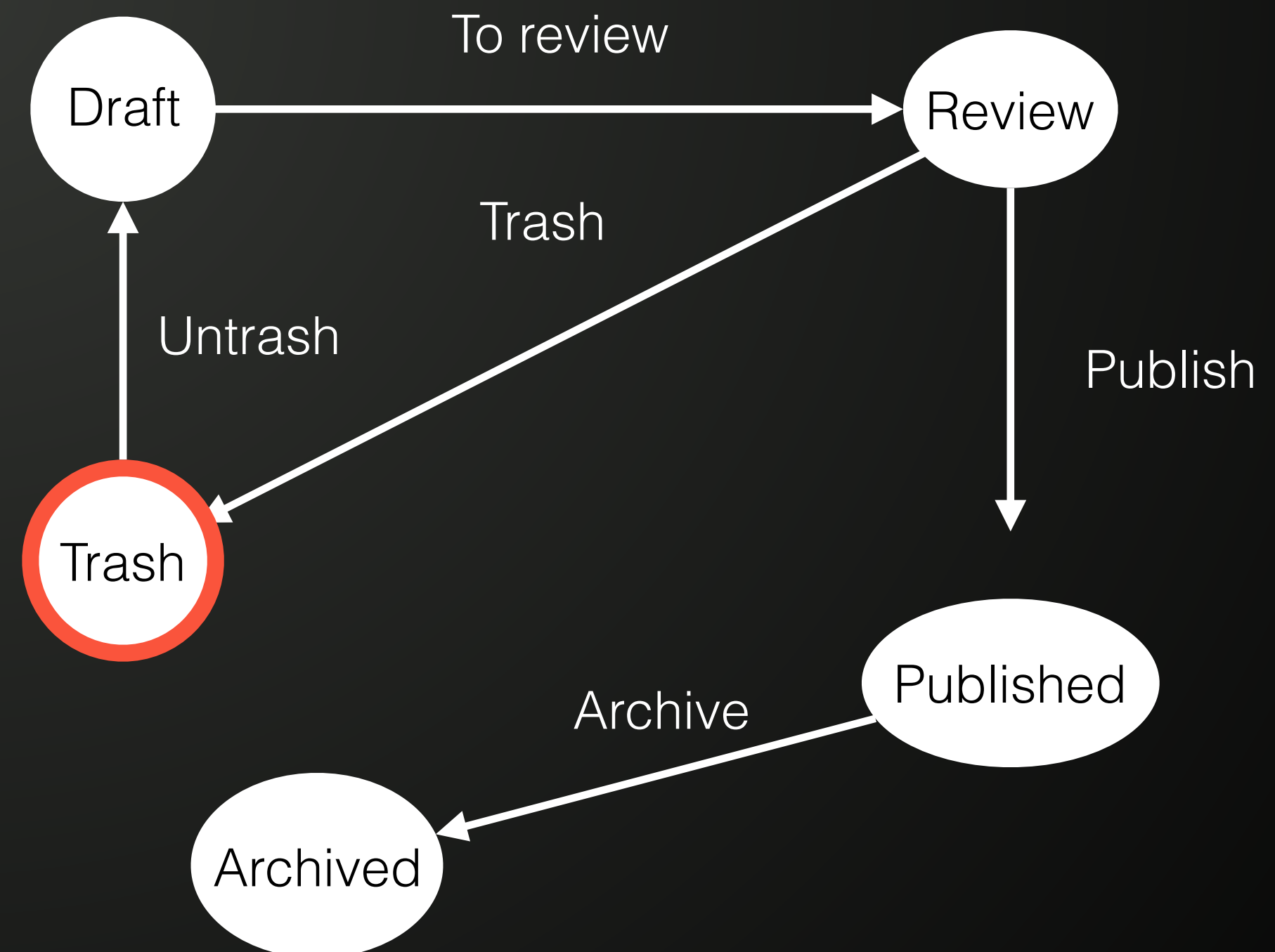
```
$stateMachine->can($advert, 'publish'); // false  
$stateMachine->can($advert, 'to_review'); // true  
  
$stateMachine->apply($advert, 'to_review');
```



Examples

```
$stateMachine->can($advert, 'publish'); // false  
$stateMachine->can($advert, 'to_review'); // true
```

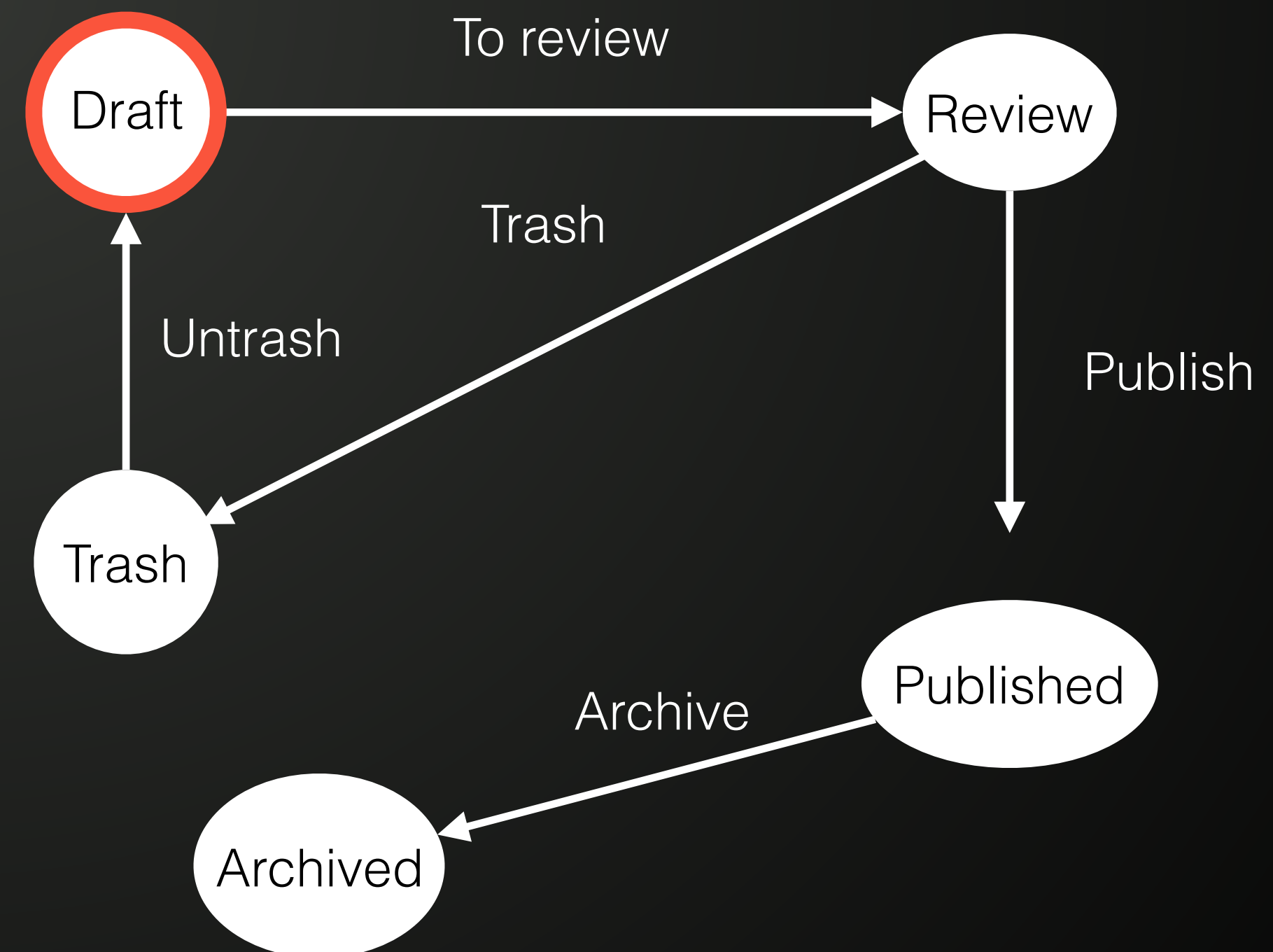
```
$stateMachine->apply($advert, 'to_review');  
$stateMachine->apply($advert, 'trash');
```



