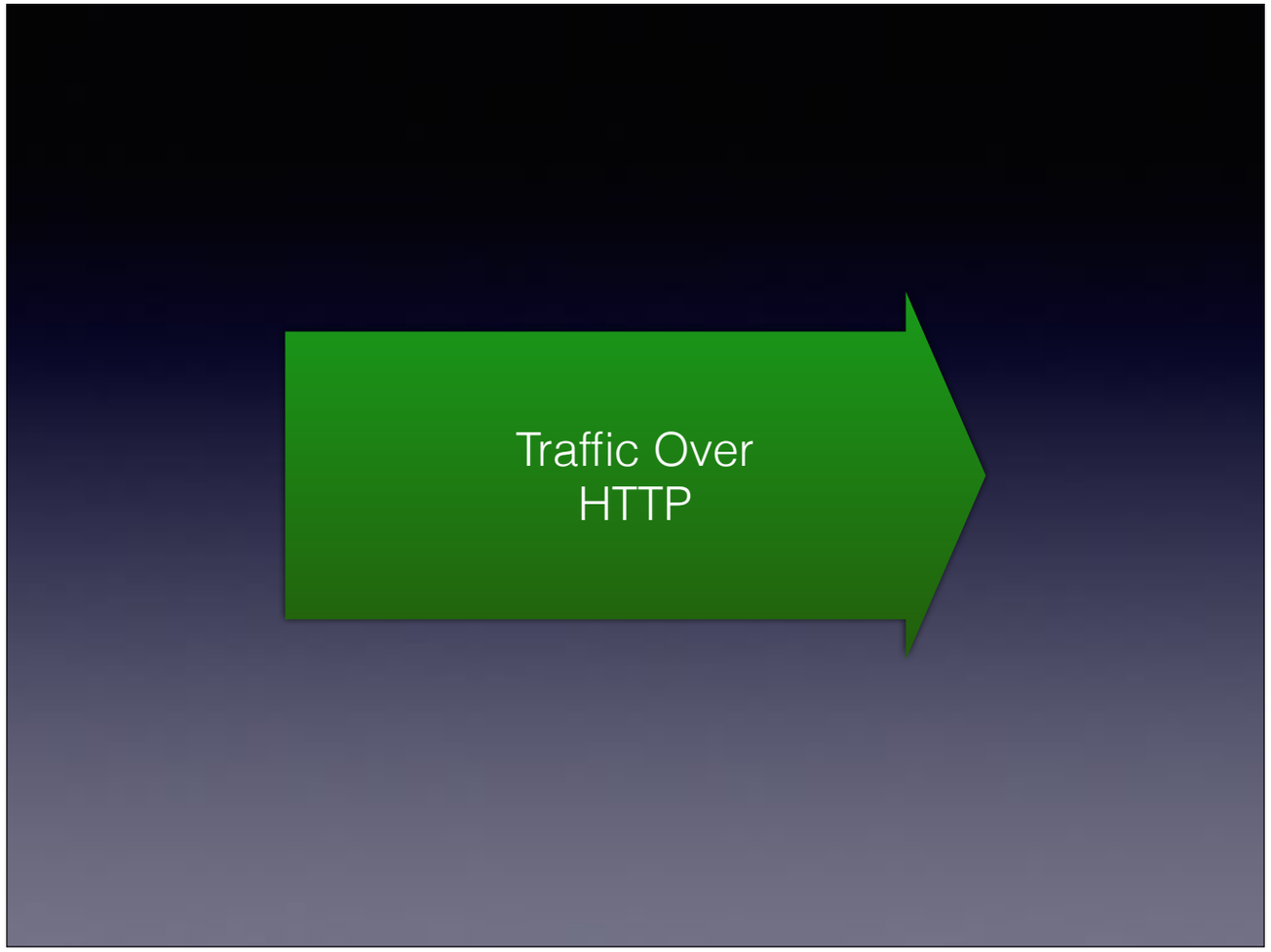
My name is Jason Goodell, I'm a developer with Global Knowledge and my expertise is in server side development in Python. Unfortunately Sam Boyarsky is unable to join us today at Open edX. This talk is going to cover the integration of Open edX into an enterprise systems, specifically the Global Knowledge enterprise environment. Instead of talking in detail about development specifics, this talk will concentrate on architecture and design of the extensions to Open edX created to integrate Open edX. This talk is also a review of Global Knowledge's integration approach, warts and all. After deciding on this topic, and making the decision to not delve deeply into implementation specifics I realized there is still a lot of content to cover. So lets get started.

Extending Open edX to Support Integration with Enterprise Systems

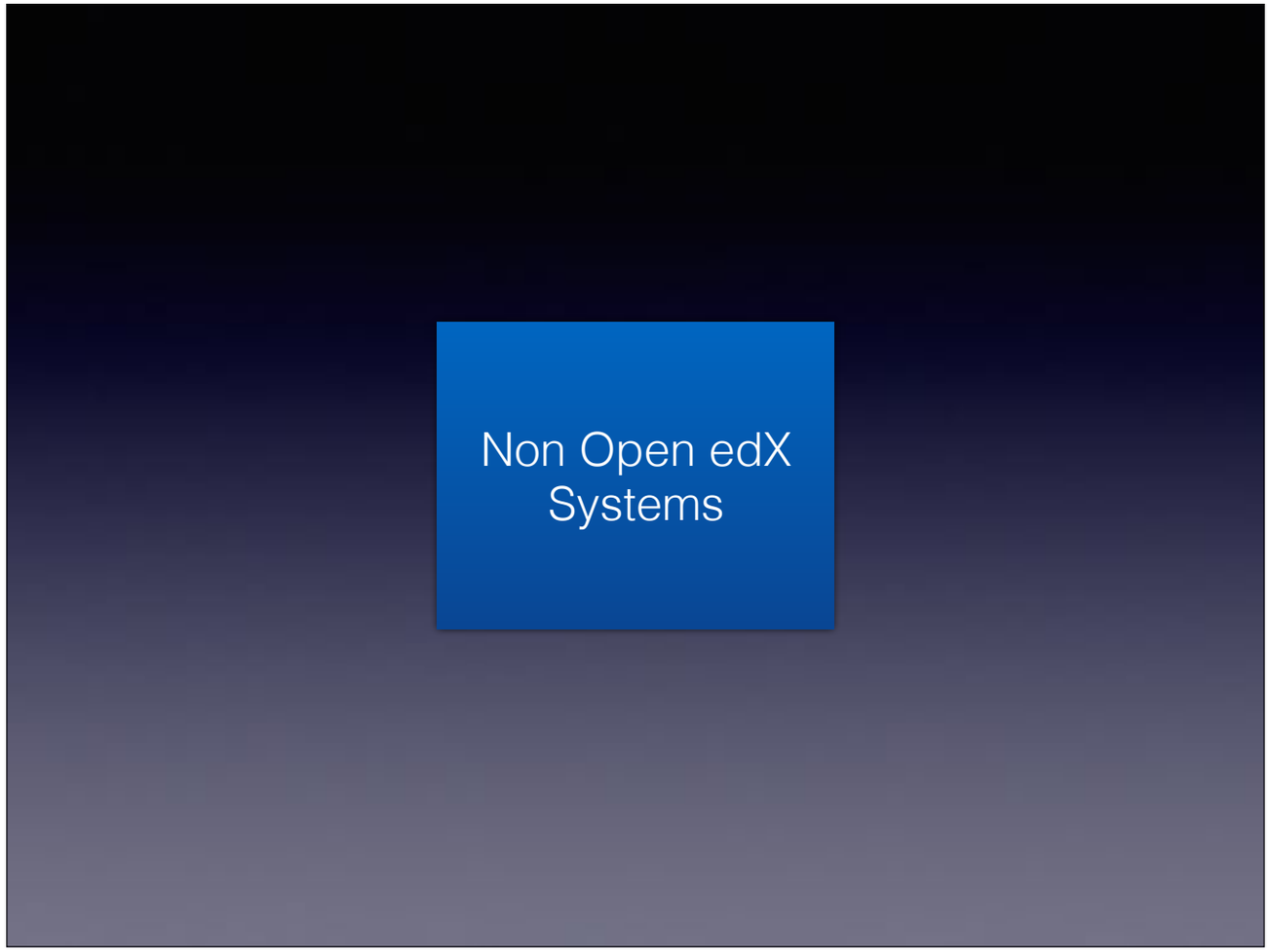
Jason Goodell
Software Engineer
Global Knowledge Training LLC.

My name is Jason Goodell, I'm a developer with Global Knowledge and my expertise is in server side development in Python. Unfortunately Sam Boyarsky is unable to join us today at Open edX. This talk is going to cover the integration of Open edX into an enterprise systems, specifically the Global Knowledge enterprise environment. Instead of talking in detail about development specifics, this talk will concentrate on architecture and design of the extensions to Open edX created to integrate Open edX. This talk is also a review of Global Knowledge's integration approach, warts and all. After deciding on this topic, and making the decision to not delve deeply into implementation specifics I realized there is still a lot of content to cover. So lets get started.

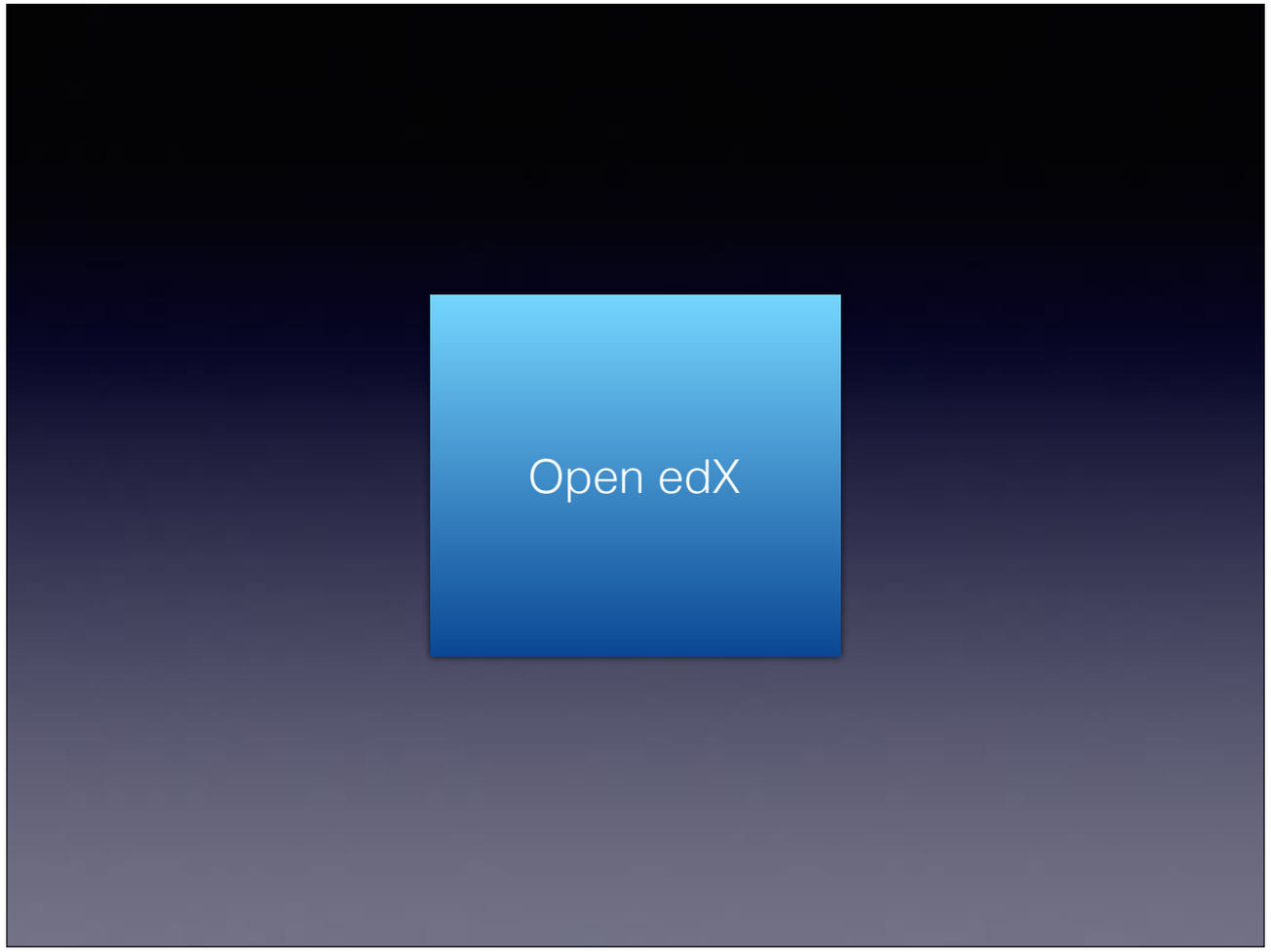
First I'd like to take a moment to talk about the graphics used in this presentation. So that we all know what we are looking at. [click] Green objects are used to indicate requests and responses. [click] Solid blue objects are used to indicate peripheral systems to Open edX in the Global Knowledge environment. [click] Blue fade objects are used to indicate Open edX instances. [click] Red objects are used to indicate data systems. [click] Turquoise objects are used to indicate additions to Open edX such as Django apps and APIs. [click] Finally, grey objects are miscellaneous other entities in the Global Knowledge environment.



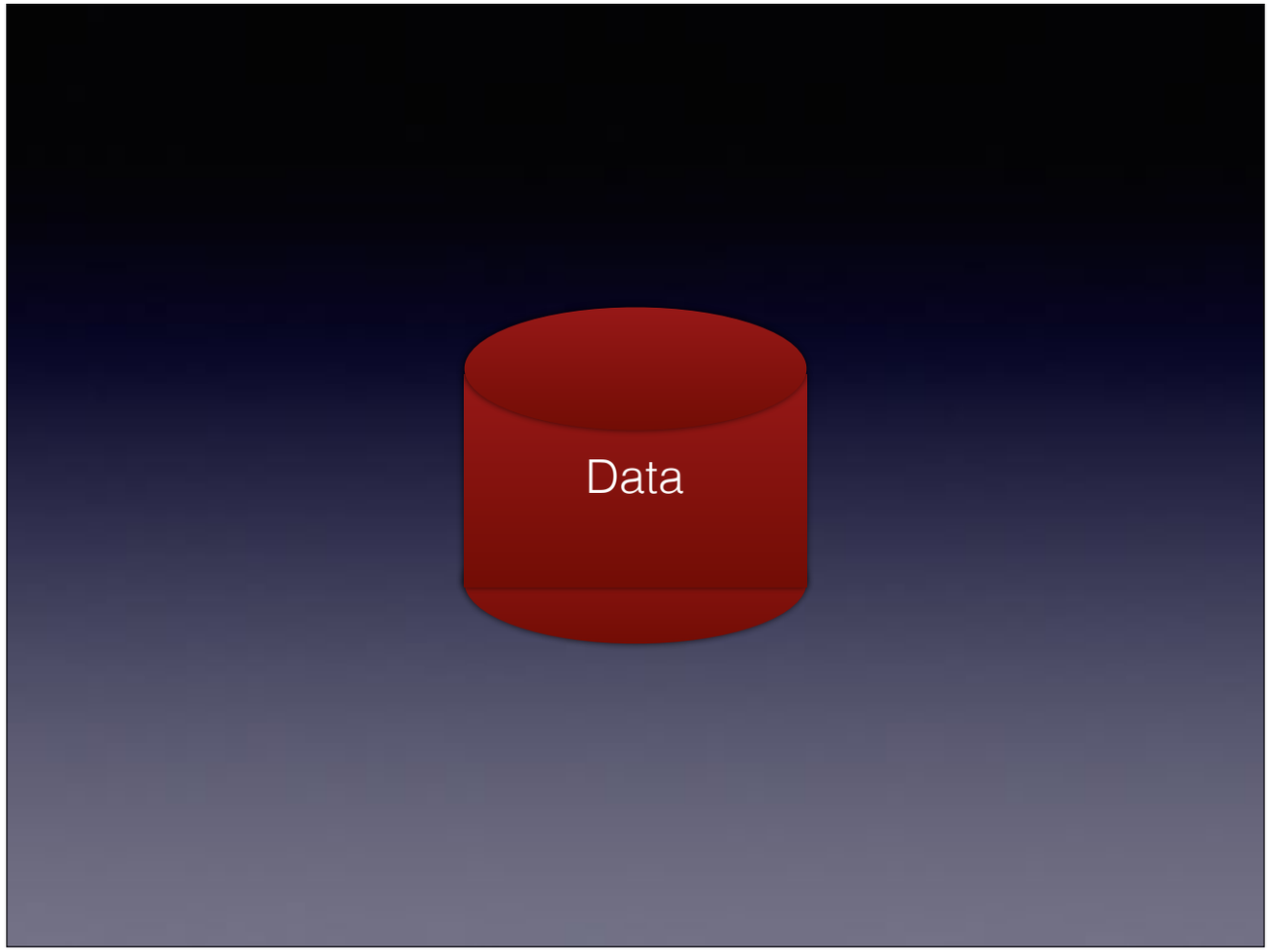
First I'd like to take a moment to talk about the graphics used in this presentation. So that we all know what we are looking at. [click] Green objects are used to indicate requests and responses. [click] Solid blue objects are used to indicate peripheral systems to Open edX in the Global Knowledge environment. [click] Blue fade objects are used to indicate Open edX instances. [click] Red objects are used to indicate data systems. [click] Turquoise objects are used to indicate additions to Open edX such as Django apps and APIs. [click] Finally, grey objects are miscellaneous other entities in the Global Knowledge environment.



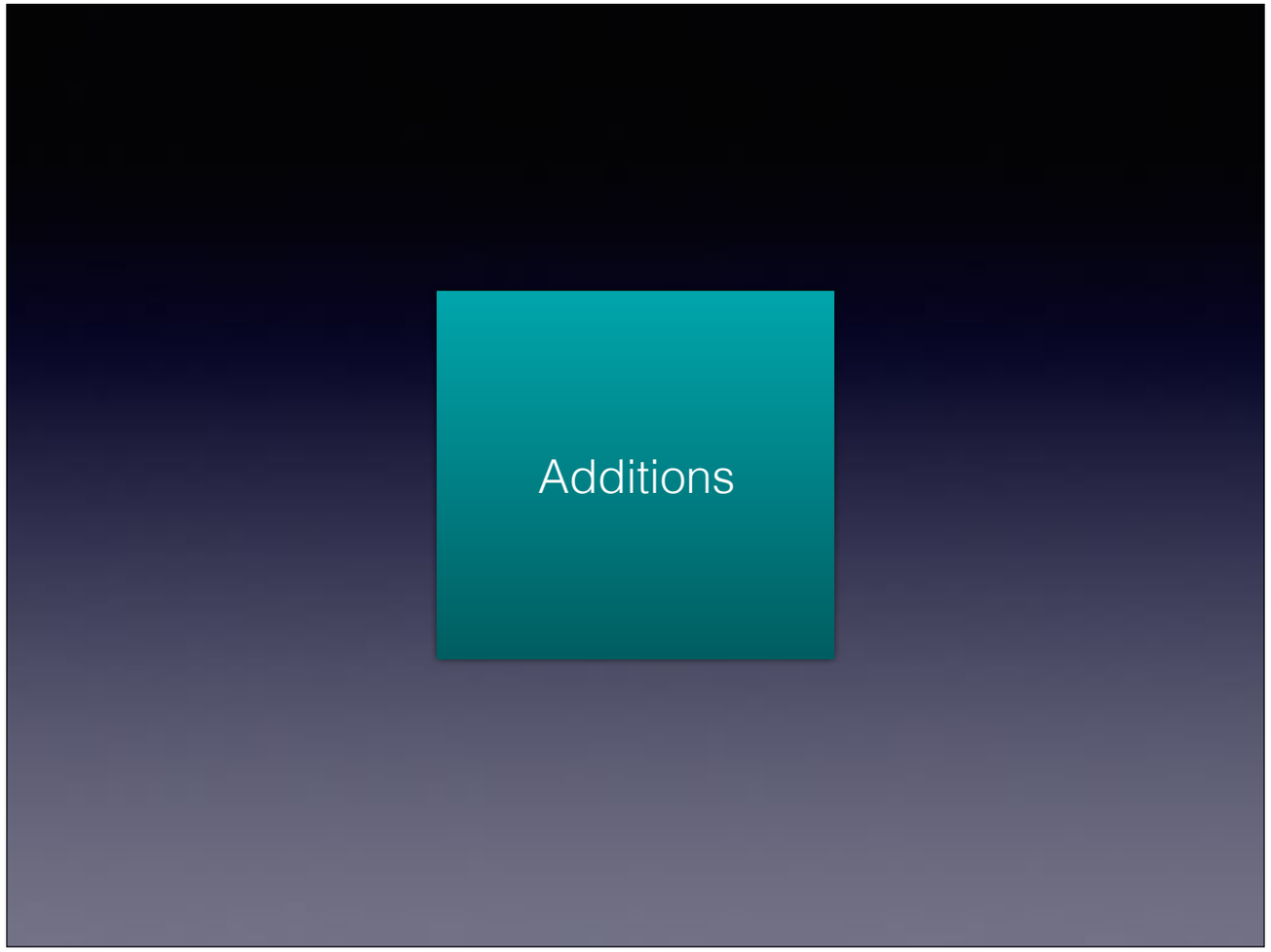
First I'd like to take a moment to talk about the graphics used in this presentation. So that we all know what we are looking at. [click] Green objects are used to indicate requests and responses. [click] Solid blue objects are used to indicate peripheral systems to Open edX in the Global Knowledge environment. [click] Blue fade objects are used to indicate Open edX instances. [click] Red objects are used to indicate data systems. [click] Turquoise objects are used to indicate additions to Open edX such as Django apps and APIs. [click] Finally, grey objects are miscellaneous other entities in the Global Knowledge environment.