Examples

```
$stateMachine->can($advert, 'publish'); // false  
$stateMachine->can($advert, 'to_review'); // true
```

```
$stateMachine->apply($advert, 'to_review');  
$stateMachine->apply($advert, 'trash');  
$stateMachine->apply($advert, 'untrash');
```



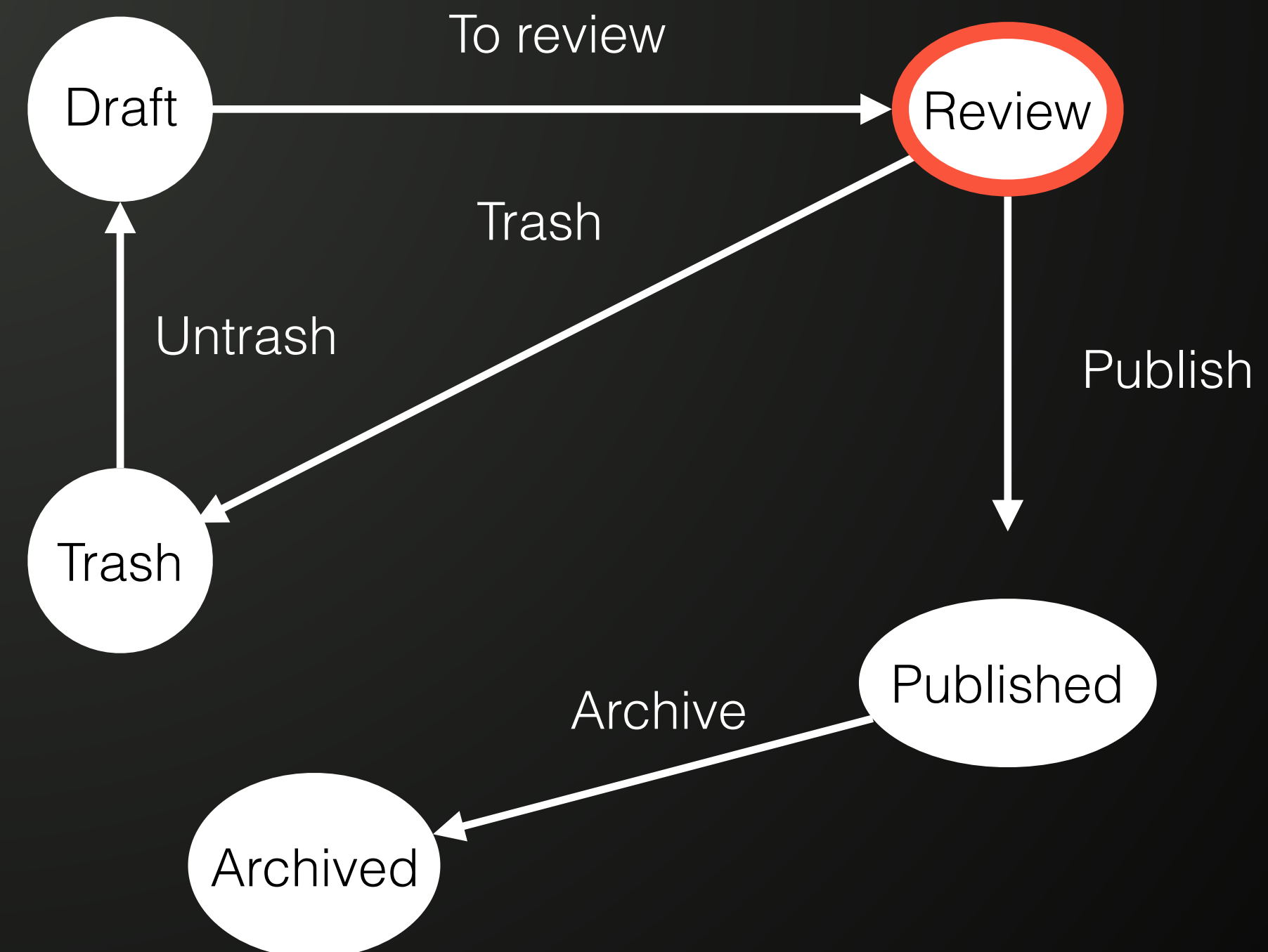
Examples

```
$stateMachine->can($advert, 'publish'); // false  
$stateMachine->can($advert, 'to_review'); // true
```

```
$stateMachine->apply($advert, 'to_review');  
$stateMachine->apply($advert, 'trash');  
$stateMachine->apply($advert, 'untrash');  
$stateMachine->apply($advert, 'to_review');
```

```
$transitions = $stateMachine->getEnabledTransitions($advert);  
foreach ($transitions as $transition) {  
    echo $transition->getName();  
}
```

```
// publish  
// trash
```