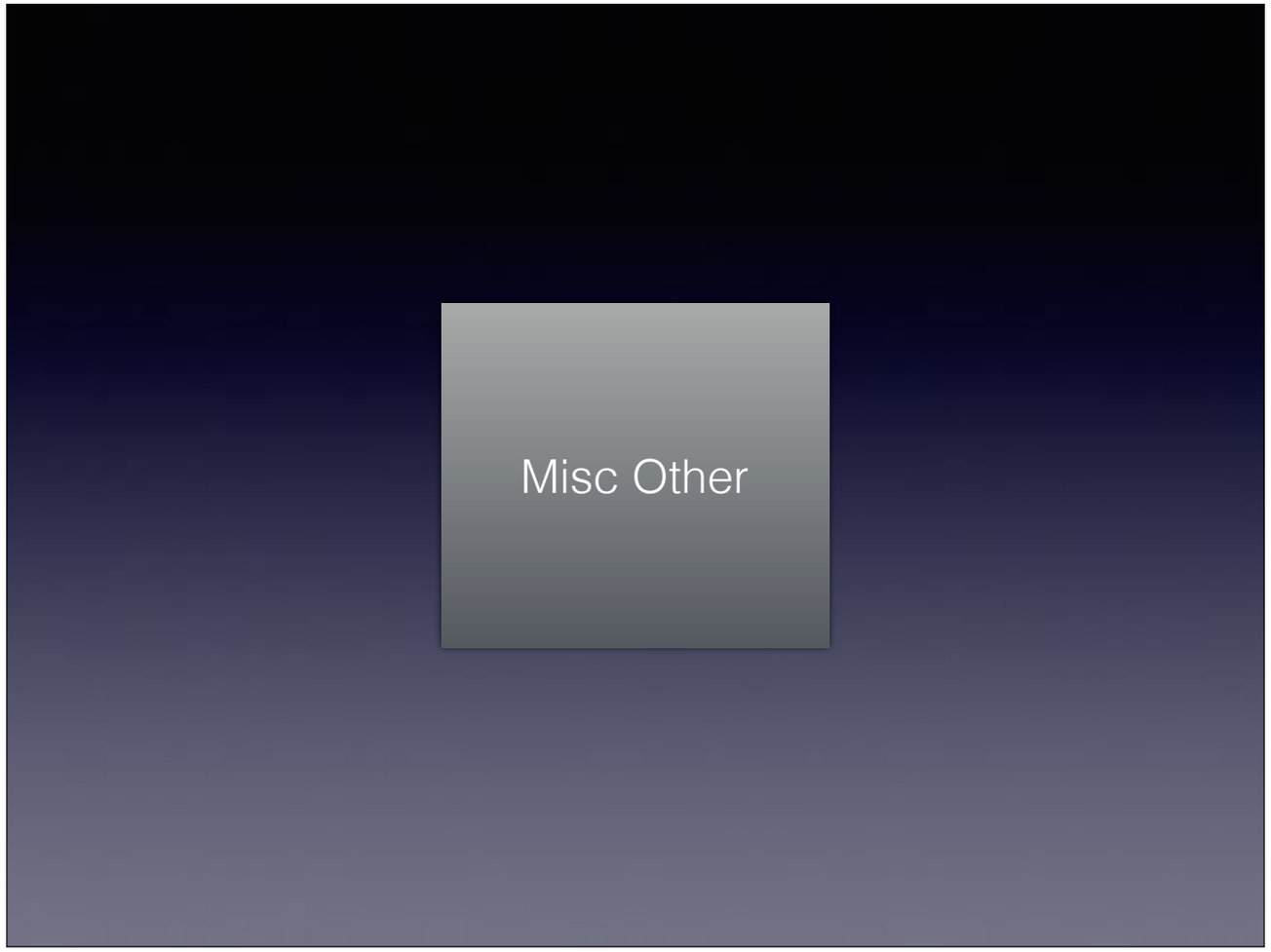
First I'd like to take a moment to talk about the graphics used in this presentation. So that we all know what we are looking at. [click] Green objects are used to indicate requests and responses. [click] Solid blue objects are used to indicate peripheral systems to Open edX in the Global Knowledge environment. [click] Blue fade objects are used to indicate Open edX instances. [click] Red objects are used to indicate data systems. [click] Turquoise objects are used to indicate additions to Open edX such as Django apps and APIs. [click] Finally, grey objects are miscellaneous other entities in the Global Knowledge environment.



First I'd like to take a moment to talk about the graphics used in this presentation. So that we all know what we are looking at. [click] Green objects are used to indicate requests and responses. [click] Solid blue objects are used to indicate peripheral systems to Open edX in the Global Knowledge environment. [click] Blue fade objects are used to indicate Open edX instances. [click] Red objects are used to indicate data systems. [click] Turquoise objects are used to indicate additions to Open edX such as Django apps and APIs. [click] Finally, grey objects are miscellaneous other entities in the Global Knowledge environment.



First I'd like to take a moment to talk about the graphics used in this presentation. So that we all know what we are looking at. [click] Green objects are used to indicate requests and responses. [click] Solid blue objects are used to indicate peripheral systems to Open edX in the Global Knowledge environment. [click] Blue fade objects are used to indicate Open edX instances. [click] Red objects are used to indicate data systems. [click] Turquoise objects are used to indicate additions to Open edX such as Django apps and APIs. [click] Finally, grey objects are miscellaneous other entities in the Global Knowledge environment.



First I'd like to take a moment to talk about the graphics used in this presentation. So that we all know what we are looking at. [click] Green objects are used to indicate requests and responses. [click] Solid blue objects are used to indicate peripheral systems to Open edX in the Global Knowledge environment. [click] Blue fade objects are used to indicate Open edX instances. [click] Red objects are used to indicate data systems. [click] Turquoise objects are used to indicate additions to Open edX such as Django apps and APIs. [click] Finally, grey objects are miscellaneous other entities in the Global Knowledge environment.

First I'd like to take a moment to talk about the graphics used in this presentation. So that we all know what we are looking at. [click] Green objects are used to indicate requests and responses. [click] Solid blue objects are used to indicate peripheral systems to Open edX in the Global Knowledge environment. [click] Blue fade objects are used to indicate Open edX instances. [click] Red objects are used to indicate data systems. [click] Turquoise objects are used to indicate additions to Open edX such as Django apps and APIs. [click] Finally, grey objects are miscellaneous other entities in the Global Knowledge environment.

To appreciate the approach to integrating Open edX at Global Knowledge, there are key characteristics to our approach to understand. [click] Open edX is a delegate system to Global Knowledge's existing back office processes and systems. [click] User and enrollment records migrate to Open edX from Global Knowledge's back office systems. For Global Knowledge user's and enrollments will not be created in Open edX. Instead these records will be created in Global Knowledge's back office systems. [click] Integrations are designed to mitigating risk from changes to the Open edX platform. We create additional functionality by adding Django apps as plugins. With our plugins depending on Open edX core code instead of modifying Open edX. [click] Global Knowledge records are pushed to Open edX through custom web APIs, and placed on Open edX's relational database management system in a set of landing tables. Global Knowledge's existing environment of back office systems, which manage purchasing, student account management and course enrollment are the systems of truth for Global Knowledge's business. These back office systems not only support self-paced courses within Open edX, but also support Global Knowledge's other course modalities; instructor lead and virtual instructor lead.

Characteristics

To appreciate the approach to integrating Open edX at Global Knowledge, there are key characteristics to our approach to understand. [click] Open edX is a delegate system to Global Knowledge's existing back office processes and systems. [click] User and enrollment records migrate to Open edX from Global Knowledge's back office systems. For Global Knowledge user's and enrollments will not be created in Open edX. Instead these records will be created in Global Knowledge's back office systems. [click] Integrations are designed to mitigating risk from changes to the Open edX platform. We create additional functionality by adding Django apps as plugins. With our plugins depending on Open edX core code instead of modifying Open edX. [click] Global Knowledge records are pushed to Open edX through custom web APIs, and placed on Open edX's relational database management system in a set of landing tables. Global Knowledge's existing environment of back office systems, which manage purchasing, student account management and course enrollment are the systems of truth for Global Knowledge's business. These back office systems not only support self-paced courses within Open edX, but also support Global Knowledge's other course modalities; instructor lead and virtual instructor lead.

Characteristics

- Open edX will be a delegate system to existing back office process' and systems.

To appreciate the approach to integrating Open edX at Global Knowledge, there are key characteristics to our approach to understand. [click] Open edX is a delegate system to Global Knowledge's existing back office processes and systems. [click] User and enrollment records migrate to Open edX from Global Knowledge's back office systems. For Global Knowledge user's and enrollments will not be created in Open edX. Instead these records will be created in Global Knowledge's back office systems. [click] Integrations are designed to mitigating risk from changes to the Open edX platform. We create additional functionality by adding Django apps as plugins. With our plugins depending on Open edX core code instead of modifying Open edX. [click] Global Knowledge records are pushed to Open edX through custom web APIs, and placed on Open edX's relational database management system in a set of landing tables. Global Knowledge's existing environment of back office systems, which manage purchasing, student account management and course enrollment are the systems of truth for Global Knowledge's business. These back office systems not only support self-paced courses within Open edX, but also support Global Knowledge's other course modalities; instructor lead and virtual instructor lead.

Characteristics

- Open edX will be a delegate system to existing back office process' and systems.
  - User Records and Enrollment records will migrate to Open edX from the back office.

To appreciate the approach to integrating Open edX at Global Knowledge, there are key characteristics to our approach to understand. [click] Open edX is a delegate system to Global Knowledge's existing back office processes and systems. [click] User and enrollment records migrate to Open edX from Global Knowledge's back office systems. For Global Knowledge user's and enrollments will not be created in Open edX. Instead these records will be created in Global Knowledge's back office systems. [click] Integrations are designed to mitigating risk from changes to the Open edX platform. We create additional functionality by adding Django apps as plugins. With our plugins depending on Open edX core code instead of modifying Open edX. [click] Global Knowledge records are pushed to Open edX through custom web APIs, and placed on Open edX's relational database management system in a set of landing tables. Global Knowledge's existing environment of back office systems, which manage purchasing, student account management and course enrollment are the systems of truth for Global Knowledge's business. These back office systems not only support self-paced courses within Open edX, but also support Global Knowledge's other course modalities; instructor lead and virtual instructor lead.

Characteristics

- Open edX will be a delegate system to existing back office process' and systems.
  - User Records and Enrollment records will migrate to Open edX from the back office.
- Mitigate risk from changes to the Open edX platform.

To appreciate the approach to integrating Open edX at Global Knowledge, there are key characteristics to our approach to understand. [click] Open edX is a delegate system to Global Knowledge's existing back office processes and systems. [click] User and enrollment records migrate to Open edX from Global Knowledge's back office systems. For Global Knowledge user's and enrollments will not be created in Open edX. Instead these records will be created in Global Knowledge's back office systems. [click] Integrations are designed to mitigating risk from changes to the Open edX platform. We create additional functionality by adding Django apps as plugins. With our plugins depending on Open edX core code instead of modifying Open edX. [click] Global Knowledge records are pushed to Open edX through custom web APIs, and placed on Open edX's relational database management system in a set of landing tables. Global Knowledge's existing environment of back office systems, which manage purchasing, student account management and course enrollment are the systems of truth for Global Knowledge's business. These back office systems not only support self-paced courses within Open edX, but also support Global Knowledge's other course modalities; instructor lead and virtual instructor lead.

Characteristics

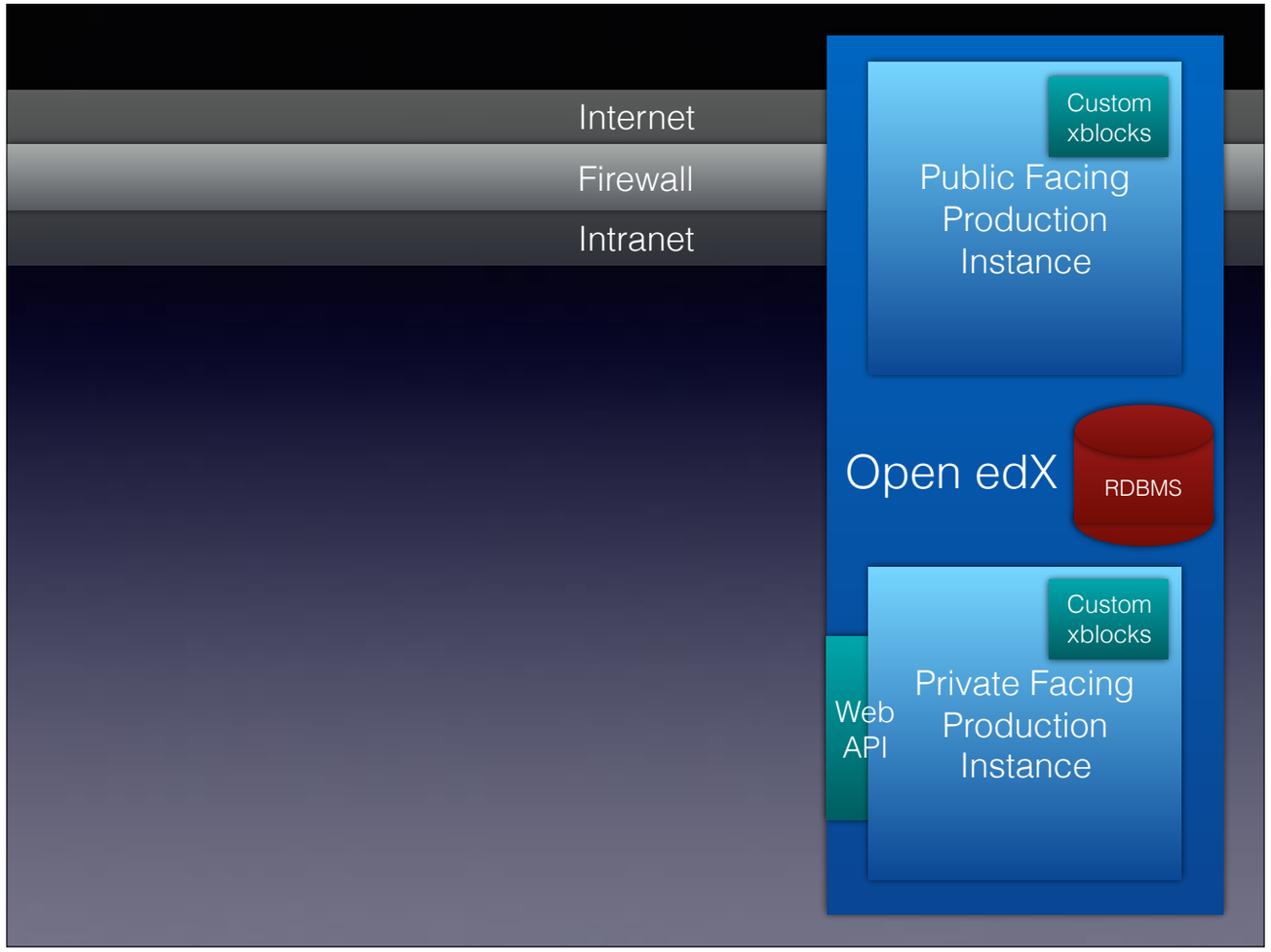
- Open edX will be a delegate system to existing back office process' and systems.
  - User Records and Enrollment records will migrate to Open edX from the back office.
- Mitigate risk from changes to the Open edX platform.
  - Push records to Open edX through a web API, placing records in landing tables on Open edX's RDBMS.

To appreciate the approach to integrating Open edX at Global Knowledge, there are key characteristics to our approach to understand. [click] Open edX is a delegate system to Global Knowledge's existing back office processes and systems. [click] User and enrollment records migrate to Open edX from Global Knowledge's back office systems. For Global Knowledge user's and enrollments will not be created in Open edX. Instead these records will be created in Global Knowledge's back office systems. [click] Integrations are designed to mitigating risk from changes to the Open edX platform. We create additional functionality by adding Django apps as plugins. With our plugins depending on Open edX core code instead of modifying Open edX. [click] Global Knowledge records are pushed to Open edX through custom web APIs, and placed on Open edX's relational database management system in a set of landing tables. Global Knowledge's existing environment of back office systems, which manage purchasing, student account management and course enrollment are the systems of truth for Global Knowledge's business. These back office systems not only support self-paced courses within Open edX, but also support Global Knowledge's other course modalities; instructor lead and virtual instructor lead.

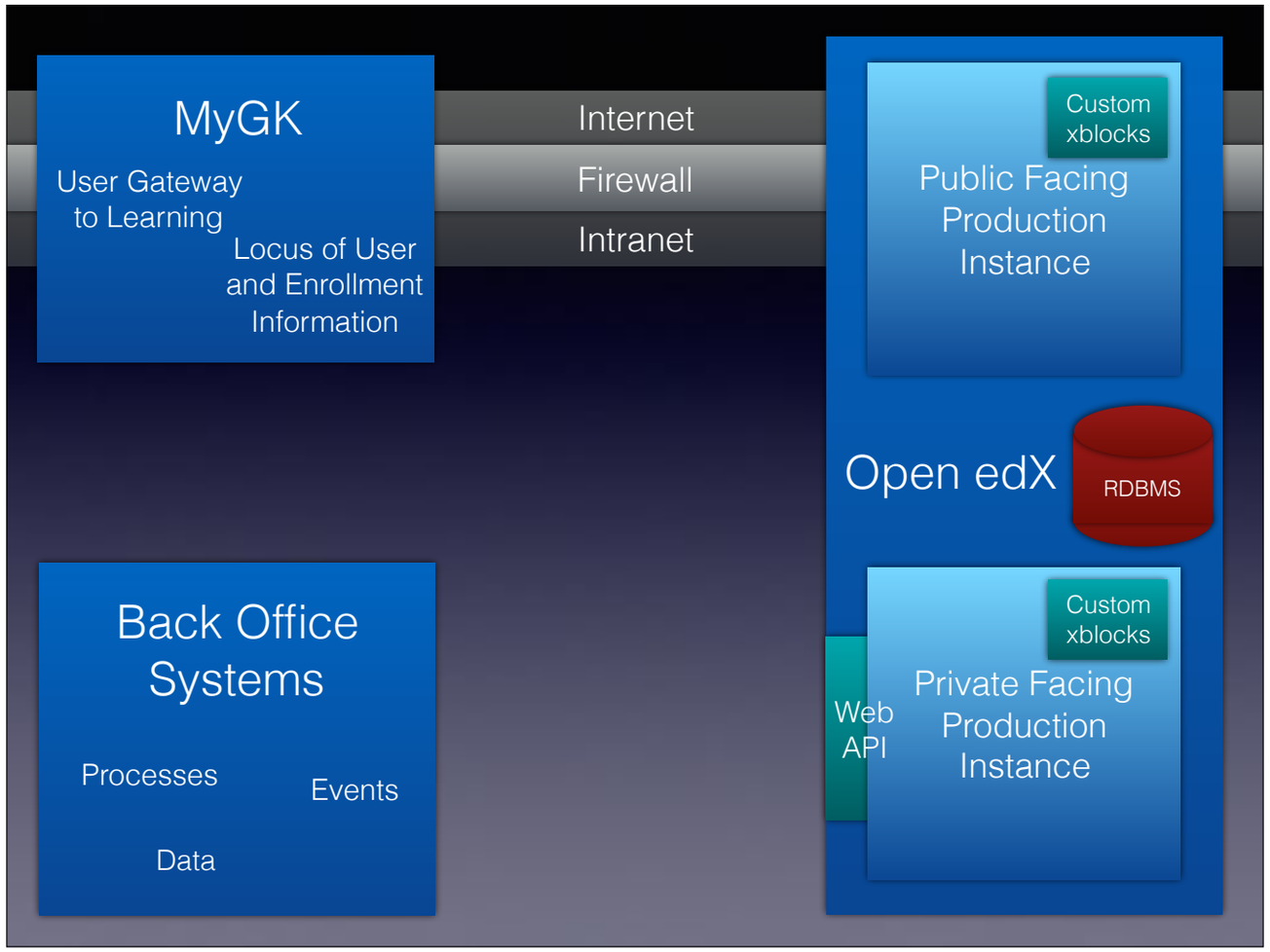
I’d like to start with a brief overview of the environment and systems that out instance of Open edX interacts with, to provide context for how Open edX is integrated within Global Knowledge.

Environment Overview

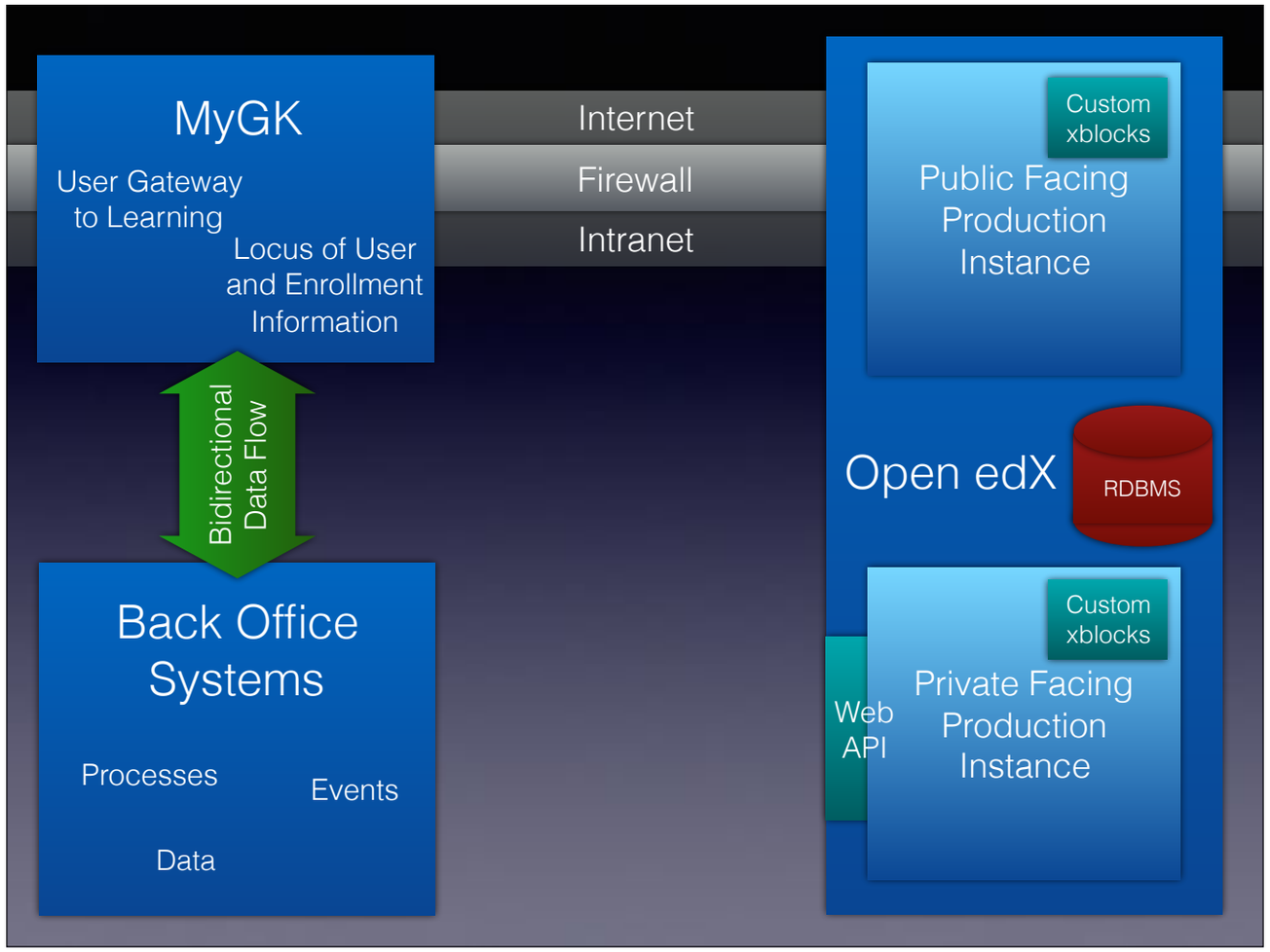
I'd like to start with a brief overview of the environment and systems that out instance of Open edX interacts with, to provide context for how Open edX is integrated within Global Knowledge.