Twig examples

```
<h4>Sidebar actions</h4>
{% if workflow_can(advert, 'to_review') %}
    <a href="{{ path('advert_to_review', {id:advert.id}) }}">
        Send to review</a><br>
{% endif %}

{% if workflow_can(advert, 'publish') %}
    <a href="{{ path('advert_publish', {id:advert.id}) }}">
        Publish advert</a><br>
{% endif %}

{% if workflow_can(advert, 'archive') %}
    <a href="{{ path('advert_archive', {id:advert.id}) }}">
        Archive</a><br>
{% endif %}

{% if workflow_can(advert, 'trash') %}
    <a href="{{ path('advert_trash', {id:advert.id}) }}">
        Trash</a><br>
{% endif %}

{% if workflow_can(advert, 'untrash') %}
    <a href="{{ path('advert_untrash', {id:advert.id}) }}">
        Restore for trash</a><br>
{% endif %}
```


Twig examples

Sidebar actions

Send to review

Sidebar actions

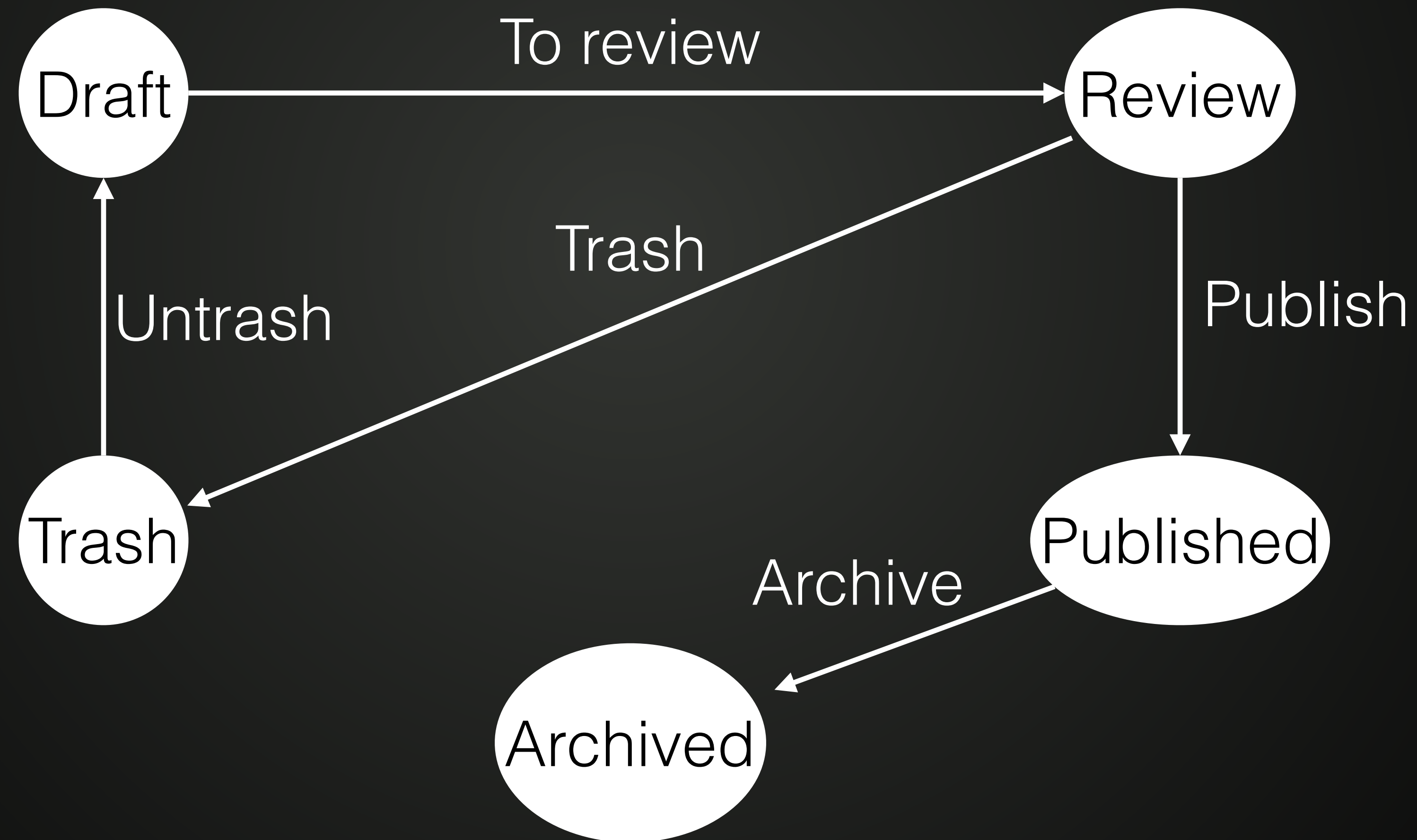
Publish advert

Trash

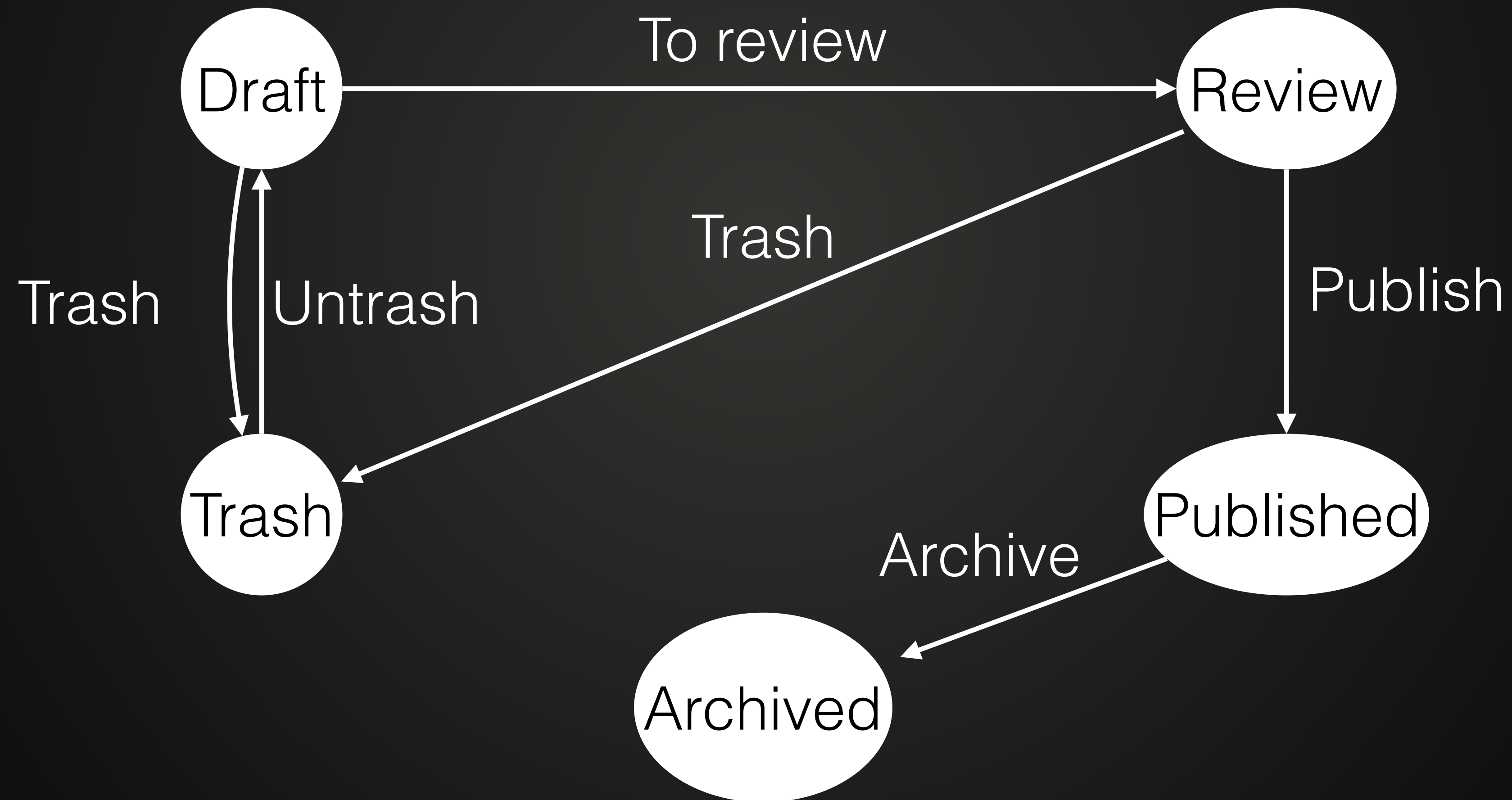
Twig examples

```
<h4>Sidebar actions</h4>
{% for transition in workflow_transitions(advert) %}
    <a href="{{ path('advert_~transition.name', {id:advert.id}) }}">
        {{ transition.name|humanize }}
    </a><br />
{% else %}
    No actions available for this advert.
{% endfor %}
```