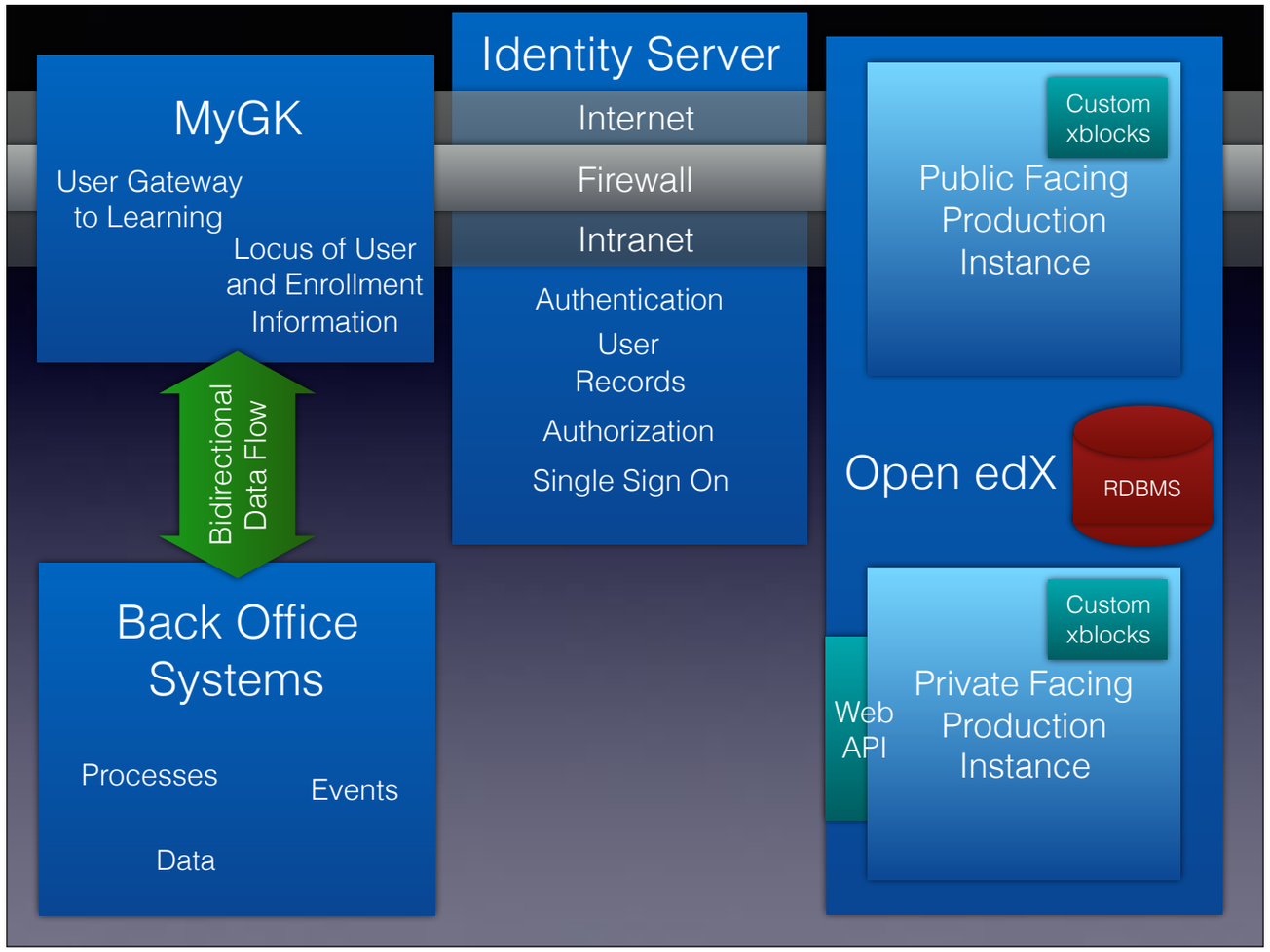
Open edX integration at Global Knowledge happens behind the firewall. [click] The MyGK web application ties together the user experience across Global Knowledge's web applications, and is the gateway for users at Global Knowledge. [click] Data flows bidirectionally between MyGK and the back office systems, passing user information and enrollment data as user interactions and back office processes create change in those records. [click] An Identity Server provides authentication and authorization services [click] for Global Knowledge applications [click], including MyGK and Open edX. The Open edX authentication integration was achieved via Open edX's third party authentication integration. [click] Open edX is deployed as a distributed system at Global Knowledge, with both public and private facing production instances. These separate Open edX instances share services such as data systems. Events in the back office, initiated by updates to user and enrollment records, trigger the migration [click] of records from Global Knowledge's back office to Open edX. The public facing Open edX instance handles user traffic, while the private facing Open edX instance handles internal Global Knowledge traffic, including traffic to the web API, through which the migrated records are received.



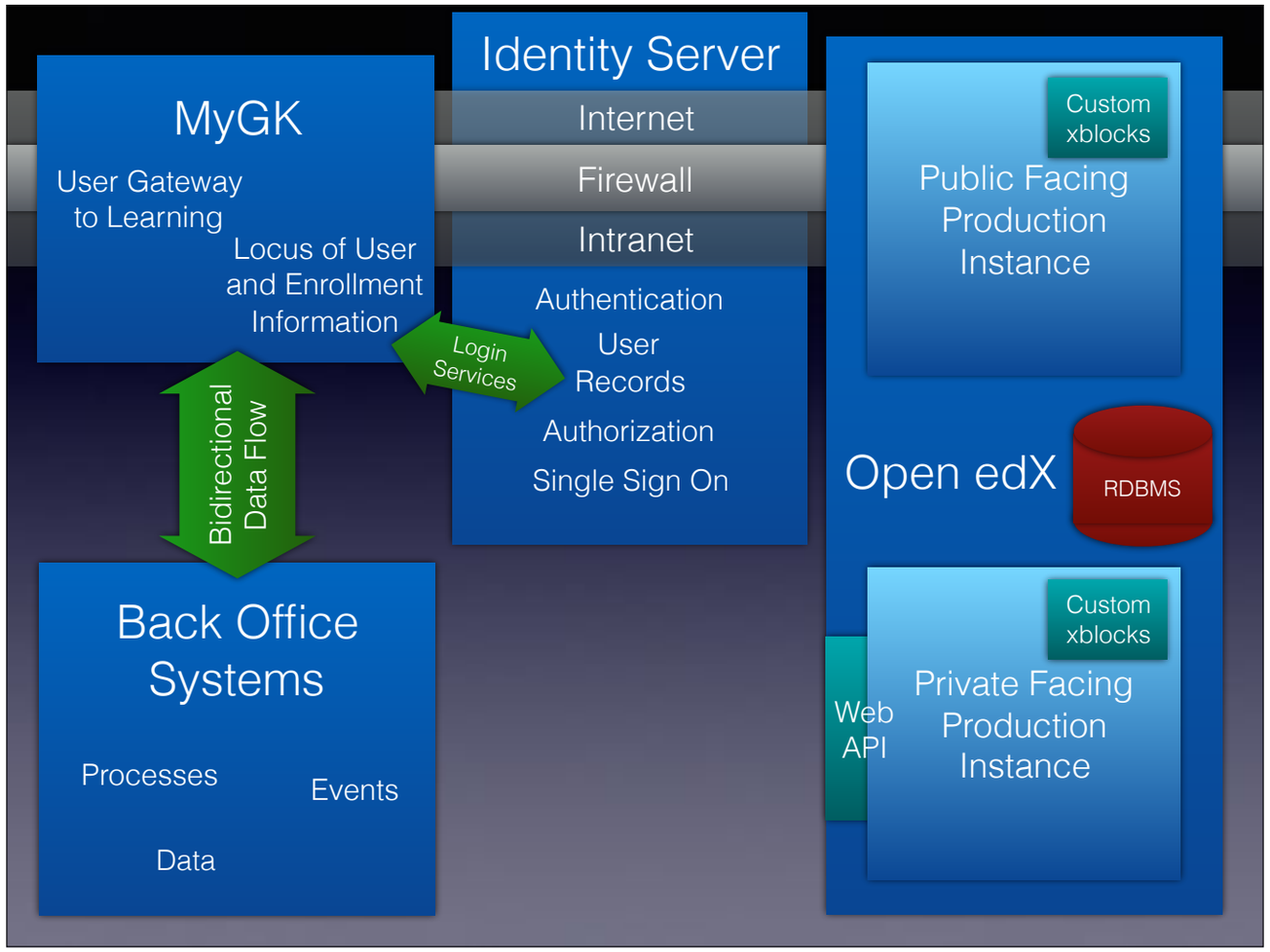
Open edX integration at Global Knowledge happens behind the firewall. [click] The MyGK web application ties together the user experience across Global Knowledge's web applications, and is the gateway for users at Global Knowledge. [click] Data flows bidirectionally between MyGK and the back office systems, passing user information and enrollment data as user interactions and back office processes create change in those records. [click] An Identity Server provides authentication and authorization services [click] for Global Knowledge applications [click], including MyGK and Open edX. The Open edX authentication integration was achieved via Open edX's third party authentication integration. [click] Open edX is deployed as a distributed system at Global Knowledge, with both public and private facing production instances. These separate Open edX instances share services such as data systems. Events in the back office, initiated by updates to user and enrollment records, trigger the migration [click] of records from Global Knowledge's back office to Open edX. The public facing Open edX instance handles user traffic, while the private facing Open edX instance handles internal Global Knowledge traffic, including traffic to the web API, through which the migrated records are received.



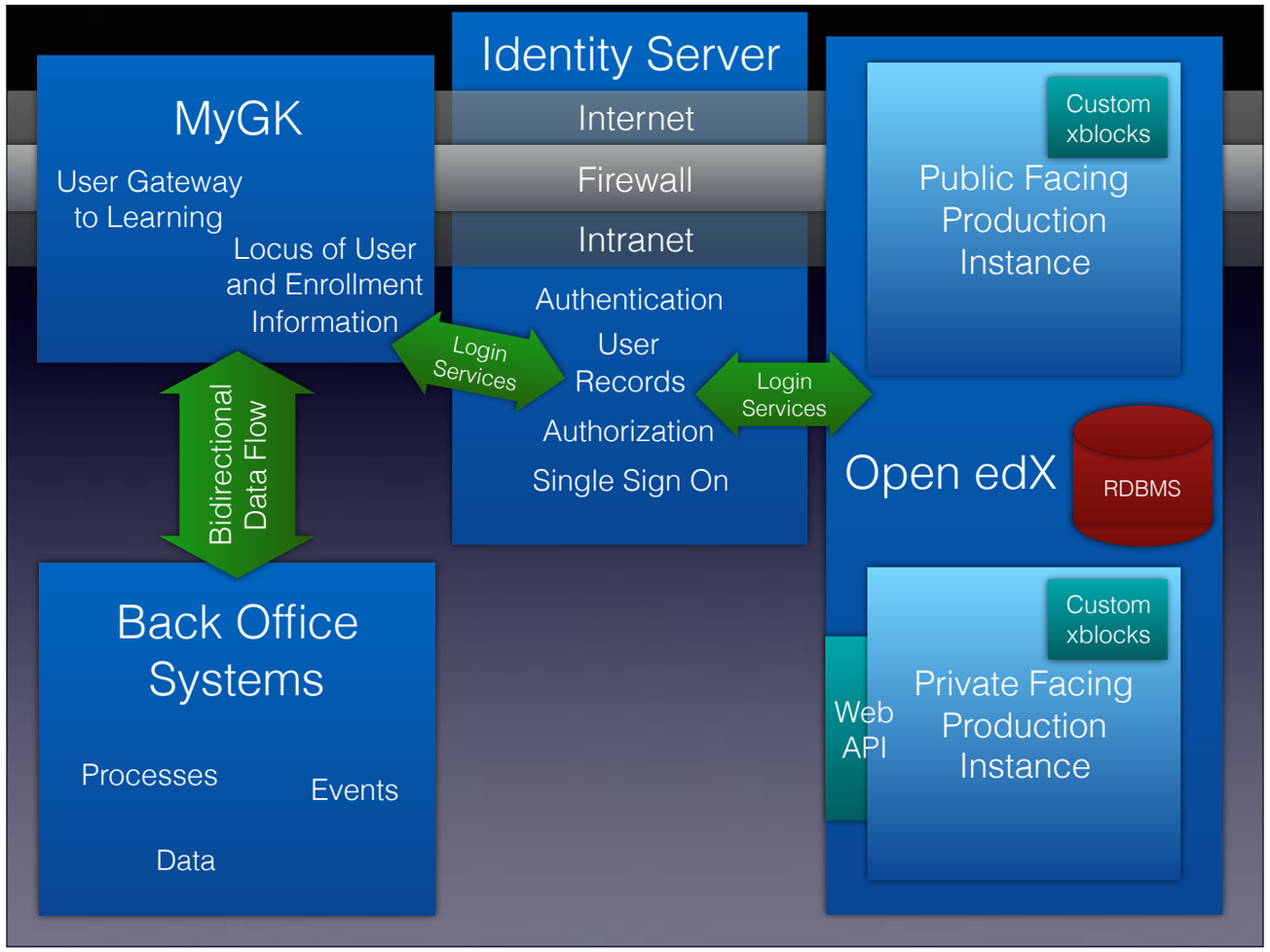
Open edX integration at Global Knowledge happens behind the firewall. [click] The MyGK web application ties together the user experience across Global Knowledge's web applications, and is the gateway for users at Global Knowledge. [click] Data flows bidirectionally between MyGK and the back office systems, passing user information and enrollment data as user interactions and back office processes create change in those records. [click] An Identity Server provides authentication and authorization services [click] for Global Knowledge applications [click], including MyGK and Open edX. The Open edX authentication integration was achieved via Open edX's third party authentication integration. [click] Open edX is deployed as a distributed system at Global Knowledge, with both public and private facing production instances. These separate Open edX instances share services such as data systems. Events in the back office, initiated by updates to user and enrollment records, trigger the migration [click] of records from Global Knowledge's back office to Open edX. The public facing Open edX instance handles user traffic, while the private facing Open edX instance handles internal Global Knowledge traffic, including traffic to the web API, through which the migrated records are received.



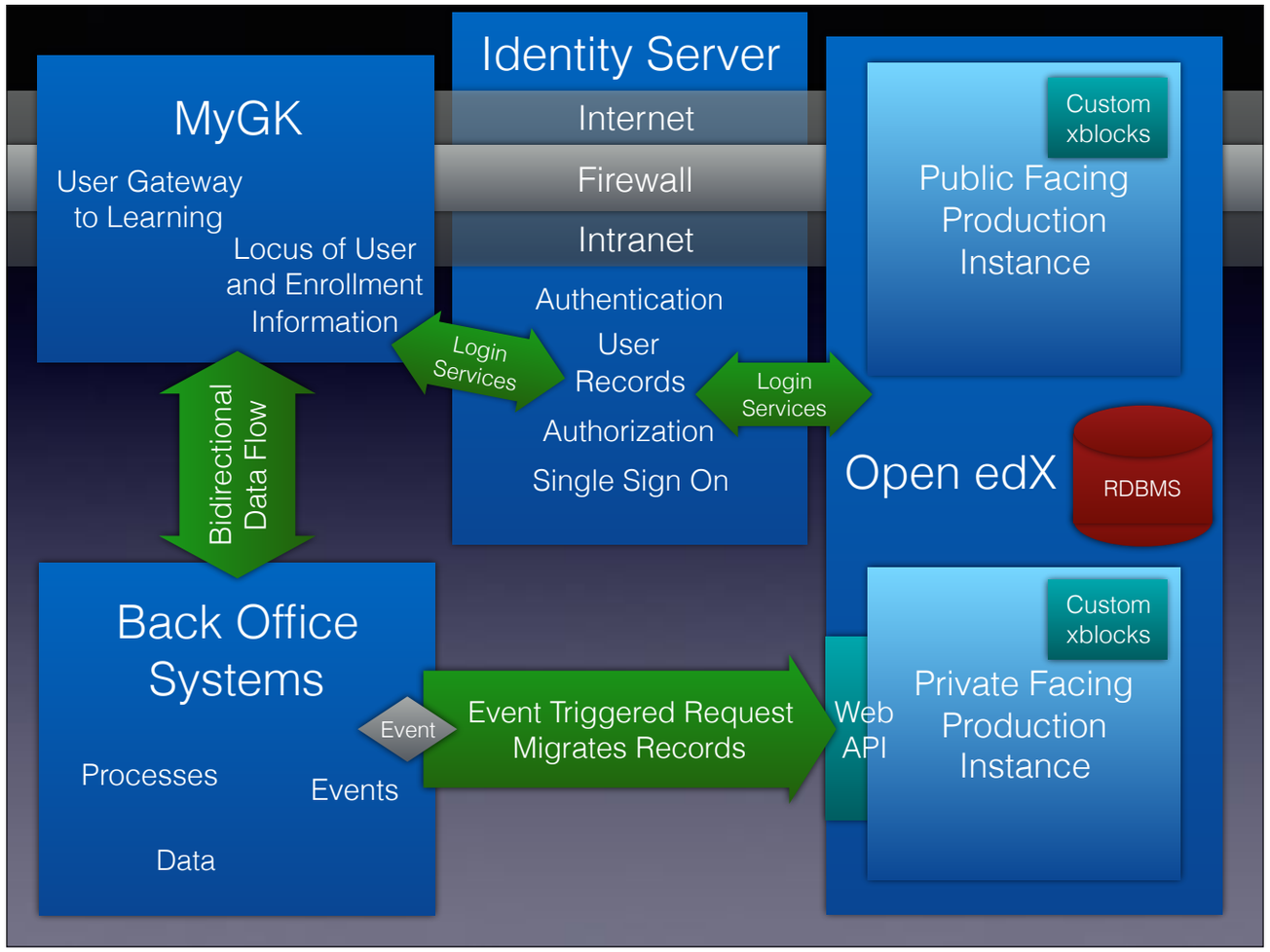
Open edX integration at Global Knowledge happens behind the firewall. [click] The MyGK web application ties together the user experience across Global Knowledge's web applications, and is the gateway for users at Global Knowledge. [click] Data flows bidirectionally between MyGK and the back office systems, passing user information and enrollment data as user interactions and back office processes create change in those records. [click] An Identity Server provides authentication and authorization services [click] for Global Knowledge applications [click], including MyGK and Open edX. The Open edX authentication integration was achieved via Open edX's third party authentication integration. [click] Open edX is deployed as a distributed system at Global Knowledge, with both public and private facing production instances. These separate Open edX instances share services such as data systems. Events in the back office, initiated by updates to user and enrollment records, trigger the migration [click] of records from Global Knowledge's back office to Open edX. The public facing Open edX instance handles user traffic, while the private facing Open edX instance handles internal Global Knowledge traffic, including traffic to the web API, through which the migrated records are received.



Open edX integration at Global Knowledge happens behind the firewall. [click] The MyGK web application ties together the user experience across Global Knowledge's web applications, and is the gateway for users at Global Knowledge. [click] Data flows bidirectionally between MyGK and the back office systems, passing user information and enrollment data as user interactions and back office processes create change in those records. [click] An Identity Server provides authentication and authorization services [click] for Global Knowledge applications [click], including MyGK and Open edX. The Open edX authentication integration was achieved via Open edX's third party authentication integration. [click] Open edX is deployed as a distributed system at Global Knowledge, with both public and private facing production instances. These separate Open edX instances share services such as data systems. Events in the back office, initiated by updates to user and enrollment records, trigger the migration [click] of records from Global Knowledge's back office to Open edX. The public facing Open edX instance handles user traffic, while the private facing Open edX instance handles internal Global Knowledge traffic, including traffic to the web API, through which the migrated records are received.



Open edX integration at Global Knowledge happens behind the firewall. [click] The MyGK web application ties together the user experience across Global Knowledge's web applications, and is the gateway for users at Global Knowledge. [click] Data flows bidirectionally between MyGK and the back office systems, passing user information and enrollment data as user interactions and back office processes create change in those records. [click] An Identity Server provides authentication and authorization services [click] for Global Knowledge applications [click], including MyGK and Open edX. The Open edX authentication integration was achieved via Open edX's third party authentication integration. [click] Open edX is deployed as a distributed system at Global Knowledge, with both public and private facing production instances. These separate Open edX instances share services such as data systems. Events in the back office, initiated by updates to user and enrollment records, trigger the migration [click] of records from Global Knowledge's back office to Open edX. The public facing Open edX instance handles user traffic, while the private facing Open edX instance handles internal Global Knowledge traffic, including traffic to the web API, through which the migrated records are received.



Open edX integration at Global Knowledge happens behind the firewall. [click] The MyGK web application ties together the user experience across Global Knowledge's web applications, and is the gateway for users at Global Knowledge. [click] Data flows bidirectionally between MyGK and the back office systems, passing user information and enrollment data as user interactions and back office processes create change in those records. [click] An Identity Server provides authentication and authorization services [click] for Global Knowledge applications [click], including MyGK and Open edX. The Open edX authentication integration was achieved via Open edX's third party authentication integration. [click] Open edX is deployed as a distributed system at Global Knowledge, with both public and private facing production instances. These separate Open edX instances share services such as data systems. Events in the back office, initiated by updates to user and enrollment records, trigger the migration [click] of records from Global Knowledge's back office to Open edX. The public facing Open edX instance handles user traffic, while the private facing Open edX instance handles internal Global Knowledge traffic, including traffic to the web API, through which the migrated records are received.

I’d like to start with the migration of user and enrollment records into Open edX. Open edX is not a system of record at Global Knowledge. Creation of users, courses and enrollments happen outside of Open edX and are migrated and imported into Open edX. The necessary data is migrated from other systems into Open edX through APIs. User and enrollment data is migrated thru the API, while course creation and importing is a separate set of systems and processes that are outside the scope of this talk.

Migration of Records

I'd like to start with the migration of user and enrollment records into Open edX. Open edX is not a system of record at Global Knowledge. Creation of users, courses and enrollments happen outside of Open edX and are migrated and imported into Open edX. The necessary data is migrated from other systems into Open edX through APIs. User and enrollment data is migrated thru the API, while course creation and importing is a separate set of systems and processes that are outside the scope of this talk.

Resource Document Definition

The migrated enrollment records are defined as a web resource for portability and cross platform compatibility. [click] The records are defined as JSON and adhere to an agreement between the different teams responsible for the different Global Knowledge systems involved. [click] The record is coarse by intention, containing both student/contact data and course enrollment data in the same JSON object definition. [click] The record is complete in its representation of the state of the student and enrollment resource, [click] containing all information to create, update or replace a record in the landing tables on Open edX.