Job advert



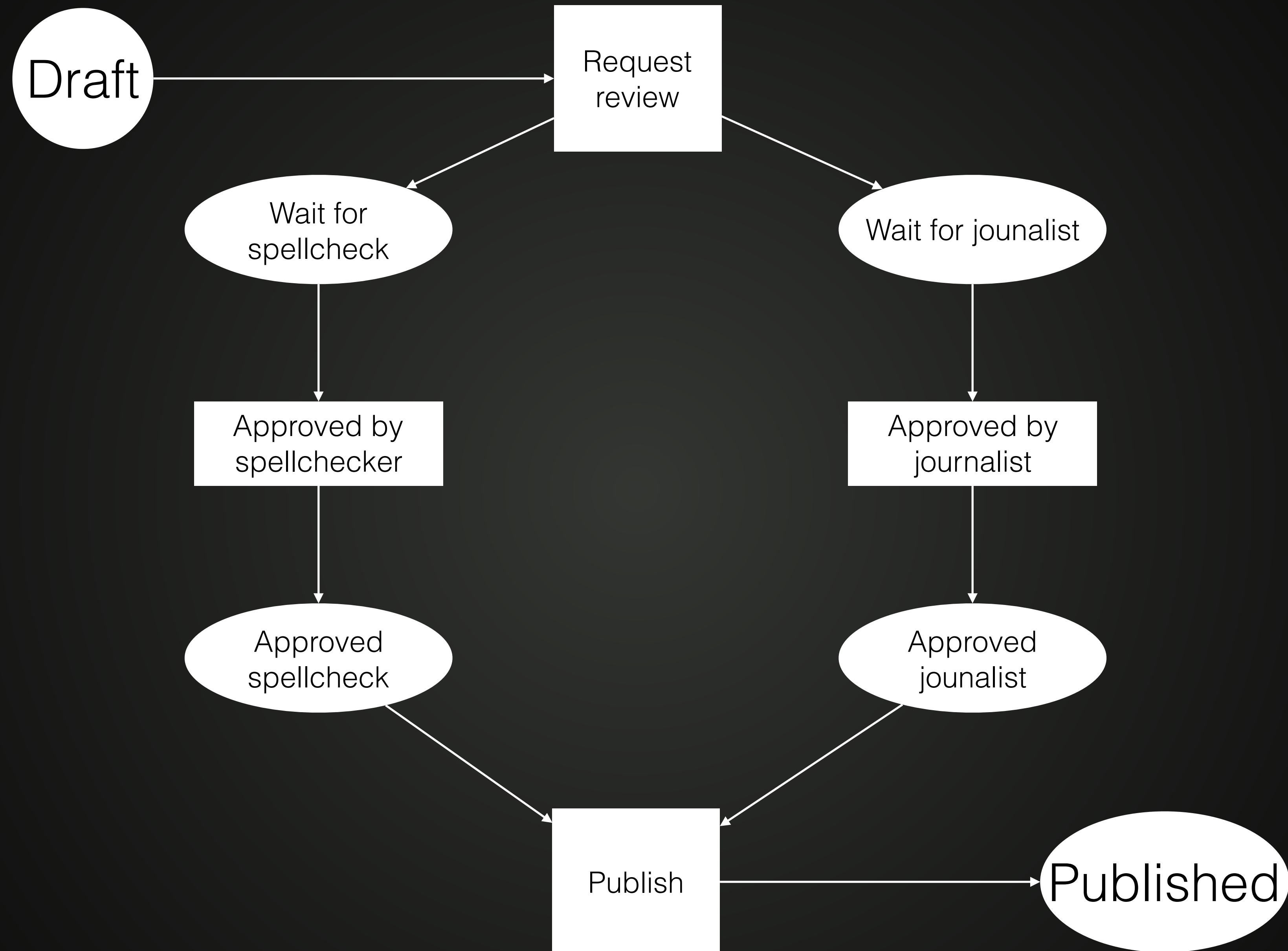
Job advert

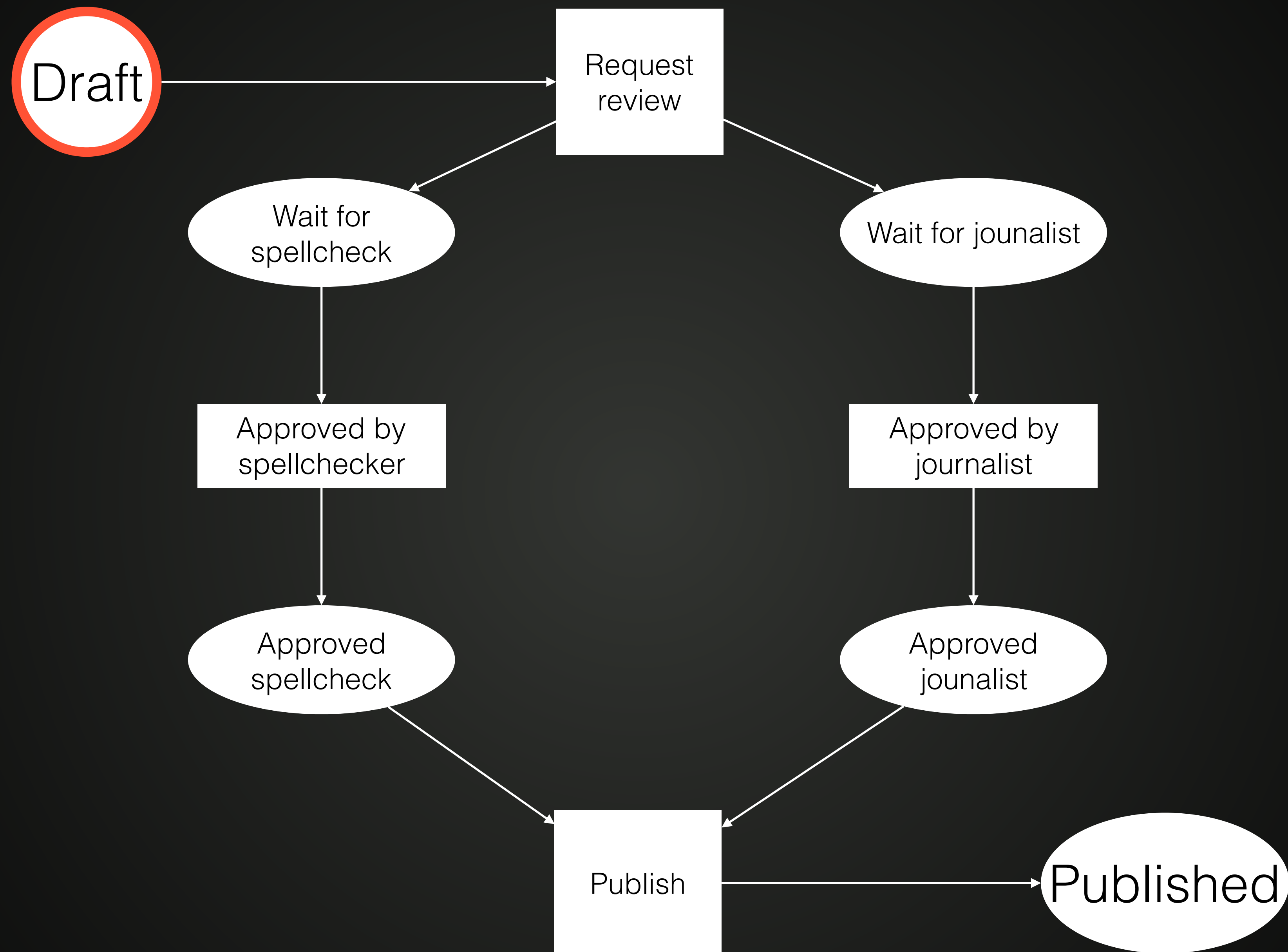


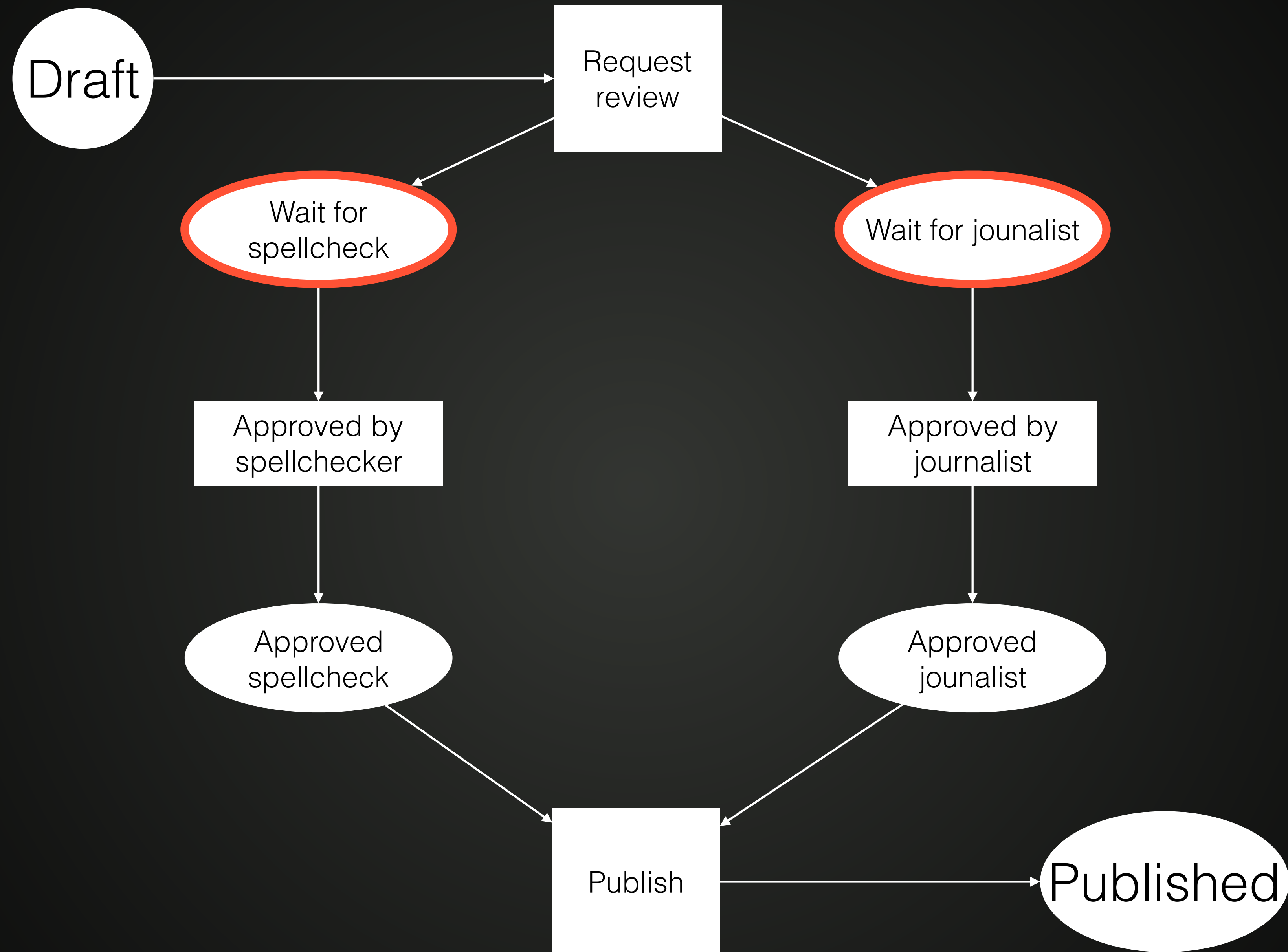
Configuration

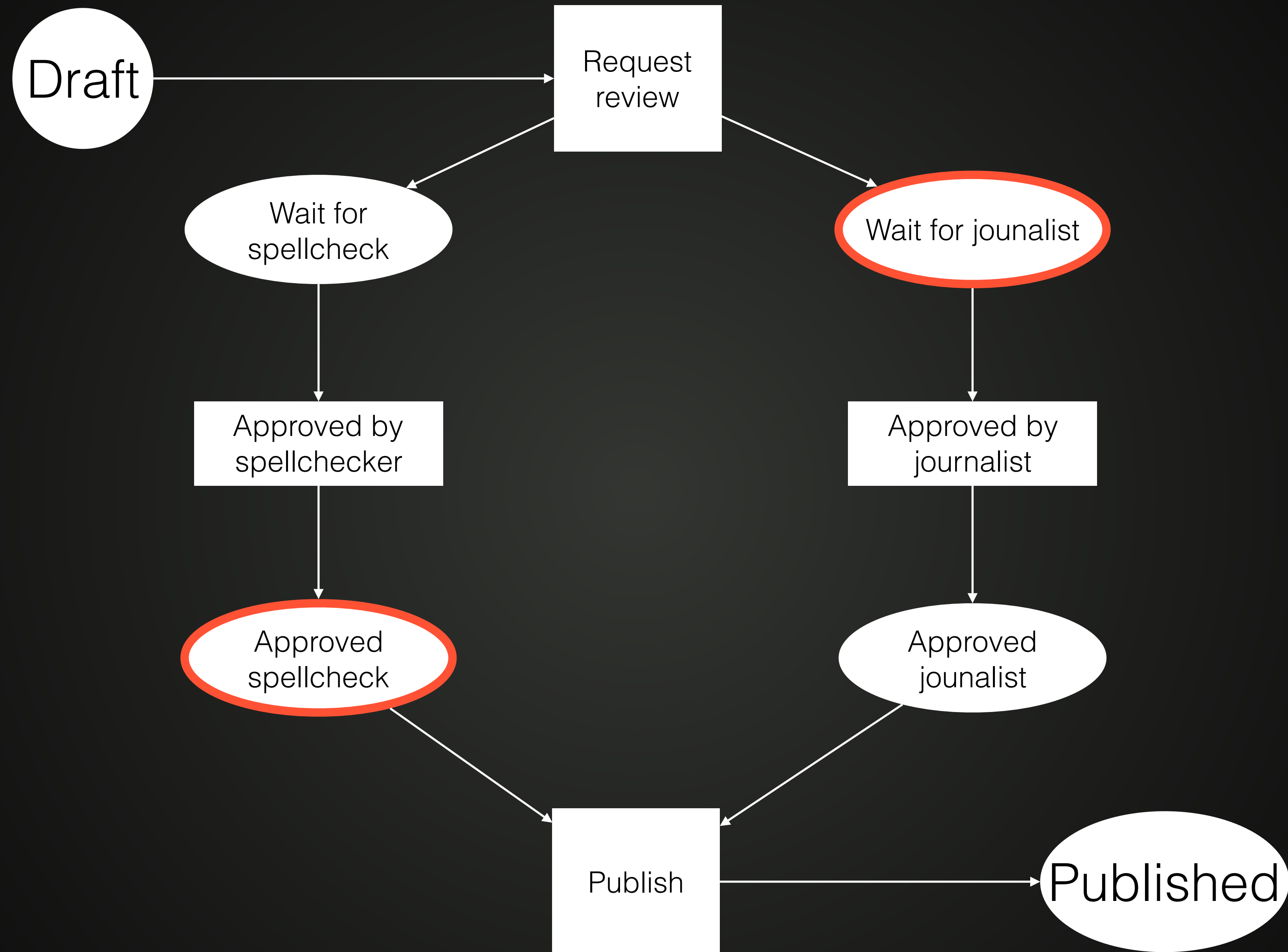
```
framework:
  workflows:
    job_advert:
      type: 'state_machine'
      supports:
        - AppBundle\Entity\Advert
      places:
        - draft
        - review
        - published
        - trash
        - archived
      transitions:
        to_review:
          from: draft
          to: review
        publish:
          from: review
          to: published
        trash:
          from: [draft, review]
          to: trash
        archive:
          from: published
          to: archived
        untrash:
          from: trash
          to: draft
```

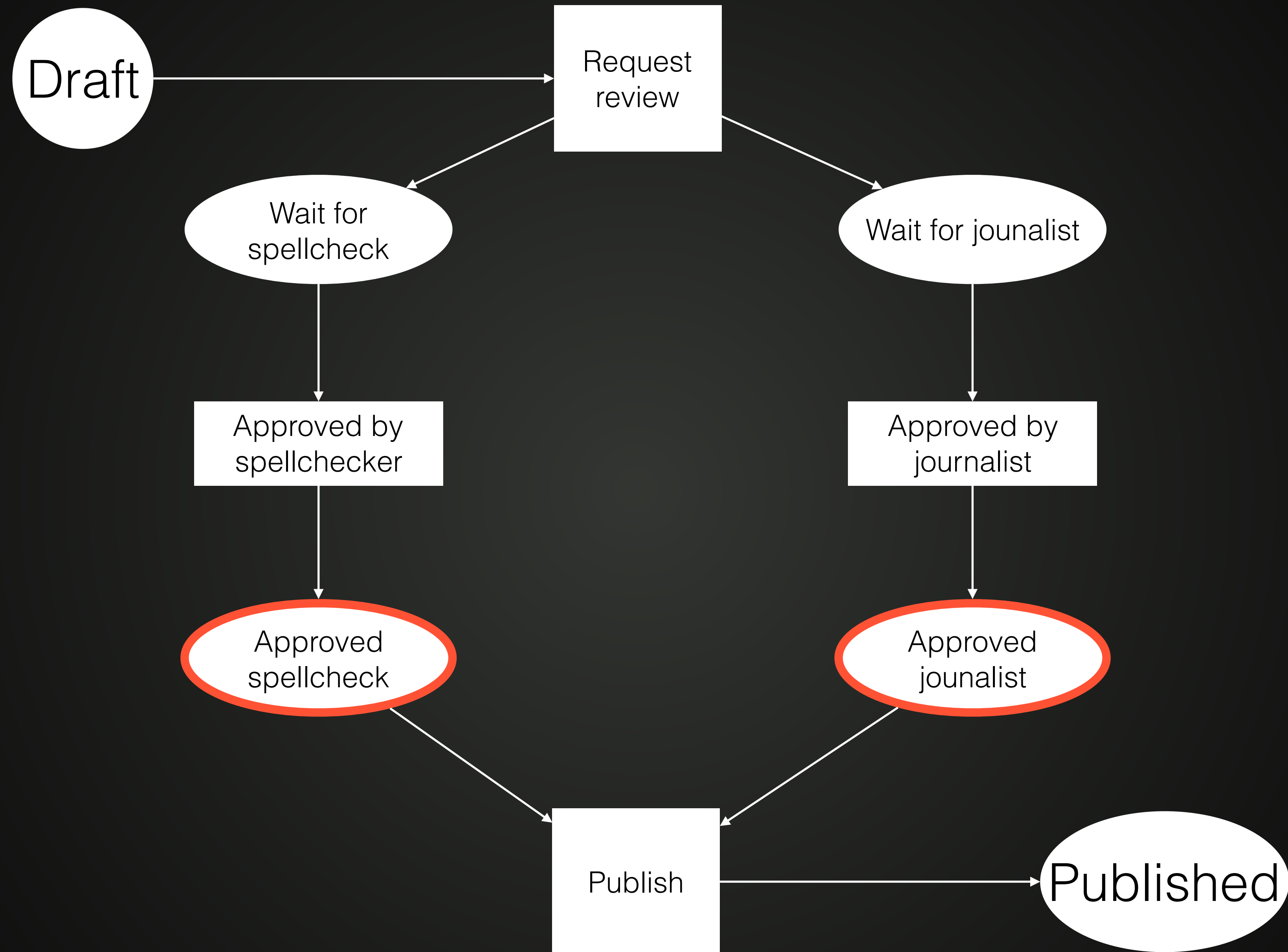


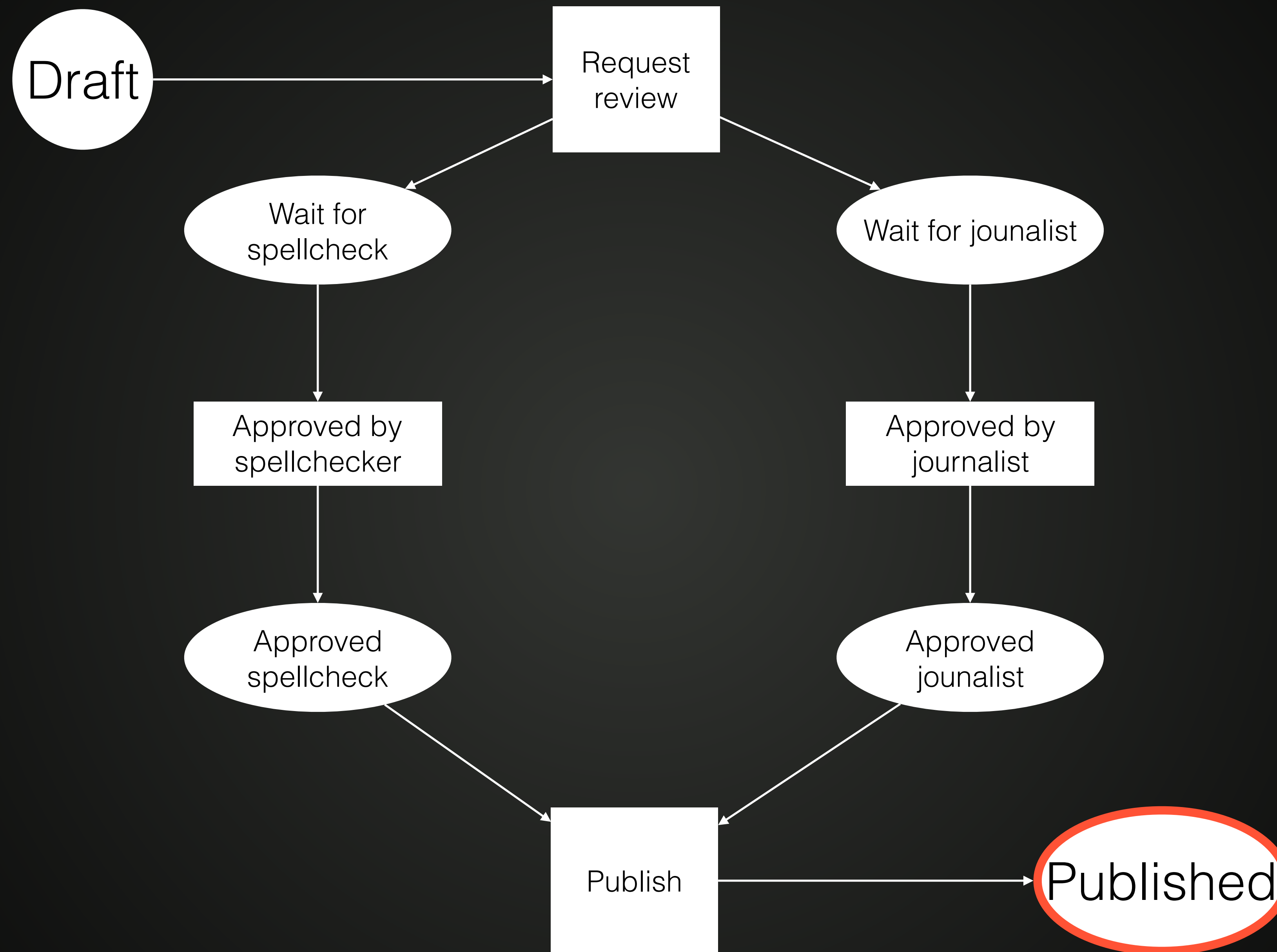












Events

workflow.leave

workflow.[*job_advert*].leave

workflow.[*job_advert*].leave.[*draft*]

workflow.transition

workflow.[*job_advert*].transition

workflow.[*job_advert*].transition.[*to_review*]