Resource Document Definition

- MIME Type
  `application/json`

The migrated enrollment records are defined as a web resource for portability and cross platform compatibility. [click] The records are defined as JSON and adhere to an agreement between the different teams responsible for the different Global Knowledge systems involved. [click] The record is coarse by intention, containing both student/contact data and course enrollment data in the same JSON object definition. [click] The record is complete in its representation of the state of the student and enrollment resource, [click] containing all information to create, update or replace a record in the landing tables on Open edX.

Resource Document Definition

- MIME Type
  `application/json`

- Coarse, containing both student/contact data and enrollment data.

The migrated enrollment records are defined as a web resource for portability and cross platform compatibility. [click] The records are defined as JSON and adhere to an agreement between the different teams responsible for the different Global Knowledge systems involved. [click] The record is coarse by intention, containing both student/contact data and course enrollment data in the same JSON object definition. [click] The record is complete in its representation of the state of the student and enrollment resource, [click] containing all information to create, update or replace a record in the landing tables on Open edX.

Resource Document Definition

- MIME Type
  `application/json`
- Coarse, containing both student/contact data and enrollment data.
- Atomic, each instance contains the full state of the resource.

The migrated enrollment records are defined as a web resource for portability and cross platform compatibility. [click] The records are defined as JSON and adhere to an agreement between the different teams responsible for the different Global Knowledge systems involved. [click] The record is coarse by intention, containing both student/contact data and course enrollment data in the same JSON object definition. [click] The record is complete in its representation of the state of the student and enrollment resource, [click] containing all information to create, update or replace a record in the landing tables on Open edX.

Resource Document Definition

- MIME Type
  application/json
- Coarse, containing both student/contact data and enrollment data.
- Atomic, each instance contains the full state of the resource.

```
{
    "Contact": {
        "ContactID": "XXX123",
        "FirstName": "Joe",
        "LastName": "Smith",
        "Email": "joe.smith@acme.com",
        "Enrollments": [
            {
                "CourseCode": "1234A",
                "AccessStartDate": "2017-10-01",
                "AccessEndDate": "2018-09-30"
            },
            {
                "CourseCode": "1235A",
                "AccessStartDate": "2017-10-01",
                "AccessEndDate": "2018-09-30"
            },
            {
                "CourseCode": "1236A",
                "AccessStartDate": "2017-10-01",
                "AccessEndDate": "2018-09-30"
            },
        ]
    }
}
```

The migrated enrollment records are defined as a web resource for portability and cross platform compatibility. [click] The records are defined as JSON and adhere to an agreement between the different teams responsible for the different Global Knowledge systems involved. [click] The record is coarse by intention, containing both student/contact data and course enrollment data in the same JSON object definition. [click] The record is complete in its representation of the state of the student and enrollment resource, [click] containing all information to create, update or replace a record in the landing tables on Open edX.

Web API Definition

The records are transferred to Open edX thru a web API with the following characteristics. [click] The web API uses bookmark-able URLs. The URL will always address this student/contact, either returning the record or a 404 to indicate the record does not exist. The book markable URLs improve log entries by including the ID in the time stamped logs, indicating when a particular record was migrated from Global Knowledge's back office to Open edX. [click] The API constrains the accepted resources to the JSON content type. Rejecting requests formatted in other types; XML, form data, etc. [click] The API makes idiomatic use of the methods and status' available through the HTTP protocol. Idiomatic use of the HTTP protocol removes the need to define additional actions or error indicators in the request or response bodies.

Web API Definition

- Book-Markable URL: *`/contacts/<contact_id>/enrollments/`*

The records are transferred to Open edX thru a web API with the following characteristics. [click] The web API uses bookmark-able URLs. The URL will always address this student/contact, either returning the record or a 404 to indicate the record does not exist. The book markable URLs improve log entries by including the ID in the time stamped logs, indicating when a particular record was migrated from Global Knowledge's back office to Open edX. [click] The API constrains the accepted resources to the JSON content type. Rejecting requests formatted in other types; XML, form data, etc. [click] The API makes idiomatic use of the methods and status' available through the HTTP protocol. Idiomatic use of the HTTP protocol removes the need to define additional actions or error indicators in the request or response bodies.

Web API Definition

- Book-Markable URL: ***/contacts/<contact_id>/enrollments/***
- Constrains Resources by Content-Type: ***application/json***

The records are transferred to Open edX thru a web API with the following characteristics. [click] The web API uses bookmark-able URLs. The URL will always address this student/contact, either returning the record or a 404 to indicate the record does not exist. The book markable URLs improve log entries by including the ID in the time stamped logs, indicating when a particular record was migrated from Global Knowledge's back office to Open edX. [click] The API constrains the accepted resources to the JSON content type. Rejecting requests formatted in other types; XML, form data, etc. [click] The API makes idiomatic use of the methods and status' available through the HTTP protocol. Idiomatic use of the HTTP protocol removes the need to define additional actions or error indicators in the request or response bodies.

Web API Definition

- Book-Markable URL: ***/contacts/<contact_id>/enrollments/***
- Constrains Resources by Content-Type: ***application/json***
- Idiomatic Use of HTTP Methods and Status Codes: Put for Create/Update with codes (200 for updated, 201 for created, 403, 405, 409 & 500) used to indicate status of request.

The records are transferred to Open edX thru a web API with the following characteristics. [click] The web API uses bookmark-able URLs. The URL will always address this student/contact, either returning the record or a 404 to indicate the record does not exist. The book markable URLs improve log entries by including the ID in the time stamped logs, indicating when a particular record was migrated from Global Knowledge's back office to Open edX. [click] The API constrains the accepted resources to the JSON content type. Rejecting requests formatted in other types; XML, form data, etc. [click] The API makes idiomatic use of the methods and status' available through the HTTP protocol. Idiomatic use of the HTTP protocol removes the need to define additional actions or error indicators in the request or response bodies.

Web API Implementation

How the API was developed and integrated into Open edX was equally important. The API implementation approach adheres to the following characteristics. [click] The API will is implemented as a Django 1.8 compliant app.[click] The API uses the Django ReST Framework, standardizing the view implementation with existing views in Open edX. [click] The Web API Django app was designed and developed to be Python package installable. Django apps are Python packages, and can be installed by Python packaging tools like setuptools. [click] The API app is integrated through configuration with Open edX, no modifications to Open edX code are required, other than additions to settings and configuration files. [click] The Web API URLs occupy a new namespace separate from Open edX URLs, avoiding URL collisions with existing Open edX URLs. [click] The API has dependencies on core Open edX libraries for creating and updating edX native users and enrollments. Orchestrating rather than modifying existing Open edX code to meet Global Knowledge's business requirements.

Web API Implementation

- Django 1.8 Compliant App

How the API was developed and integrated into Open edX was equally important. The API implementation approach adheres to the following characteristics. [click] The API will is implemented as a Django 1.8 compliant app.[click] The API uses the Django ReST Framework, standardizing the view implementation with existing views in Open edX. [click] The Web API Django app was designed and developed to be Python package installable. Django apps are Python packages, and can be installed by Python packaging tools like setuptools. [click] The API app is integrated through configuration with Open edX, no modifications to Open edX code are required, other than additions to settings and configuration files. [click] The Web API URLs occupy a new namespace separate from Open edX URLs, avoiding URL collisions with existing Open edX URLs. [click] The API has dependencies on core Open edX libraries for creating and updating edX native users and enrollments. Orchestrating rather than modifying existing Open edX code to meet Global Knowledge's business requirements.

Web API Implementation

- Django 1.8 Compliant App
- Uses the Django ReST Framework Package

How the API was developed and integrated into Open edX was equally important. The API implementation approach adheres to the following characteristics. [click] The API will is implemented as a Django 1.8 compliant app.[click] The API uses the Django ReST Framework, standardizing the view implementation with existing views in Open edX. [click] The Web API Django app was designed and developed to be Python package installable. Django apps are Python packages, and can be installed by Python packaging tools like setuptools. [click] The API app is integrated through configuration with Open edX, no modifications to Open edX code are required, other than additions to settings and configuration files. [click] The Web API URLs occupy a new namespace separate from Open edX URLs, avoiding URL collisions with existing Open edX URLs. [click] The API has dependencies on core Open edX libraries for creating and updating edX native users and enrollments. Orchestrating rather than modifying existing Open edX code to meet Global Knowledge's business requirements.

Web API Implementation

- Django 1.8 Compliant App
- Uses the Django ReST Framework Package
- Python Package Installable

How the API was developed and integrated into Open edX was equally important. The API implementation approach adheres to the following characteristics. [click] The API will is implemented as a Django 1.8 compliant app.[click] The API uses the Django ReST Framework, standardizing the view implementation with existing views in Open edX. [click] The Web API Django app was designed and developed to be Python package installable. Django apps are Python packages, and can be installed by Python packaging tools like setuptools. [click] The API app is integrated through configuration with Open edX, no modifications to Open edX code are required, other than additions to settings and configuration files. [click] The Web API URLs occupy a new namespace separate from Open edX URLs, avoiding URL collisions with existing Open edX URLs. [click] The API has dependencies on core Open edX libraries for creating and updating edX native users and enrollments. Orchestrating rather than modifying existing Open edX code to meet Global Knowledge's business requirements.

Web API Implementation

- Django 1.8 Compliant App
- Uses the Django ReST Framework Package
- Python Package Installable
- Integration Through Configuration Instead of Modification

How the API was developed and integrated into Open edX was equally important. The API implementation approach adheres to the following characteristics. [click] The API will is implemented as a Django 1.8 compliant app.[click] The API uses the Django ReST Framework, standardizing the view implementation with existing views in Open edX. [click] The Web API Django app was designed and developed to be Python package installable. Django apps are Python packages, and can be installed by Python packaging tools like setuptools. [click] The API app is integrated through configuration with Open edX, no modifications to Open edX code are required, other than additions to settings and configuration files. [click] The Web API URLs occupy a new namespace separate from Open edX URLs, avoiding URL collisions with existing Open edX URLs. [click] The API has dependencies on core Open edX libraries for creating and updating edX native users and enrollments. Orchestrating rather than modifying existing Open edX code to meet Global Knowledge's business requirements.

Web API Implementation

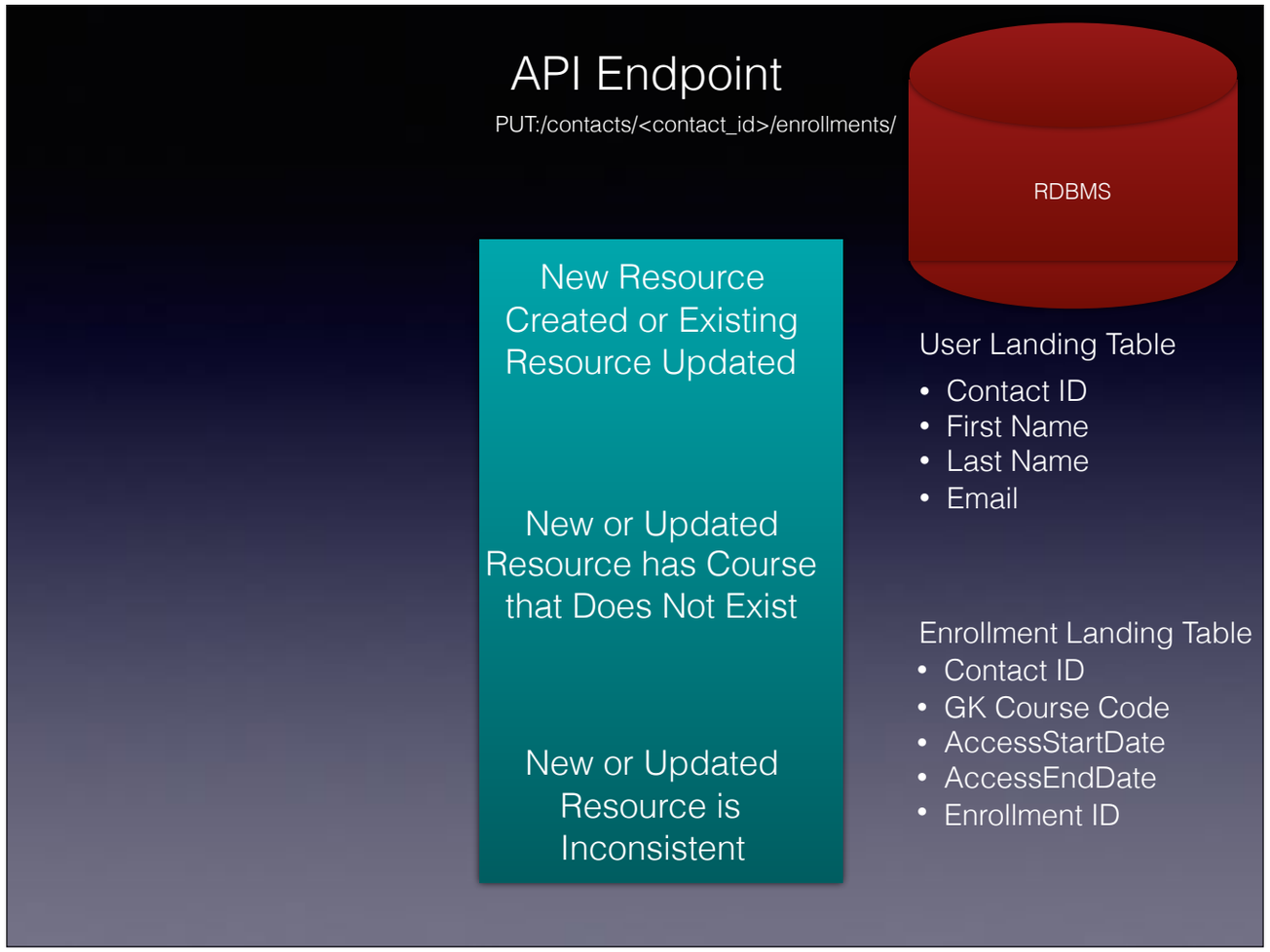
- Django 1.8 Compliant App
- Uses the Django ReST Framework Package
- Python Package Installable
- Integration Through Configuration Instead of Modification
- URL Mapping Extends Open edX URLs Under a ***/gk/api/*** Namespace

How the API was developed and integrated into Open edX was equally important. The API implementation approach adheres to the following characteristics. [click] The API will is implemented as a Django 1.8 compliant app.[click] The API uses the Django ReST Framework, standardizing the view implementation with existing views in Open edX. [click] The Web API Django app was designed and developed to be Python package installable. Django apps are Python packages, and can be installed by Python packaging tools like setuptools. [click] The API app is integrated through configuration with Open edX, no modifications to Open edX code are required, other than additions to settings and configuration files. [click] The Web API URLs occupy a new namespace separate from Open edX URLs, avoiding URL collisions with existing Open edX URLs. [click] The API has dependencies on core Open edX libraries for creating and updating edX native users and enrollments. Orchestrating rather than modifying existing Open edX code to meet Global Knowledge's business requirements.

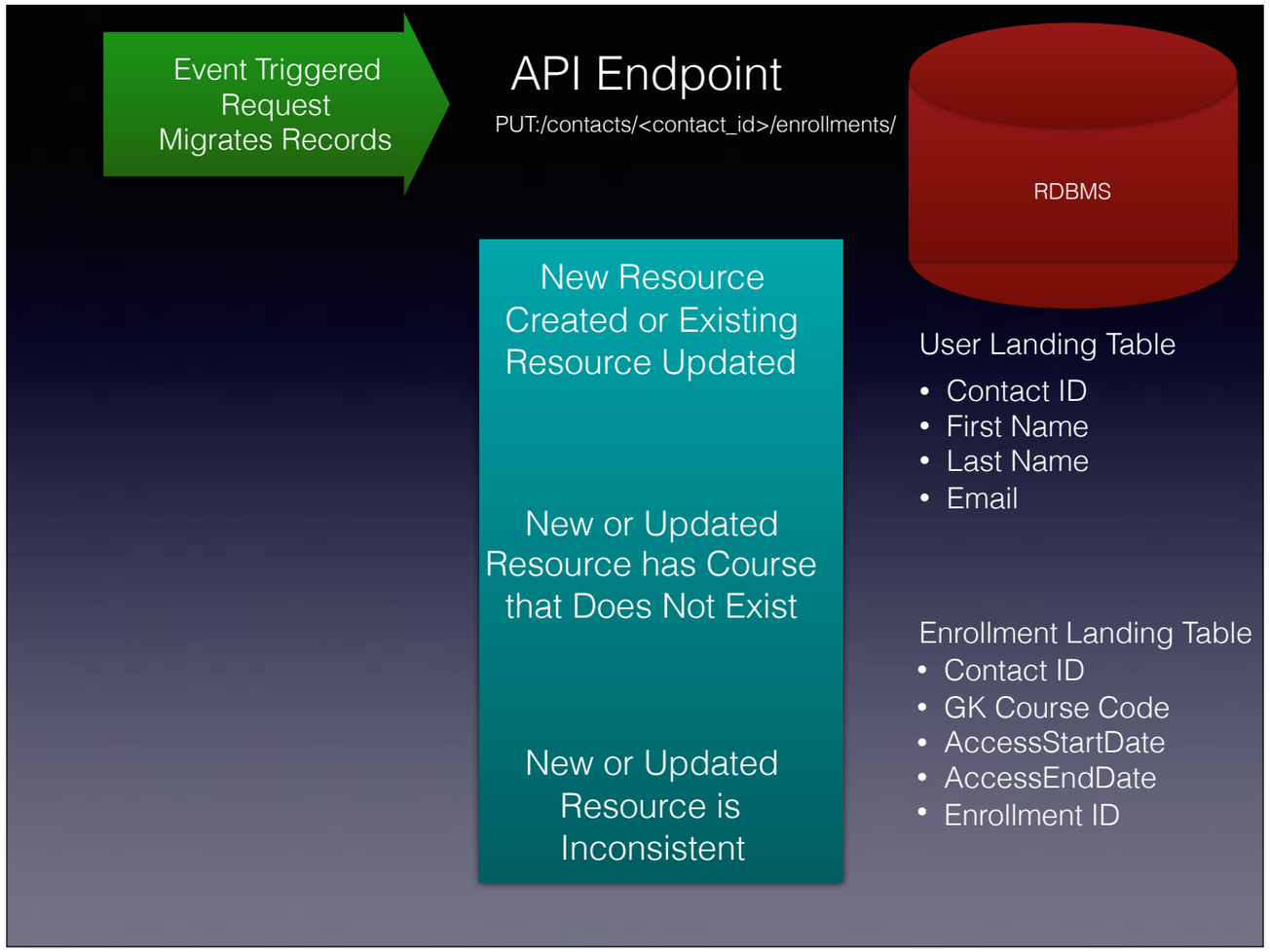
Web API Implementation

- Django 1.8 Compliant App
- Uses the Django ReST Framework Package
- Python Package Installable
- Integration Through Configuration Instead of Modification
- URL Mapping Extends Open edX URLs Under a ***/gk/api/*** Namespace
- App Depends on Open edX Core Libraries for Creating and Updating Users and Enrollments

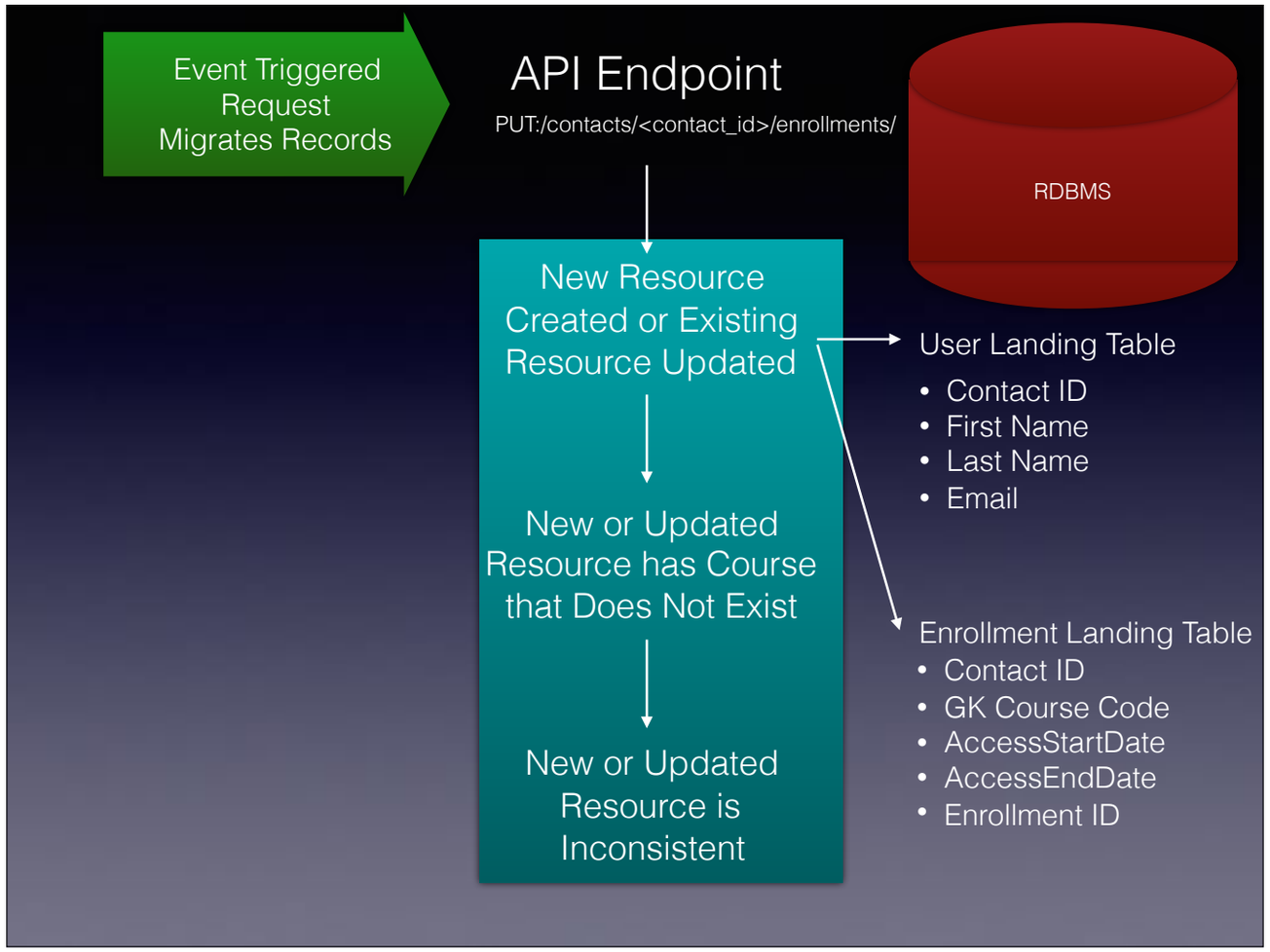
How the API was developed and integrated into Open edX was equally important. The API implementation approach adheres to the following characteristics. [click] The API will is implemented as a Django 1.8 compliant app.[click] The API uses the Django ReST Framework, standardizing the view implementation with existing views in Open edX. [click] The Web API Django app was designed and developed to be Python package installable. Django apps are Python packages, and can be installed by Python packaging tools like setuptools. [click] The API app is integrated through configuration with Open edX, no modifications to Open edX code are required, other than additions to settings and configuration files. [click] The Web API URLs occupy a new namespace separate from Open edX URLs, avoiding URL collisions with existing Open edX URLs. [click] The API has dependencies on core Open edX libraries for creating and updating edX native users and enrollments. Orchestrating rather than modifying existing Open edX code to meet Global Knowledge's business requirements.



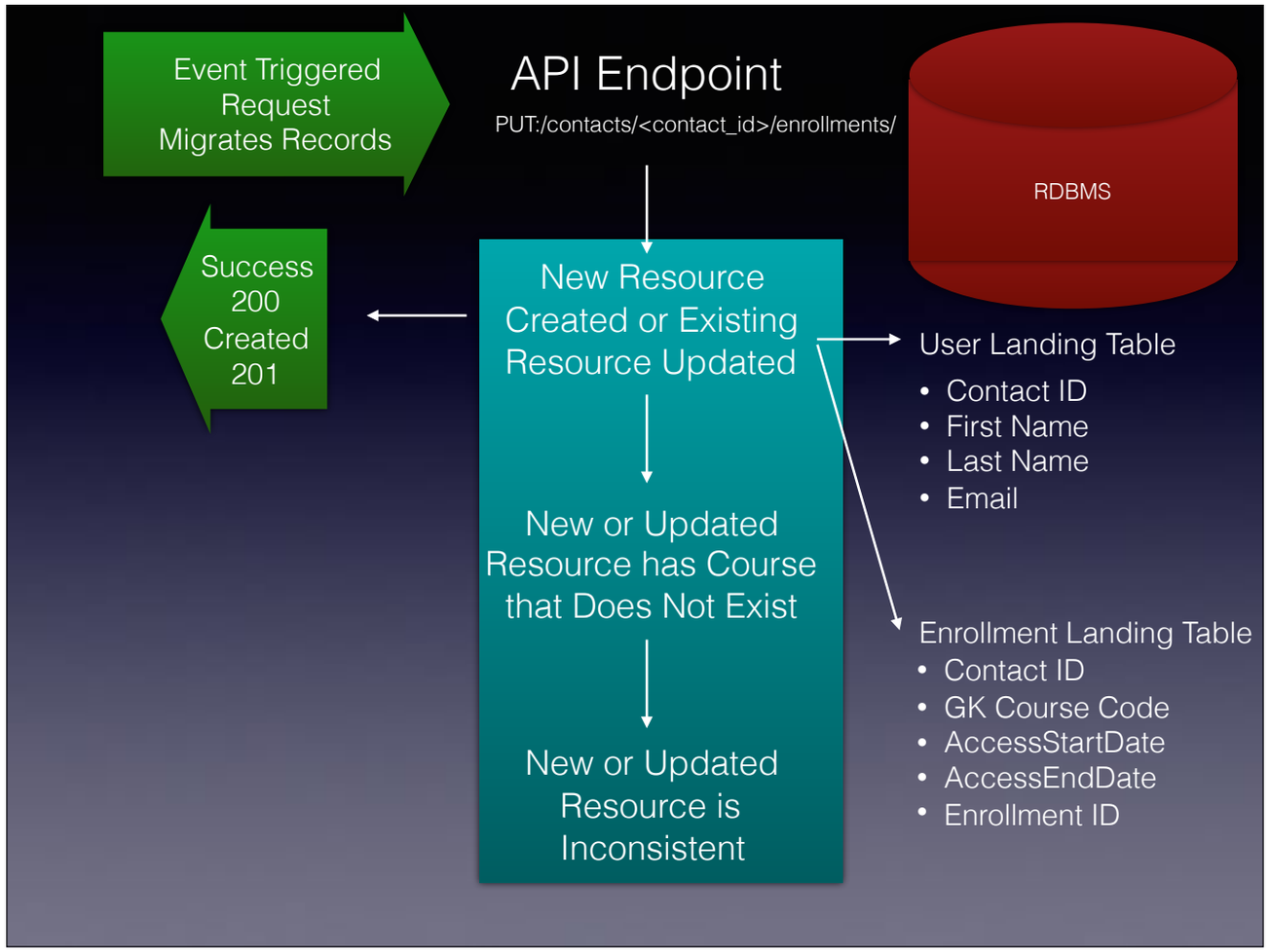
With an understanding of the what and where with regard to , lets look at how a record is received at the Web API. Records arriving at the Web API endpoint are validated [click] and compared to existing resource records in the landing tables. The correct HTTP status code is returned by the Web API, indicating the status of the request to the upstream system, 20X's for success [click] or 40X's [click] for errors. It is possible that for a 409, part of the record was been written to the landing tables successfully [click], the body in the response, details what part of the record caused the conflict. A 403 will not be processed at all, and the response body details what attribute of the resource is causing the fatal error.



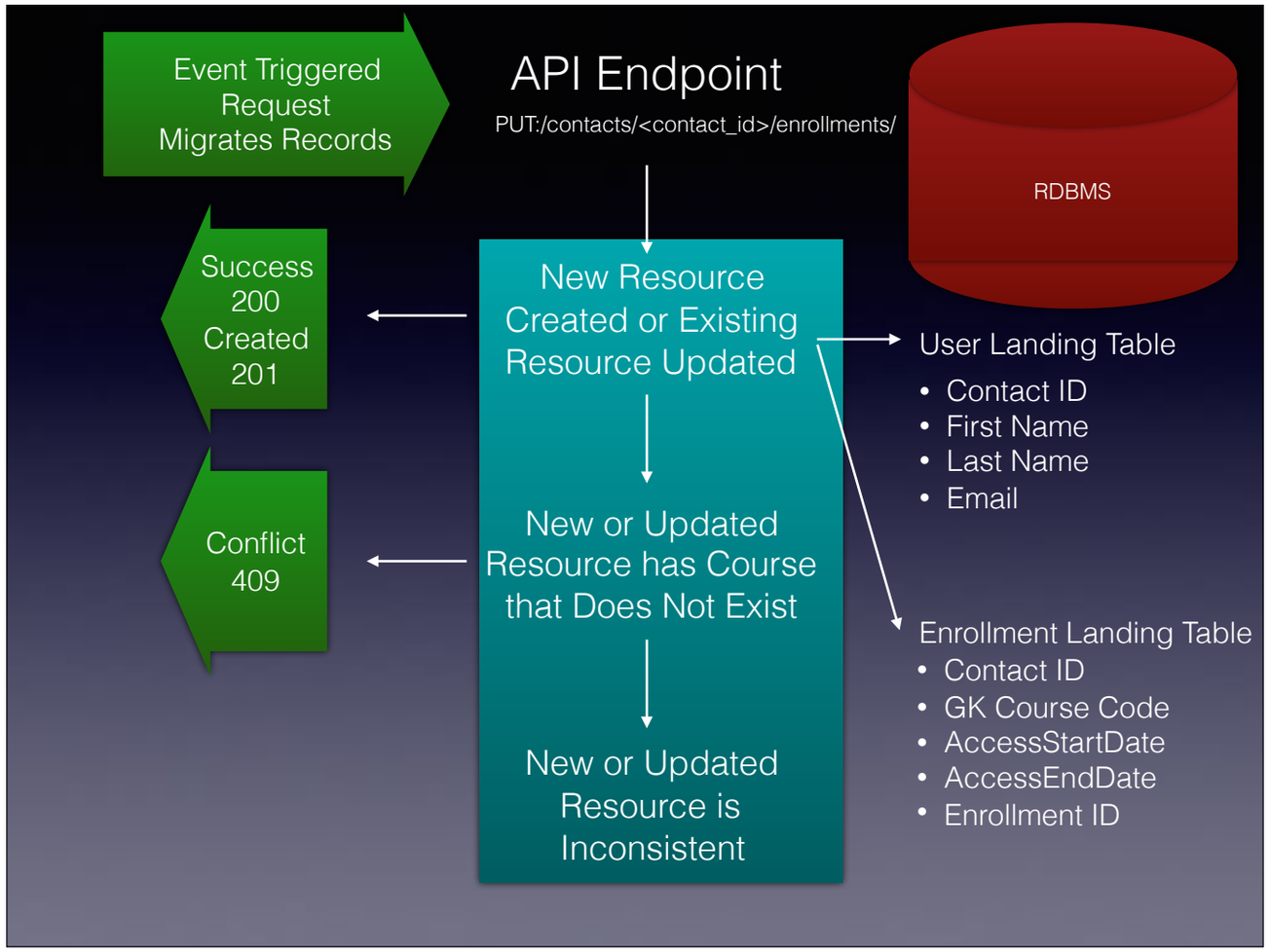
With an understanding of the what and where with regard to , lets look at how a record is received at the Web API. Records arriving at the Web API endpoint are validated [click] and compared to existing resource records in the landing tables. The correct HTTP status code is returned by the Web API, indicating the status of the request to the upstream system, 20X's for success [click] or 40X's [click] for errors. It is possible that for a 409, part of the record was been written to the landing tables successfully [click], the body in the response, details what part of the record caused the conflict. A 403 will not be processed at all, and the response body details what attribute of the resource is causing the fatal error.



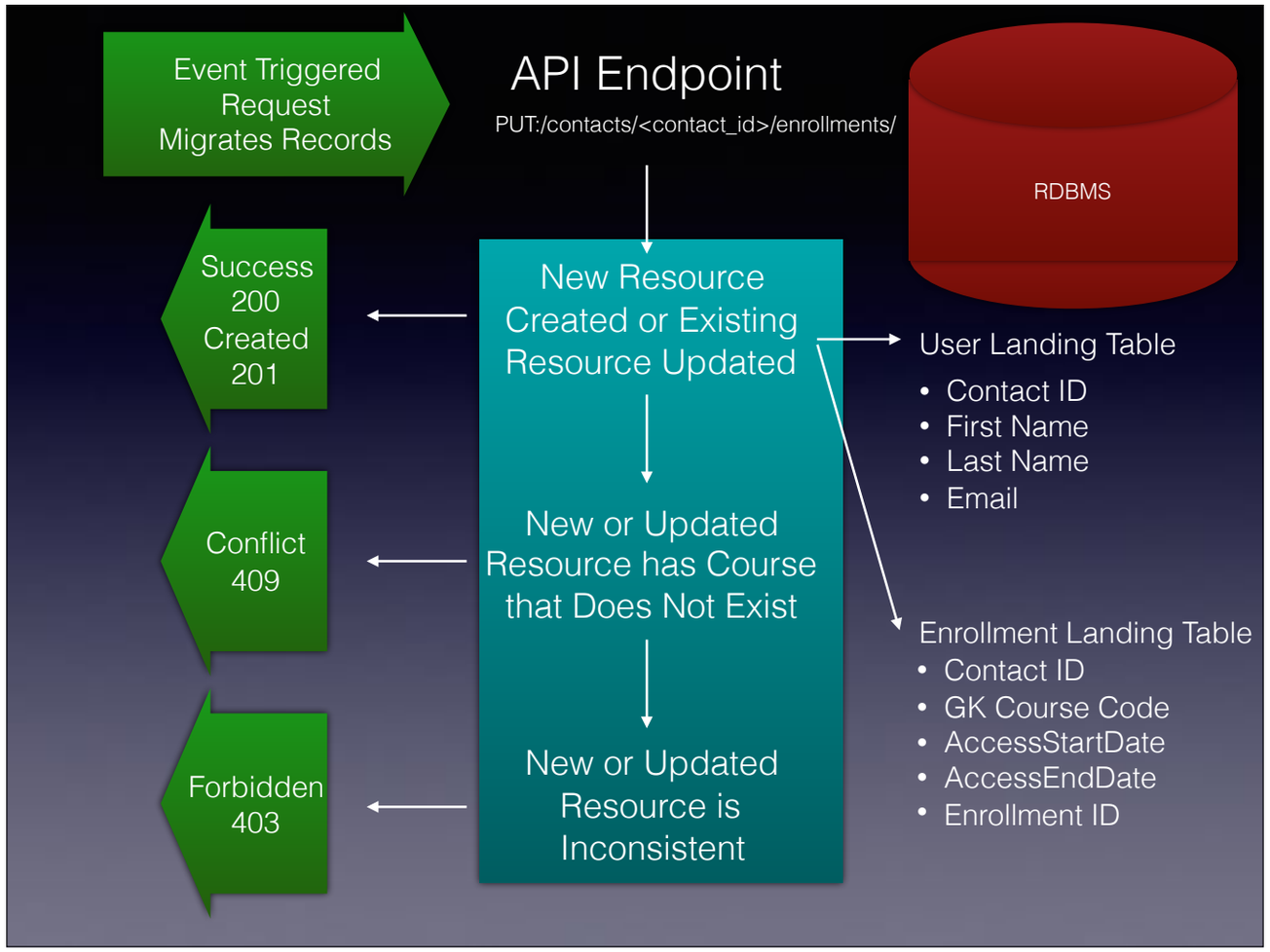
With an understanding of the what and where with regard to , lets look at how a record is received at the Web API. Records arriving at the Web API endpoint are validated [click] and compared to existing resource records in the landing tables. The correct HTTP status code is returned by the Web API, indicating the status of the request to the upstream system, 20X's for success [click] or 40X's [click] for errors. It is possible that for a 409, part of the record was been written to the landing tables successfully [click], the body in the response, details what part of the record caused the conflict. A 403 will not be processed at all, and the response body details what attribute of the resource is causing the fatal error.



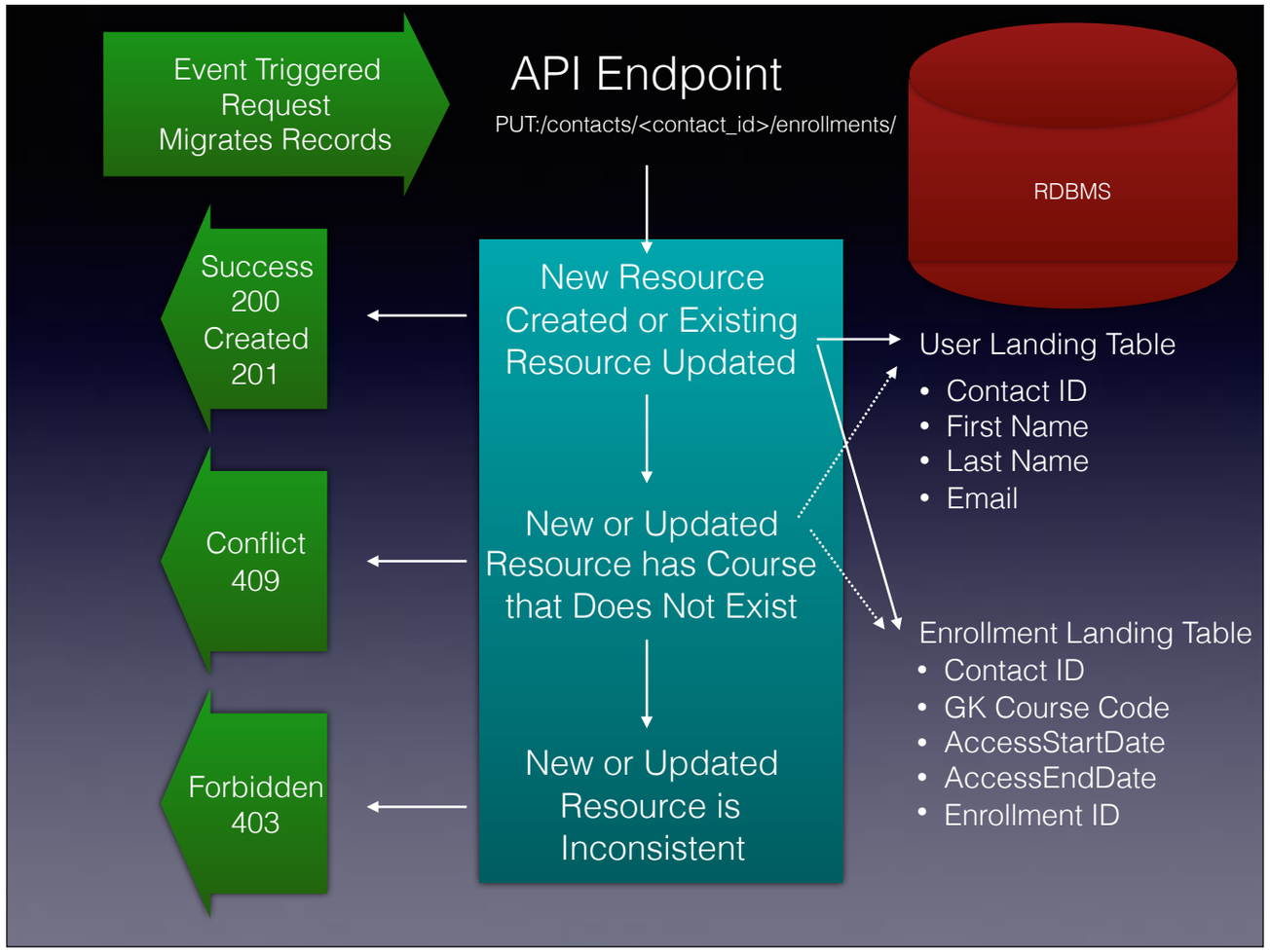
With an understanding of the what and where with regard to , lets look at how a record is received at the Web API. Records arriving at the Web API endpoint are validated [click] and compared to existing resource records in the landing tables. The correct HTTP status code is returned by the Web API, indicating the status of the request to the upstream system, 20X's for success [click] or 40X's [click] for errors. It is possible that for a 409, part of the record was been written to the landing tables successfully [click], the body in the response, details what part of the record caused the conflict. A 403 will not be processed at all, and the response body details what attribute of the resource is causing the fatal error.



With an understanding of the what and where with regard to , lets look at how a record is received at the Web API. Records arriving at the Web API endpoint are validated [click] and compared to existing resource records in the landing tables. The correct HTTP status code is returned by the Web API, indicating the status of the request to the upstream system, 20X's for success [click] or 40X's [click] for errors. It is possible that for a 409, part of the record was been written to the landing tables successfully [click], the body in the response, details what part of the record caused the conflict. A 403 will not be processed at all, and the response body details what attribute of the resource is causing the fatal error.



With an understanding of the what and where with regard to , lets look at how a record is received at the Web API. Records arriving at the Web API endpoint are validated [click] and compared to existing resource records in the landing tables. The correct HTTP status code is returned by the Web API, indicating the status of the request to the upstream system, 20X's for success [click] or 40X's [click] for errors. It is possible that for a 409, part of the record was been written to the landing tables successfully [click], the body in the response, details what part of the record caused the conflict. A 403 will not be processed at all, and the response body details what attribute of the resource is causing the fatal error.



With an understanding of the what and where with regard to , lets look at how a record is received at the Web API. Records arriving at the Web API endpoint are validated [click] and compared to existing resource records in the landing tables. The correct HTTP status code is returned by the Web API, indicating the status of the request to the upstream system, 20X's for success [click] or 40X's [click] for errors. It is possible that for a 409, part of the record was been written to the landing tables successfully [click], the body in the response, details what part of the record caused the conflict. A 403 will not be processed at all, and the response body details what attribute of the resource is causing the fatal error.

Creating & Updating Open edX Students & Enrollments

With the migration of the resources from the Global Knowledge back office to the landing tables in Open edX complete, we'll turn our attention to how we integrated the creation of the native student and enrollment records in Open edX.

Web API App

The Web API as a Django web app, contains URL routing and view components [click], with Global Knowledge business logic [click] encapsulated away from web request and response management. The Open edX core libraries [click] are a dependency at the view and controller level. The processing or attempted processing of the resource determines the response status and how the upstream services will manage the response. The landing tables create a record on Open edX for implementing custom business logic for Global Knowledge, but Open edX native records still need to be created and updated in Open edX's tables. The controllers orchestrate the creation of the Open edX native user and enrollment records.

Web API App

URL Routing

```
from django.contrib.auth.models import User
from django.core.exceptions import ValidationError

from social_apps.django_app.default.models import UserSocialAuth

from rest_framework.views import APIView

from student.roles import GlobalStaff
from student.auth import user_has_role
from student.views import create_account_with_params, AccountValidationError
from openedx.core.lib.exceptions import CourseNotFoundError
from openedx.core.djangoapps.user_api.accounts.api import check_account_exists
from openedx.core.djangoapps.content.course_overviews.models import CourseOverview


def process_resource(request):
    '''Process resource
    [...
    ...


class WebAuthViewAPIView(
    APIView):

    def put(self, request, contact_id, format=None):
        '''Handle requests to migrate records from the back office.
        ...
        ...

        process_request(request.data)
```

The Web API as a Django web app, contains URL routing and view components [click], with Global Knowledge business logic [click] encapsulated away from web request and response management. The Open edX core libraries [click] are a dependency at the view and controller level. The processing or attempted processing of the resource determines the response status and how the upstream services will manage the response. The landing tables create a record on Open edX for implementing custom business logic for Global Knowledge, but Open edX native records still need to be created and updated in Open edX's tables. The controllers orchestrate the creation of the Open edX native user and enrollment records.

Web API App

URL Routing

View

The Web API as a Django web app, contains URL routing and view components [click], with Global Knowledge business logic [click] encapsulated away from web request and response management. The Open edX core libraries [click] are a dependency at the view and controller level. The processing or attempted processing of the resource determines the response status and how the upstream services will manage the response. The landing tables create a record on Open edX for implementing custom business logic for Global Knowledge, but Open edX native records still need to be created and updated in Open edX's tables. The controllers orchestrate the creation of the Open edX native user and enrollment records.