workflow.enter

workflow.entered

workflow.completed

workflow.announce

Guard

```
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\Security\Core\Authorization\AuthorizationCheckerInterface;
use Symfony\Component\Workflow\Event\GuardEvent;

class GuardListener implements EventSubscriberInterface
{
    public function __construct(AuthorizationCheckerInterface $checker)
    {
        $this->checker = $checker;
    }

    public function guardPublish(GuardEvent $event)
    {
        if (!$this->checker->isGranted('ROLE_PUBLISHER')) {
            $event->setBlocked(true);
        }
    }

    public static function getSubscribedEvents()
    {
        return array(
            'workflow.job_advert.guard.publish' => 'guardPublish',
        );
    }
}
```



```
use Psr\Log\LoggerInterface;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\Workflow\Event\Event;

class WorkflowLogger implements EventSubscriberInterface
{
    public function __construct(LoggerInterface $logger)
    {
        $this->logger = $logger;
    }

    public function onLeave(Event $event)
    {
        $this->logger->alert(sprintf(
            'Advert (id: "%s") preformed transition "%s" from "%s" to "%s"',
            $event->getSubject()->getId(),
            $event->getTransition()->getName(),
            implode(' ', array_keys($event->getMarking()->getPlaces())),
            implode(' ', $event->getTransition()->getTos())
        ));
    }

    public static function getSubscribedEvents()
    {
        return array(
            'workflow.job_advert.leave' => 'onLeave',
        );
    }
}
```

Security voters

```
class AdvertVoter extends Voter
{
  const VIEW = 'VIEW';
  const EDIT = 'EDIT';
  // ...
  protected function supports($attribute, $subject)
  {
    $attribute = strtoupper($attribute);
    return $subject instanceof Advert &&
      in_array($attribute, [self::VIEW, self::EDIT]);
  }

  protected function voteOnAttribute($attribute, $advert, TokenInterface $token)
  {
    $attribute = strtoupper($attribute);
    $this->currentToken = $token;
    switch ($attribute) {
      case self::VIEW:
        return $this->stateMachine->getMarking($advert)->has('published');
      case self::EDIT:
        // ...
    }

    return false;
  }
}
```

Events

```
use AppBundle\Service\EmailService;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\Workflow\Event\Event;

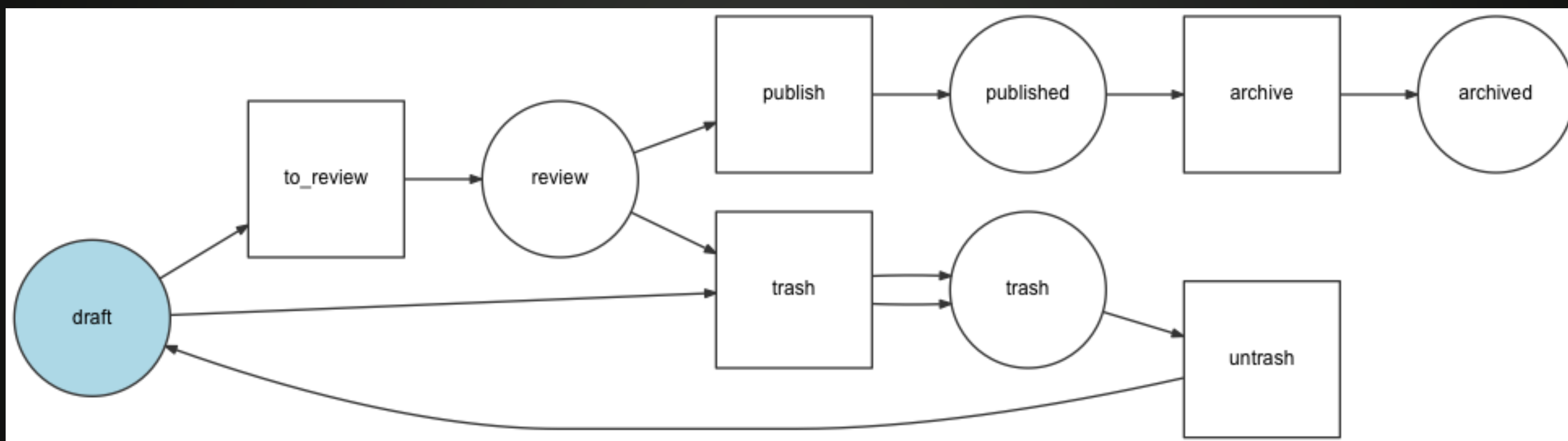
class PublishListener implements EventSubscriberInterface
{
    public function __construct(EmailService $mailer)
    {
        $this->mailer = $mailer;
    }

    public function onPublished(Event $event)
    {
        $this->mailer->emailAllUsers($event->getSubject());
    }

    public static function getSubscribedEvents()
    {
        return array(
            'workflow.job_advert.enter.published' => 'onPublished',
        );
    }
}
```


Debug

```
php bin/console workflow:dump job_advert | dot -Tpng -o out.png
```





Identify state machines

```
class Quotation
{
    const STATUS_SENT = 0;
    const STATUS_NEGOTIATION = 1;
    const STATUS_WON = 2;
    const STATUS_LOST = 3;

    /**
     * @var int status
     *
     * @ORM\Column(type="integer")
     * @Assert\NotBlank()
     */
    private $status;

    // ...
}
```


| Search: id IN 62,63,99,585 | | | | | | | |
|----------------------------|-------------|--------|-------|--------|---------------------|---|--|
| id | position_id | active | ended | closed | createdAt | u | |
| 62 | 119 → | 0 | 1 | 1 | 2014-07-28 16:01:55 | 2 | |
| 63 | 120 → | 0 | 1 | 1 | 2014-07-28 16:02:17 | 2 | |
| 99 | 204 → | 0 | 1 | 1 | 2014-08-06 13:42:27 | 2 | |
| 585 | 597 → | 0 | 1 | 1 | 2015-03-19 09:51:49 | 2 | |
| 586 | 386 → | 0 | 1 | 1 | 2015-03-19 11:41:27 | 2 | |
| 587 | 389 → | 0 | 1 | 1 | 2015-03-19 11:42:22 | 2 | |
| 614 | 82 → | 0 | 1 | 1 | 2015-04-03 07:44:26 | 2 | |
| 620 | 134 → | 0 | 1 | 1 | 2015-04-04 13:36:20 | 2 | |
| 880 | 1006 → | 0 | 1 | 1 | 2015-07-03 14:47:19 | 2 | |
| 4343 | 8968 → | 0 | 1 | 1 | 2016-06-22 09:28:50 | 2 | |
| 4344 | 23740 → | 0 | 1 | 1 | 2016-06-22 09:35:33 | 2 | |

a way of thinking

<https://joind.in/talk/cb240>