Web API App

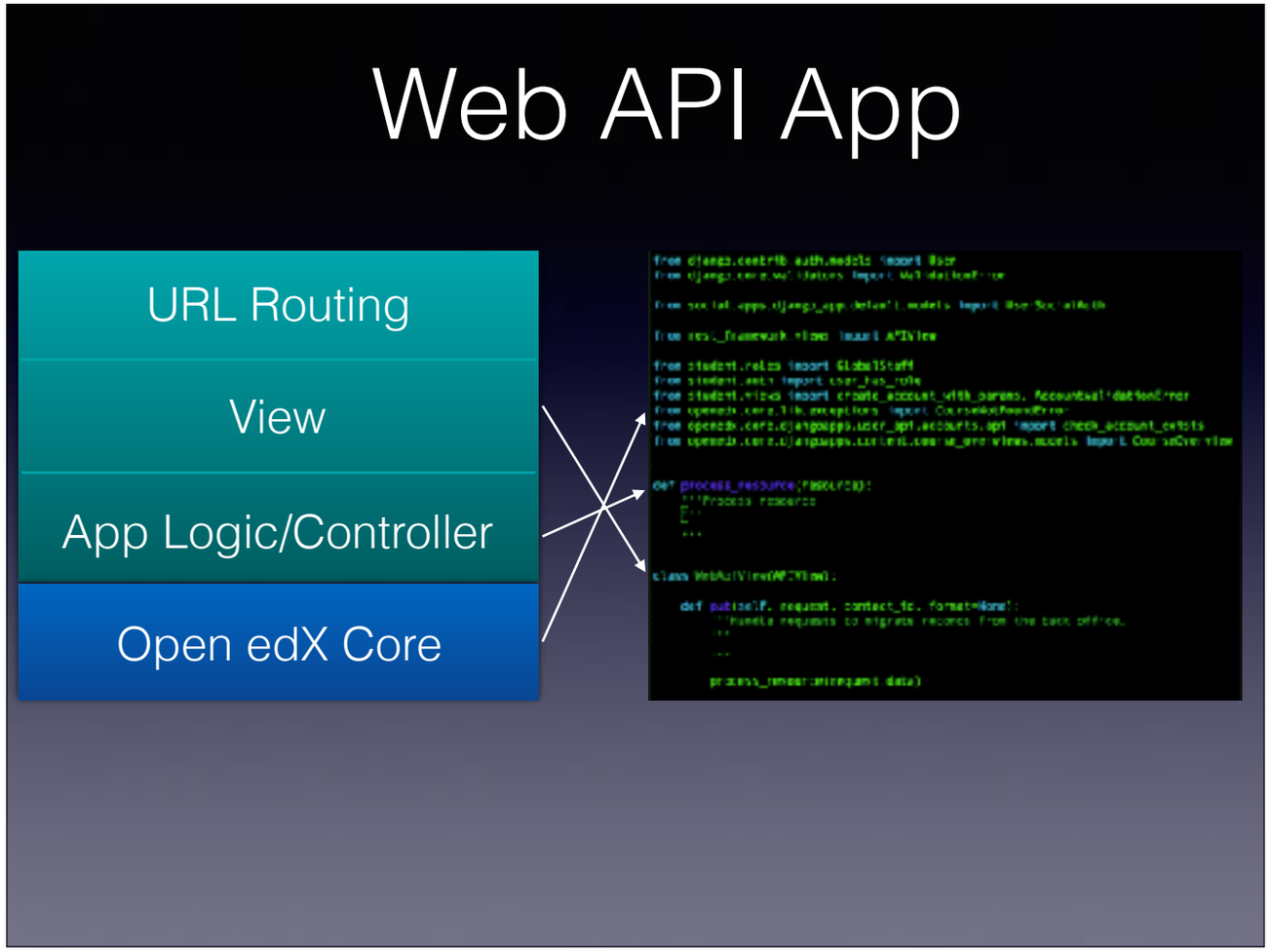
URL Routing

View

App Logic/Controller

The Web API as a Django web app, contains URL routing and view components [click], with Global Knowledge business logic [click] encapsulated away from web request and response management. The Open edX core libraries [click] are a dependency at the view and controller level. The processing or attempted processing of the resource determines the response status and how the upstream services will manage the response. The landing tables create a record on Open edX for implementing custom business logic for Global Knowledge, but Open edX native records still need to be created and updated in Open edX's tables. The controllers orchestrate the creation of the Open edX native user and enrollment records.

Web API App



The Web API as a Django web app, contains URL routing and view components [click], with Global Knowledge business logic [click] encapsulated away from web request and response management. The Open edX core libraries [click] are a dependency at the view and controller level. The processing or attempted processing of the resource determines the response status and how the upstream services will manage the response. The landing tables create a record on Open edX for implementing custom business logic for Global Knowledge, but Open edX native records still need to be created and updated in Open edX's tables. The controllers orchestrate the creation of the Open edX native user and enrollment records.

With Open edX receiving and processing student and enrollment information, the necessary data for implementing additional integrations with Global Knowledge are now available on Open edX. Lets look at some additional Open edX integrations performed at Global Knowledge.

Additional Integrations

With Open edX receiving and processing student and enrollment information, the necessary data for implementing additional integrations with Global Knowledge are now available on Open edX. Lets look at some additional Open edX integrations performed at Global Knowledge.

Some of the additional integrations we've performed at Global Knowledge are…[click through all] Next we will look at APIs for adding Ajax functionality.

- APIs for Adding Functionality via Ajax

Some of the additional integrations we've performed at Global Knowledge are…[click through all] Next we will look at APIs for adding Ajax functionality.

- APIs for Adding Functionality via Ajax
  - Custom Endpoints that Forward Student Communications

Some of the additional integrations we've performed at Global Knowledge are…[click through all] Next we will look at APIs for adding Ajax functionality.

- APIs for Adding Functionality via Ajax
  - Custom Endpoints that Forward Student Communications
    - Mentor Contact Message Relay per Course

Some of the additional integrations we've performed at Global Knowledge are…[click through all] Next we will look at APIs for adding Ajax functionality.

- APIs for Adding Functionality via Ajax
  - Custom Endpoints that Forward Student Communications
    - Mentor Contact Message Relay per Course
    - Support Contact Message Relay

Some of the additional integrations we've performed at Global Knowledge are…[click through all] Next we will look at APIs for adding Ajax functionality.

- APIs for Adding Functionality via Ajax
  - Custom Endpoints that Forward Student Communications
    - Mentor Contact Message Relay per Course
    - Support Contact Message Relay
- Single Sign On via Third Party Authentication Integration

Some of the additional integrations we've performed at Global Knowledge are…[click through all] Next we will look at APIs for adding Ajax functionality.

- APIs for Adding Functionality via Ajax
  - Custom Endpoints that Forward Student Communications
    - Mentor Contact Message Relay per Course
    - Support Contact Message Relay
- Single Sign On via Third Party Authentication Integration
- Custom xblocks

Some of the additional integrations we've performed at Global Knowledge are…[click through all] Next we will look at APIs for adding Ajax functionality.

- APIs for Adding Functionality via Ajax
  - Custom Endpoints that Forward Student Communications
    - Mentor Contact Message Relay per Course
    - Support Contact Message Relay
- Single Sign On via Third Party Authentication Integration
- Custom xblocks
  - Vimeo Player Integration

Some of the additional integrations we've performed at Global Knowledge are…[click through all] Next we will look at APIs for adding Ajax functionality.

- APIs for Adding Functionality via Ajax
  - Custom Endpoints that Forward Student Communications
    - Mentor Contact Message Relay per Course
    - Support Contact Message Relay
- Single Sign On via Third Party Authentication Integration
- Custom xblocks
  - Vimeo Player Integration
  - Custom Labs

Some of the additional integrations we've performed at Global Knowledge are…[click through all] Next we will look at APIs for adding Ajax functionality.

- APIs for Adding Functionality via Ajax
  - Custom Endpoints that Forward Student Communications
    - Mentor Contact Message Relay per Course
    - Support Contact Message Relay
- Single Sign On via Third Party Authentication Integration
- Custom xblocks
  - Vimeo Player Integration
  - Custom Labs
  - Custom Sequencing

Some of the additional integrations we've performed at Global Knowledge are…[click through all] Next we will look at APIs for adding Ajax functionality.

- APIs for Adding Functionality via Ajax
  - Custom Endpoints that Forward Student Communications
    - Mentor Contact Message Relay per Course
    - Support Contact Message Relay
- Single Sign On via Third Party Authentication Integration
- Custom xblocks
  - Vimeo Player Integration
  - Custom Labs
  - Custom Sequencing
  - Custom Matrix

Some of the additional integrations we've performed at Global Knowledge are…[click through all] Next we will look at APIs for adding Ajax functionality.

For two Global Knowledge features we realized that we needed to extend the functionality of existing Open edX pages. We found the best way to go about doing this was to add the necessary data to the pages by adding Javascript to the edX theming templates to make Ajax calls to custom web APIs. [click]

APIs for Ajax Functionality

For two Global Knowledge features we realized that we needed to extend the functionality of existing Open edX pages. We found the best way to go about doing this was to add the necessary data to the pages by adding Javascript to the edX theming templates to make Ajax calls to custom web APIs. [click]

Ajax API Definition

Similar to the web API for users and enrollments we have book markable URLs [click], request data is constrained to JSON [click], and we made idiomatic use of HTTP methods and status codes [click]. A theme is emerging, the extensibility's available through the development of Django apps for the server, and the ability to add functionality on the client side with Open edX's theming drive a lot of Open edX integrations at Global Knowledge.

Ajax API Definition

- Book-Markable URLs: *`/mentor/contacts/<contact_id>/courses/<course_id>/`* and *`/support/contacts/<contact_id>/`*

Similar to the web API for users and enrollments we have book markable URLs [click], request data is constrained to JSON [click], and we made idiomatic use of HTTP methods and status codes [click]. A theme is emerging, the extensibility's available through the development of Django apps for the server, and the ability to add functionality on the client side with Open edX's theming drive a lot of Open edX integrations at Global Knowledge.

Ajax API Definition

- Book-Markable URLs: *`/mentor/contacts/<contact_id>/courses/<course_id>/`* and *`/support/contacts/<contact_id>/`*
- Constrain by Content-Type: *`application/json`*

Similar to the web API for users and enrollments we have book markable URLs [click], request data is constrained to JSON [click], and we made idiomatic use of HTTP methods and status codes [click]. A theme is emerging, the extensibility's available through the development of Django apps for the server, and the ability to add functionality on the client side with Open edX's theming drive a lot of Open edX integrations at Global Knowledge.

Ajax API Definition

- Book-Markable URLs: *`/mentor/contacts/<contact_id>/courses/<course_id>/`* and *`/support/contacts/<contact_id>/`*
- Constrain by Content-Type: *`application/json`*
- Idiomatic Use of HTTP Methods and Status Codes: GET & POST for retrieve and create with codes (200 for retrieved, 201 for created, 401, 404 and 405) used to indicate status of request.

Similar to the web API for users and enrollments we have book markable URLs [click], request data is constrained to JSON [click], and we made idiomatic use of HTTP methods and status codes [click]. A theme is emerging, the extensibility's available through the development of Django apps for the server, and the ability to add functionality on the client side with Open edX's theming drive a lot of Open edX integrations at Global Knowledge.

Ajax API Implementation

Again, similar to the web API implementation, the Ajax APIs are Django 1.8 compliant apps[click], the apps are python package installable [click], the apps integration with Open edX is through configuration [click], the API extends the Open edX URLs under a custom namespace [click], and in this integration use of Django's CSRF token and auth access controls are used for security. [click] Using the existing CSRF and auth support protects the API without additional development work. Which is important because this API sits outside the firewall.

Ajax API Implementation

- Django 1.8 Compliant

Again, similar to the web API implementation, the Ajax APIs are Django 1.8 compliant apps[click], the apps are python package installable [click], the apps integration with Open edX is through configuration [click], the API extends the Open edX URLs under a custom namespace [click], and in this integration use of Django's CSRF token and auth access controls are used for security. [click] Using the existing CSRF and auth support protects the API without additional development work. Which is important because this API sits outside the firewall.

Ajax API Implementation

- Django 1.8 Compliant
- Python Package Installable

Again, similar to the web API implementation, the Ajax APIs are Django 1.8 compliant apps[click], the apps are python package installable [click], the apps integration with Open edX is through configuration [click], the API extends the Open edX URLs under a custom namespace [click], and in this integration use of Django's CSRF token and auth access controls are used for security. [click] Using the existing CSRF and auth support protects the API without additional development work. Which is important because this API sits outside the firewall.

Ajax API Implementation

- Django 1.8 Compliant
- Python Package Installable
- Integration Through Configuration

Again, similar to the web API implementation, the Ajax APIs are Django 1.8 compliant apps[click], the apps are python package installable [click], the apps integration with Open edX is through configuration [click], the API extends the Open edX URLs under a custom namespace [click], and in this integration use of Django's CSRF token and auth access controls are used for security. [click] Using the existing CSRF and auth support protects the API without additional development work. Which is important because this API sits outside the firewall.

Ajax API Implementation

- Django 1.8 Compliant
- Python Package Installable
- Integration Through Configuration
- URL Mapping Extends Open edX URLs Under a `/gk/api/` Namespace

Again, similar to the web API implementation, the Ajax APIs are Django 1.8 compliant apps[click], the apps are python package installable [click], the apps integration with Open edX is through configuration [click], the API extends the Open edX URLs under a custom namespace [click], and in this integration use of Django's CSRF token and auth access controls are used for security. [click] Using the existing CSRF and auth support protects the API without additional development work. Which is important because this API sits outside the firewall.

Ajax API Implementation

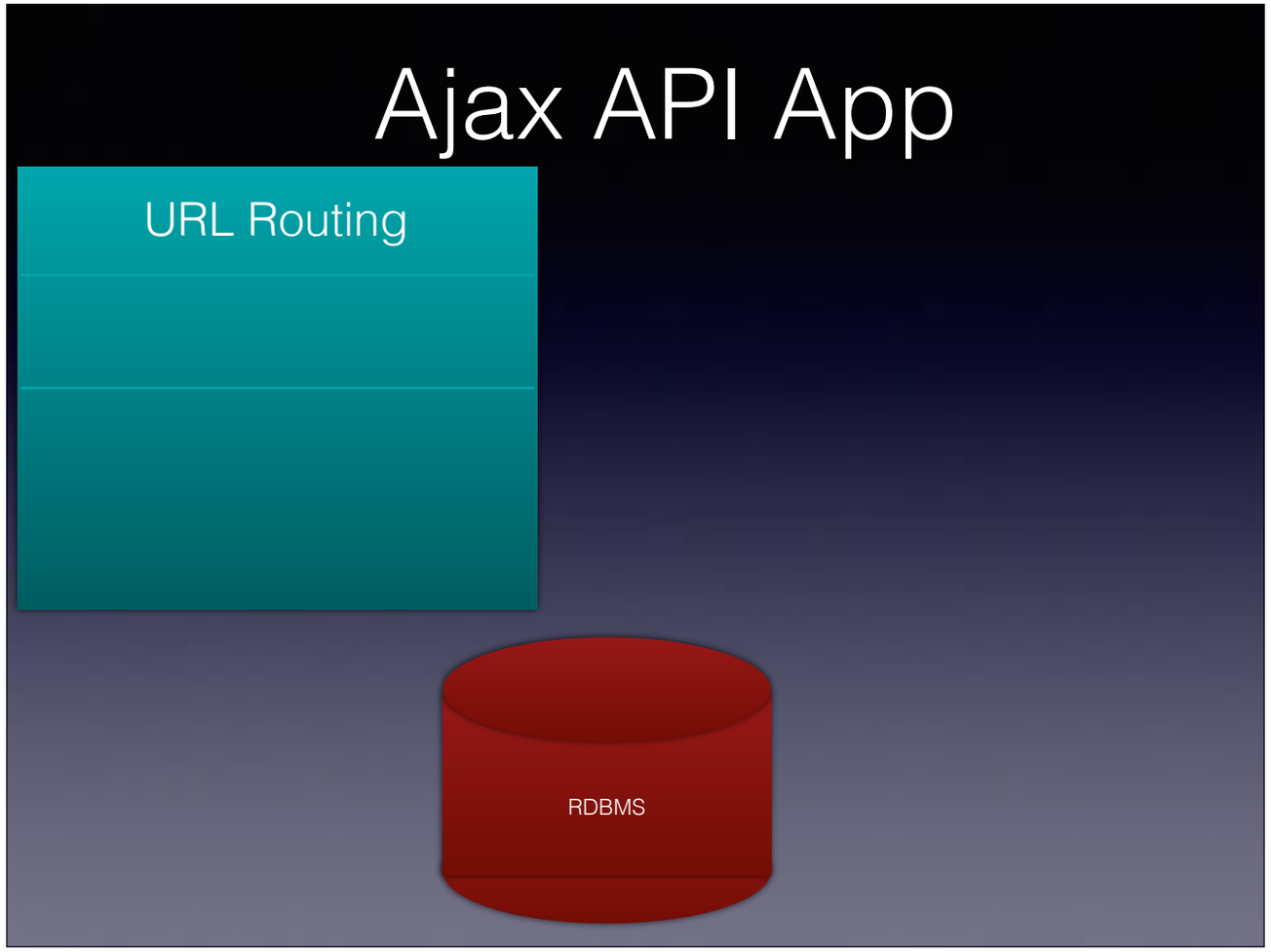
- Django 1.8 Compliant
- Python Package Installable
- Integration Through Configuration
- URL Mapping Extends Open edX URLs Under a `/gk/api/` Namespace
- Uses CSRF Token and Django Auth for Access Control

Again, similar to the web API implementation, the Ajax APIs are Django 1.8 compliant apps[click], the apps are python package installable [click], the apps integration with Open edX is through configuration [click], the API extends the Open edX URLs under a custom namespace [click], and in this integration use of Django's CSRF token and auth access controls are used for security. [click] Using the existing CSRF and auth support protects the API without additional development work. Which is important because this API sits outside the firewall.

The Ajax API has a view layer [click] that manages the request response transaction, a controller layer [click] to perform the logic associated with the request method, and an additional model layer [click] to retrieve mentor information and create records of messages sent to mentors [click].

The mentor data to drive this feature is entered and managed through the Django admin interface. Mentor information is entered per course through the Django admin. Allowing non-developers to change the mentor data as needed through a role audited and password protected interface. Each message sent through the mentor relay is logged via the creation of a MentorMessage instance which is persisted to the Open edX datastore.

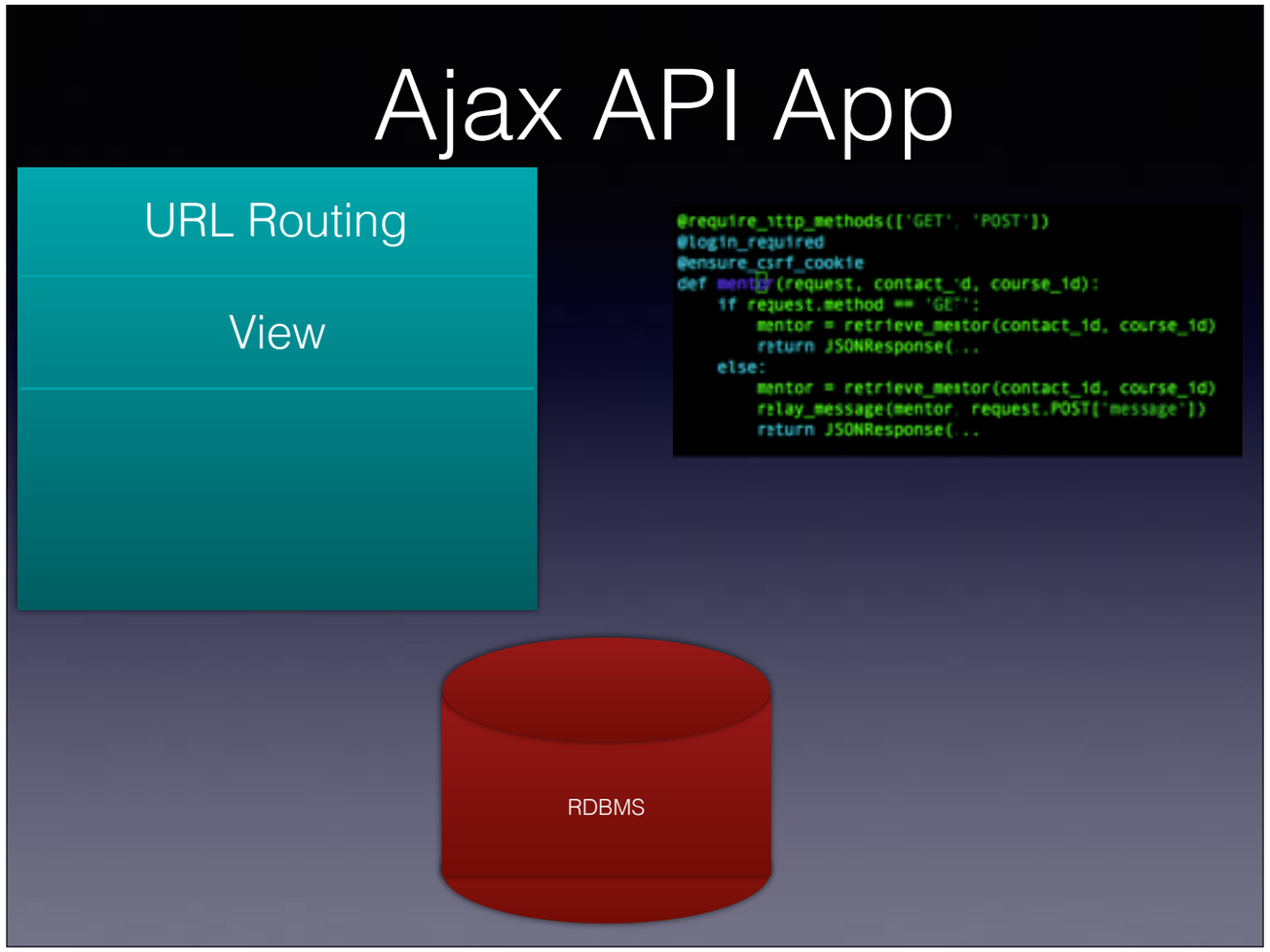
This design allows GK to avoid making changes to the views that service the context to the templates of the Open edX pages, and incurring technical debt from changes to these views as new versions of Open edX are adopted.



The Ajax API has a view layer [click] that manages the request response transaction, a controller layer [click] to perform the logic associated with the request method, and an additional model layer [click] to retrieve mentor information and create records of messages sent to mentors [click].

The mentor data to drive this feature is entered and managed through the Django admin interface. Mentor information is entered per course through the Django admin. Allowing non-developers to change the mentor data as needed through a role audited and password protected interface. Each message sent through the mentor relay is logged via the creation of a MentorMessage instance which is persisted to the Open edX datastore.

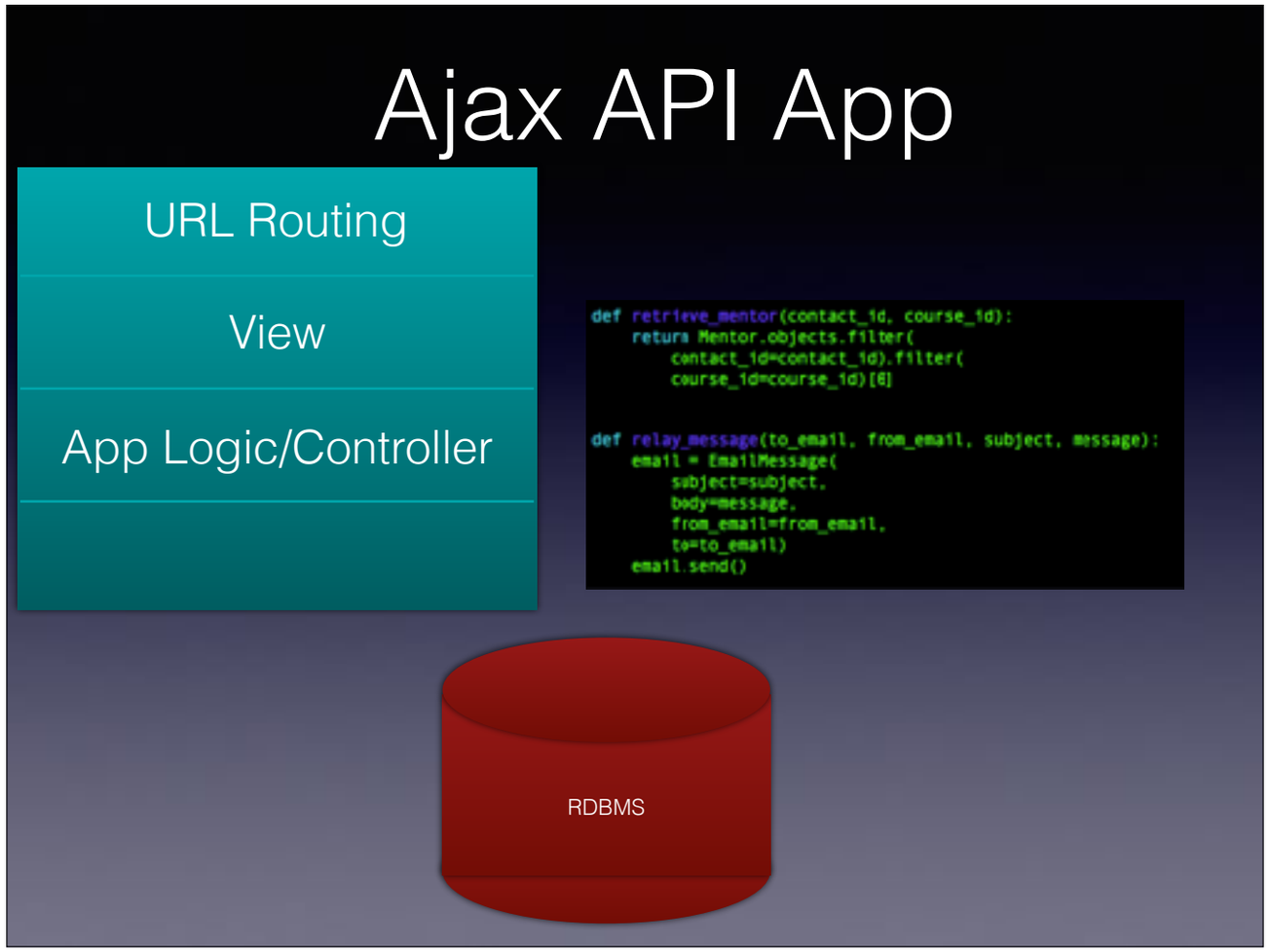
This design allows GK to avoid making changes to the views that service the context to the templates of the Open edX pages, and incurring technical debt from changes to these views as new versions of Open edX are adopted.



The Ajax API has a view layer [click] that manages the request response transaction, a controller layer [click] to perform the logic associated with the request method, and an additional model layer [click] to retrieve mentor information and create records of messages sent to mentors [click].

The mentor data to drive this feature is entered and managed through the Django admin interface. Mentor information is entered per course through the Django admin. Allowing non-developers to change the mentor data as needed through a role audited and password protected interface. Each message sent through the mentor relay is logged via the creation of a MentorMessage instance which is persisted to the Open edX datastore.

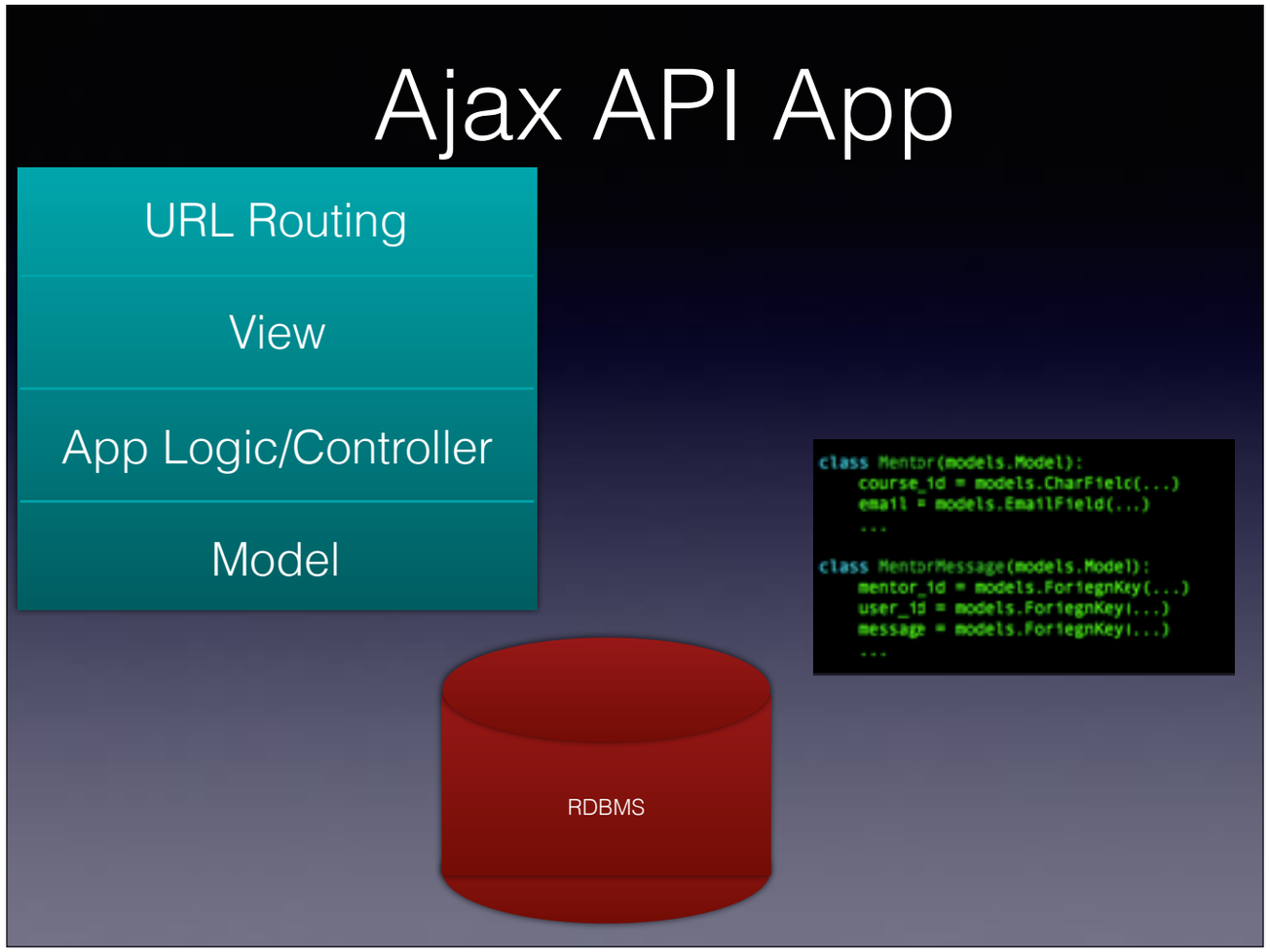
This design allows GK to avoid making changes to the views that service the context to the templates of the Open edX pages, and incurring technical debt from changes to these views as new versions of Open edX are adopted.



The Ajax API has a view layer [click] that manages the request response transaction, a controller layer [click] to perform the logic associated with the request method, and an additional model layer [click] to retrieve mentor information and create records of messages sent to mentors [click].

The mentor data to drive this feature is entered and managed through the Django admin interface. Mentor information is entered per course through the Django admin. Allowing non-developers to change the mentor data as needed through a role audited and password protected interface. Each message sent through the mentor relay is logged via the creation of a MentorMessage instance which is persisted to the Open edX datastore.

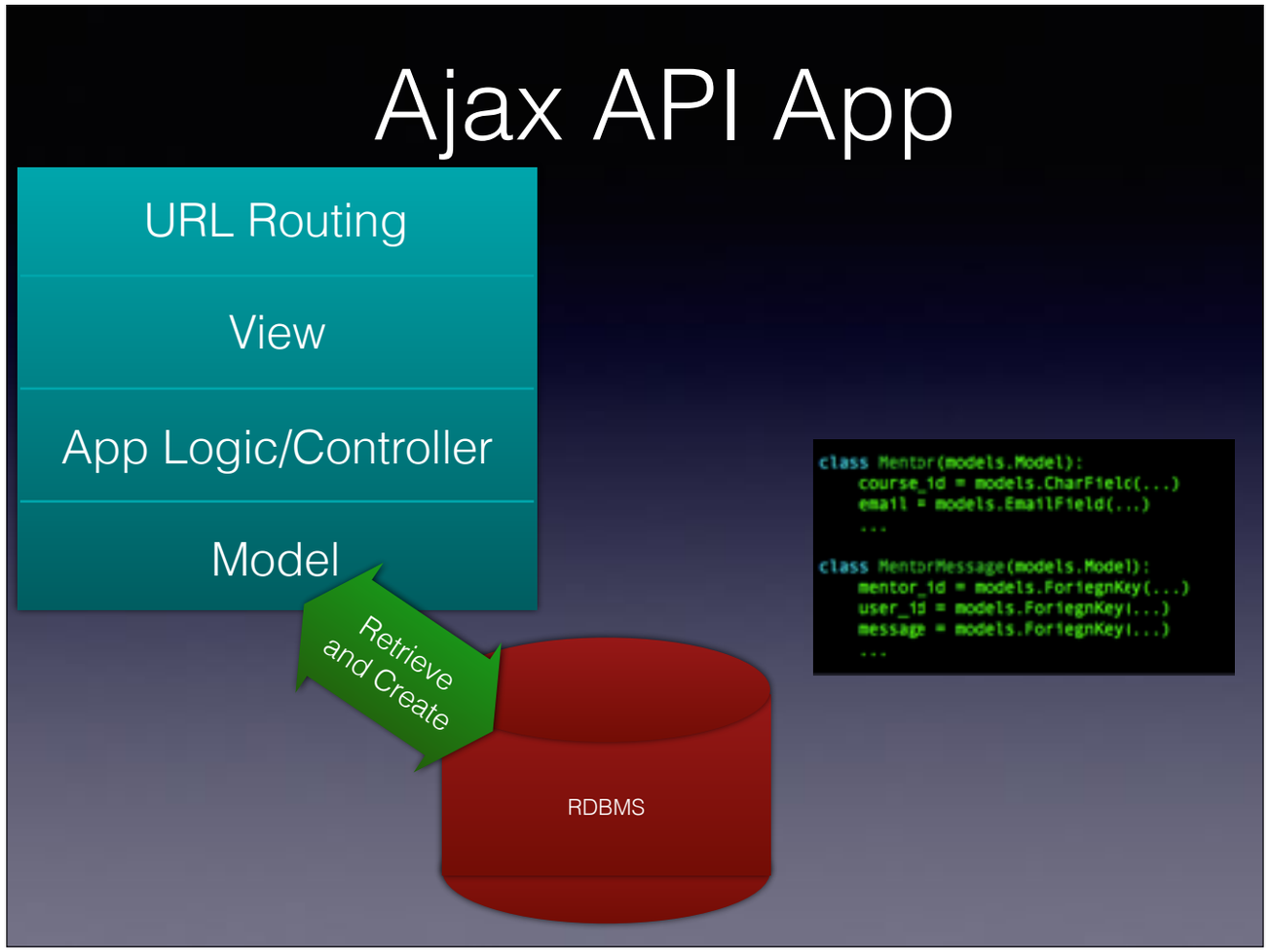
This design allows GK to avoid making changes to the views that service the context to the templates of the Open edX pages, and incurring technical debt from changes to these views as new versions of Open edX are adopted.



The Ajax API has a view layer [click] that manages the request response transaction, a controller layer [click] to perform the logic associated with the request method, and an additional model layer [click] to retrieve mentor information and create records of messages sent to mentors [click].

The mentor data to drive this feature is entered and managed through the Django admin interface. Mentor information is entered per course through the Django admin. Allowing non-developers to change the mentor data as needed through a role audited and password protected interface. Each message sent through the mentor relay is logged via the creation of a MentorMessage instance which is persisted to the Open edX datastore.

This design allows GK to avoid making changes to the views that service the context to the templates of the Open edX pages, and incurring technical debt from changes to these views as new versions of Open edX are adopted.

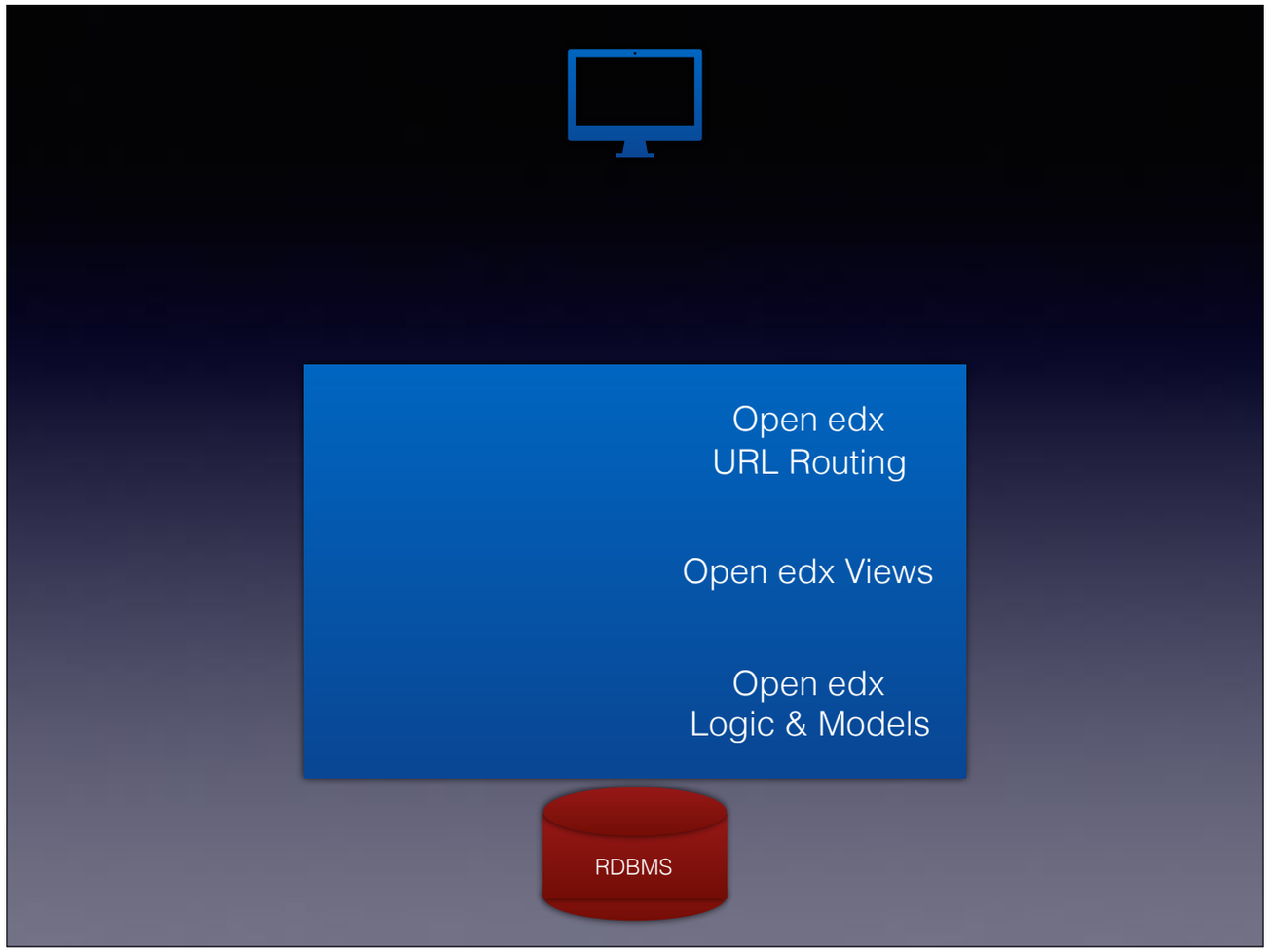


The Ajax API has a view layer [click] that manages the request response transaction, a controller layer [click] to perform the logic associated with the request method, and an additional model layer [click] to retrieve mentor information and create records of messages sent to mentors [click].

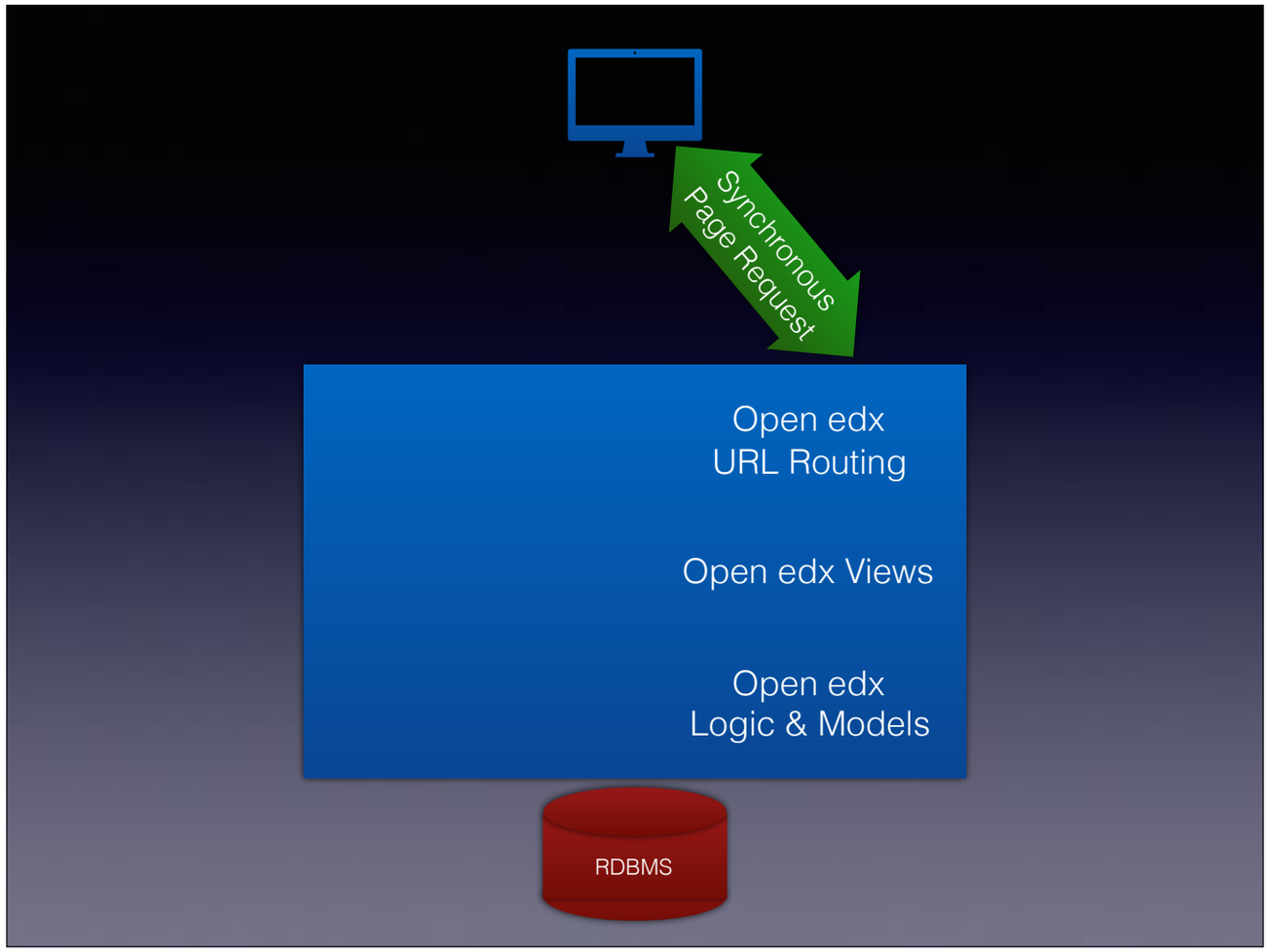
The mentor data to drive this feature is entered and managed through the Django admin interface. Mentor information is entered per course through the Django admin. Allowing non-developers to change the mentor data as needed through a role audited and password protected interface. Each message sent through the mentor relay is logged via the creation of a MentorMessage instance which is persisted to the Open edX datastore.

This design allows GK to avoid making changes to the views that service the context to the templates of the Open edX pages, and incurring technical debt from changes to these views as new versions of Open edX are adopted.

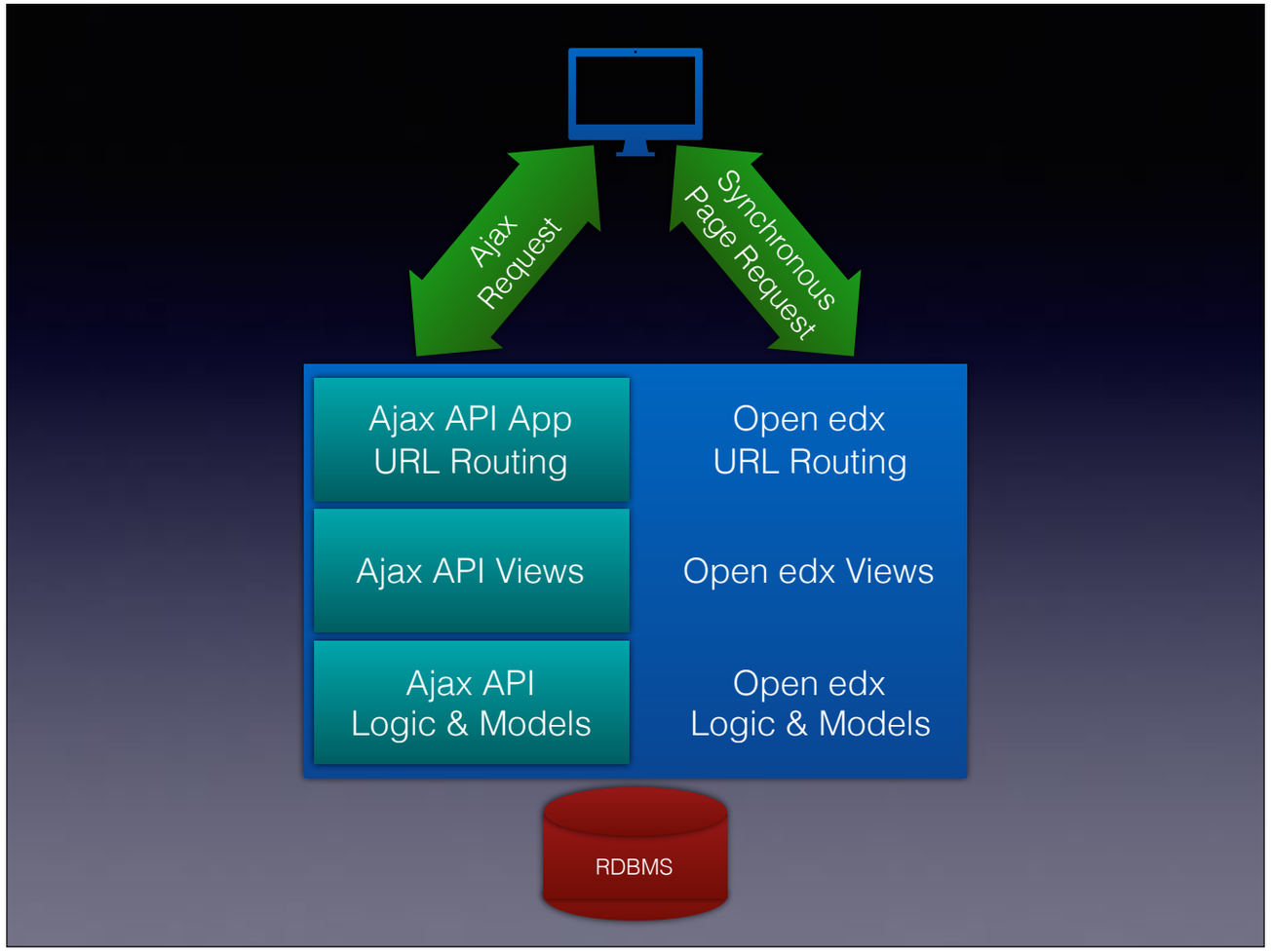
Lets take a look at how the Ajax API side-steps the need to maintain changes to Open edX views. [click] A page customized with the Ajax API is loaded by the client via a synchronous request to Open edX's view. [click] Custom javascript, on the customized page rendered by the synchronous request, accesses the Ajax API [click] to update the page with additional data that was not part of the context used by the Open edX view to build the page during the synchronous request.



Lets take a look at how the Ajax API side-steps the need to maintain changes to Open edX views. [click] A page customized with the Ajax API is loaded by the client via a synchronous request to Open edX's view. [click] Custom javascript, on the customized page rendered by the synchronous request, accesses the Ajax API [click] to update the page with additional data that was not part of the context used by the Open edX view to build the page during the synchronous request.



Lets take a look at how the Ajax API side-steps the need to maintain changes to Open edX views. [click] A page customized with the Ajax API is loaded by the client via a synchronous request to Open edX's view. [click] Custom javascript, on the customized page rendered by the synchronous request, accesses the Ajax API [click] to update the page with additional data that was not part of the context used by the Open edX view to build the page during the synchronous request.

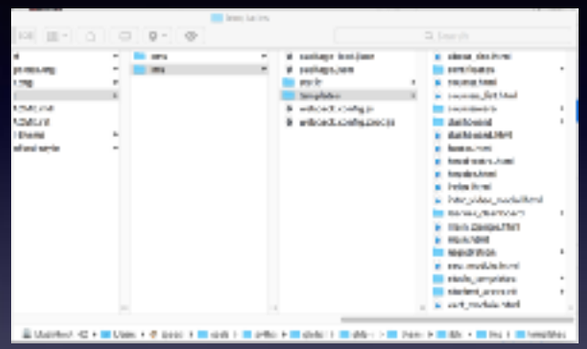


Lets take a look at how the Ajax API side-steps the need to maintain changes to Open edX views. [click] A page customized with the Ajax API is loaded by the client via a synchronous request to Open edX's view. [click] Custom javascript, on the customized page rendered by the synchronous request, accesses the Ajax API [click] to update the page with additional data that was not part of the context used by the Open edX view to build the page during the synchronous request.

With the ability to pull data from the server side through the Ajax API the template will need to be modified to access the data and render the new information. First, find the template to edit in the Open edX source [click], second, add javascript to make the request to the Ajax API [click], third and finally add javascript to handle updates to page DOM [click] received in the response. During these exchanges with the server side, the CSRF token is used to ensure that it is only the javascript from the page created by the Open edX view that is making the request to the server.

Open edX Theme

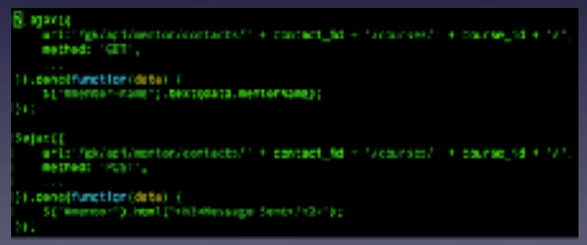
- Find Template in Theme to Edit



With the ability to pull data from the server side through the Ajax API the template will need to be modified to access the data and render the new information. First, find the template to edit in the Open edX source [click], second, add javascript to make the request to the Ajax API [click], third and finally add javascript to handle updates to page DOM [click] received in the response. During these exchanges with the server side, the CSRF token is used to ensure that it is only the javascript from the page created by the Open edX view that is making the request to the server.

Open edX Theme

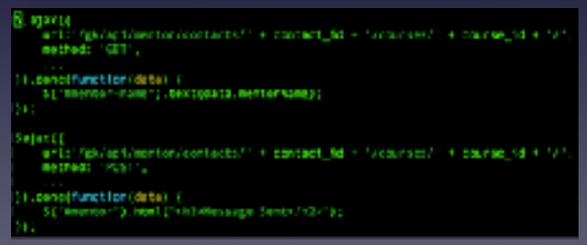
- Find Template in Theme to Edit
- Add Javascript Making Ajax Call to the API



With the ability to pull data from the server side through the Ajax API the template will need to be modified to access the data and render the new information. First, find the template to edit in the Open edX source [click], second, add javascript to make the request to the Ajax API [click], third and finally add javascript to handle updates to page DOM [click] received in the response. During these exchanges with the server side, the CSRF token is used to ensure that it is only the javascript from the page created by the Open edX view that is making the request to the server.

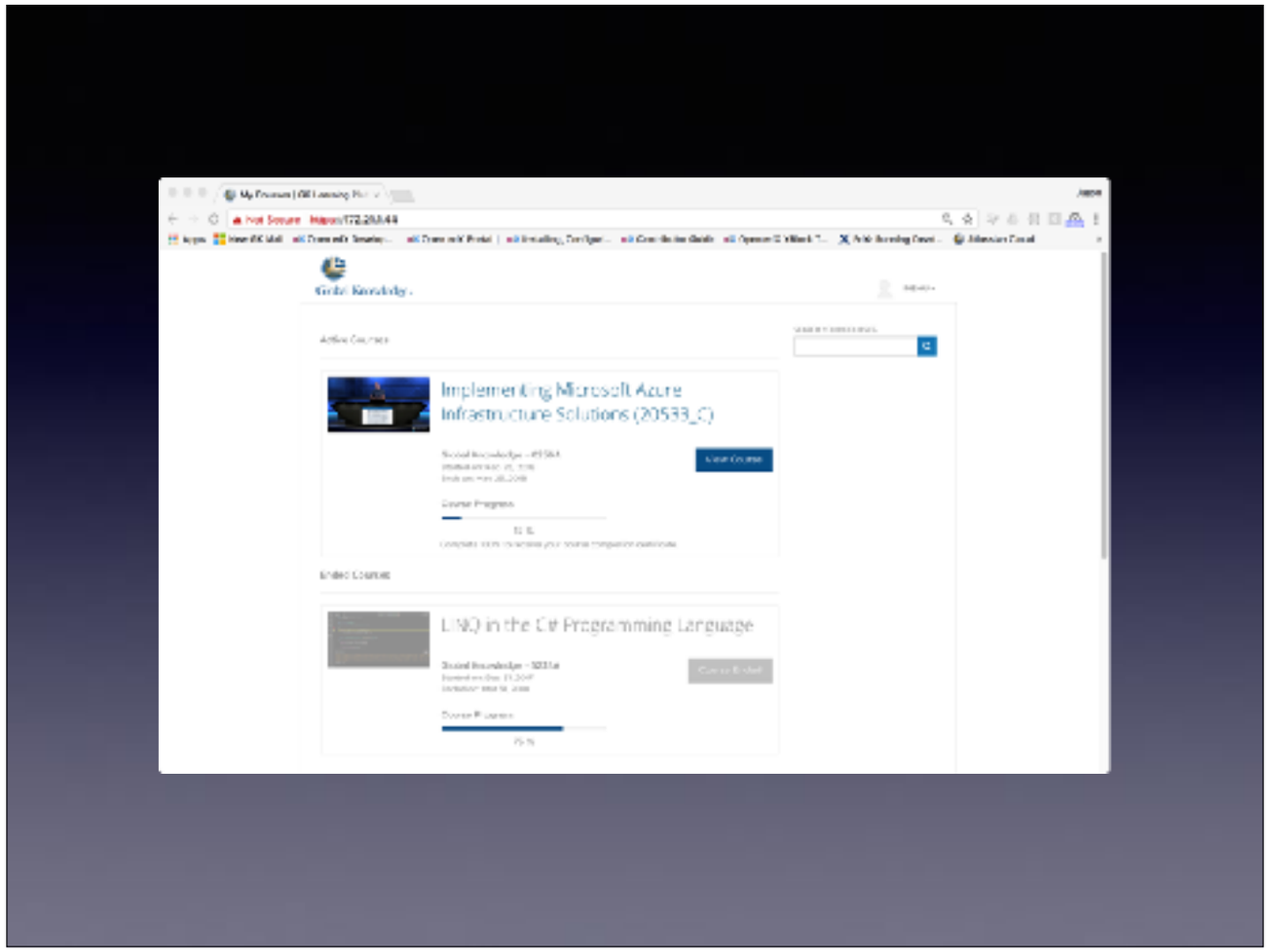
Open edX Theme

- Find Template in Theme to Edit
- Add Javascript Making Ajax Call to the API
- Add Javascript to Update the DOM



With the ability to pull data from the server side through the Ajax API the template will need to be modified to access the data and render the new information. First, find the template to edit in the Open edX source [click], second, add javascript to make the request to the Ajax API [click], third and finally add javascript to handle updates to page DOM [click] received in the response. During these exchanges with the server side, the CSRF token is used to ensure that it is only the javascript from the page created by the Open edX view that is making the request to the server.

One key location in Open edX where we have made use of this technique ,to great success, is the My Courses page. We've added the ability to display more detailed progress information with aid from IBL, and created the ability to add and remove courses from the My Course page based on business rules at Global Knowledge, such as date based enrollment windows, and course enrollment cancelations.

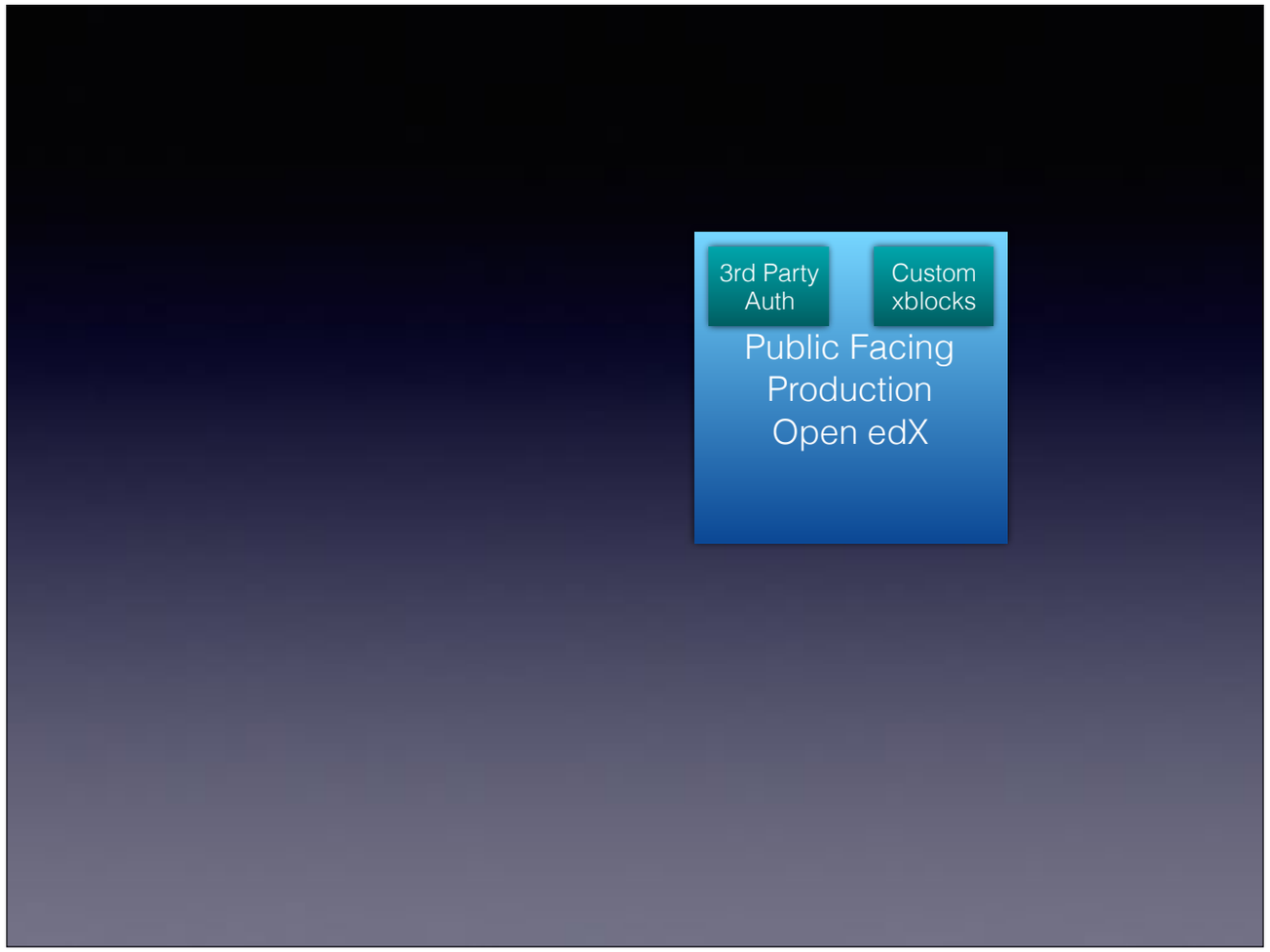


One key location in Open edX where we have made use of this technique ,to great success, is the My Courses page. We've added the ability to display more detailed progress information with aid from IBL, and created the ability to add and remove courses from the My Course page based on business rules at Global Knowledge, such as date based enrollment windows, and course enrollment cancelations.

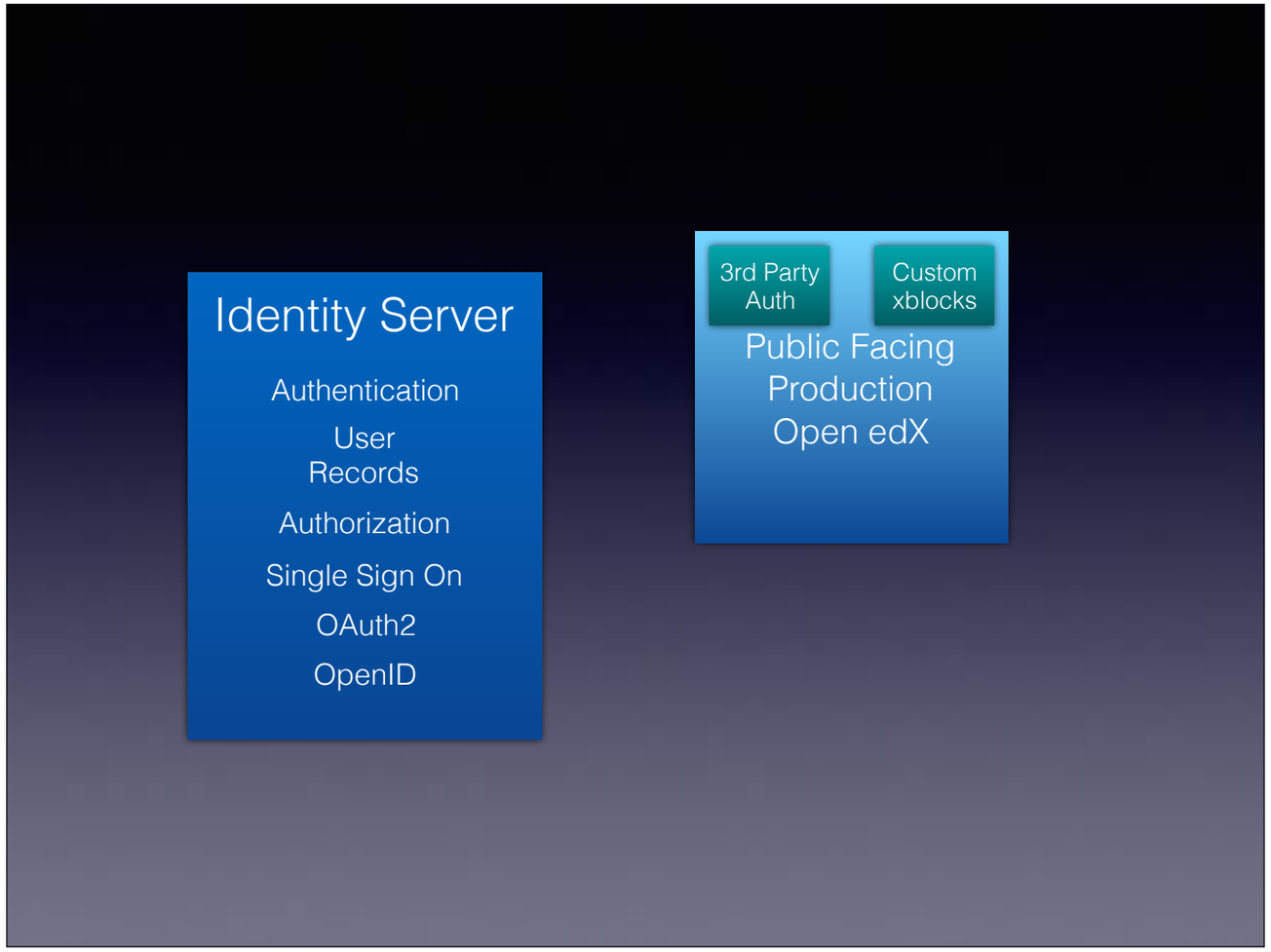
Another key integration at GK is single sign on. [click] Allowing users with accounts on other Global Knowledge systems to user their existing credentials to gain access to our Open edX implementation.

Third Party Authentication & Single Sign On

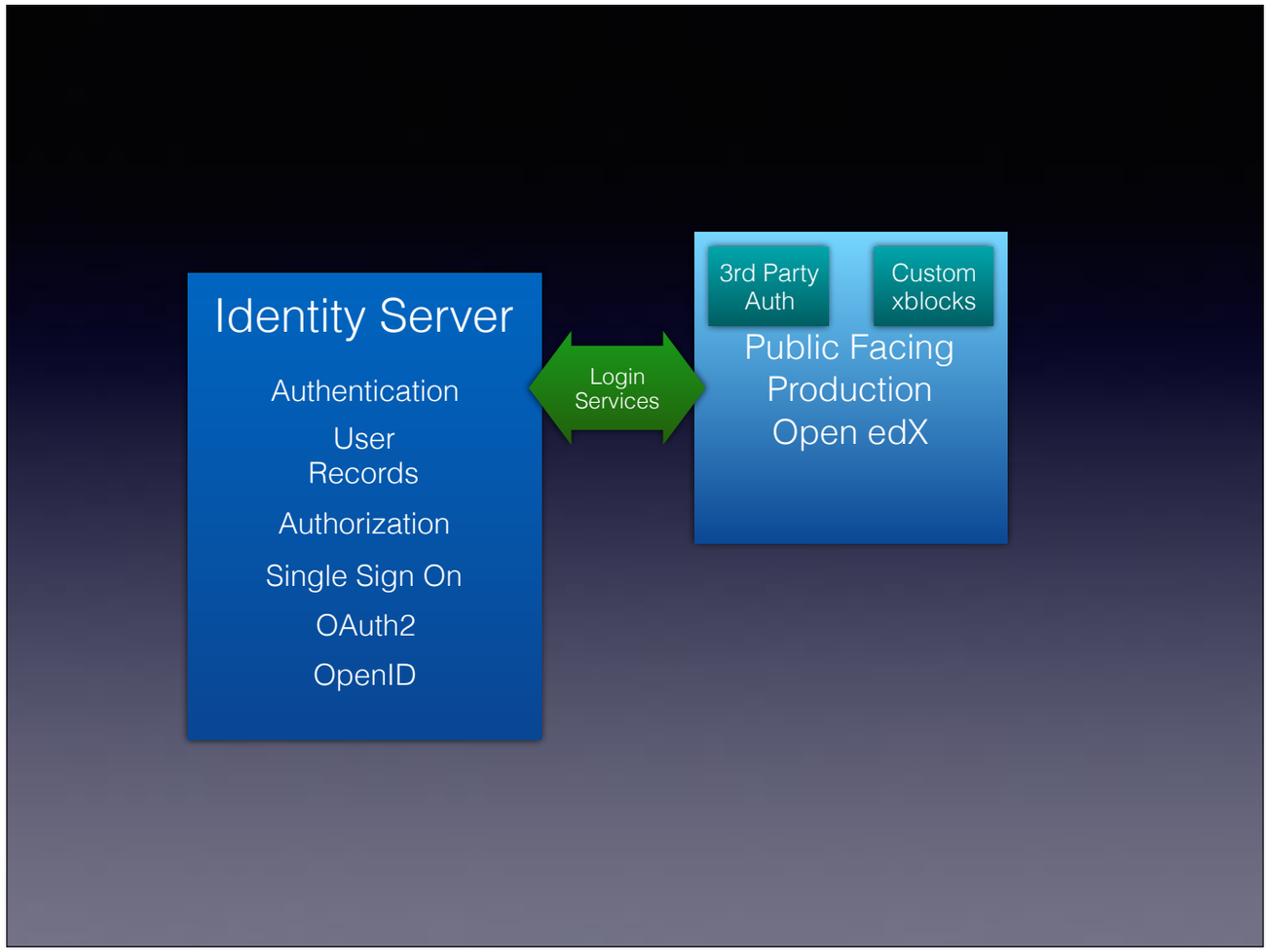
Another key integration at GK is single sign on. [click] Allowing users with accounts on other Global Knowledge systems to user their existing credentials to gain access to our Open edX implementation.



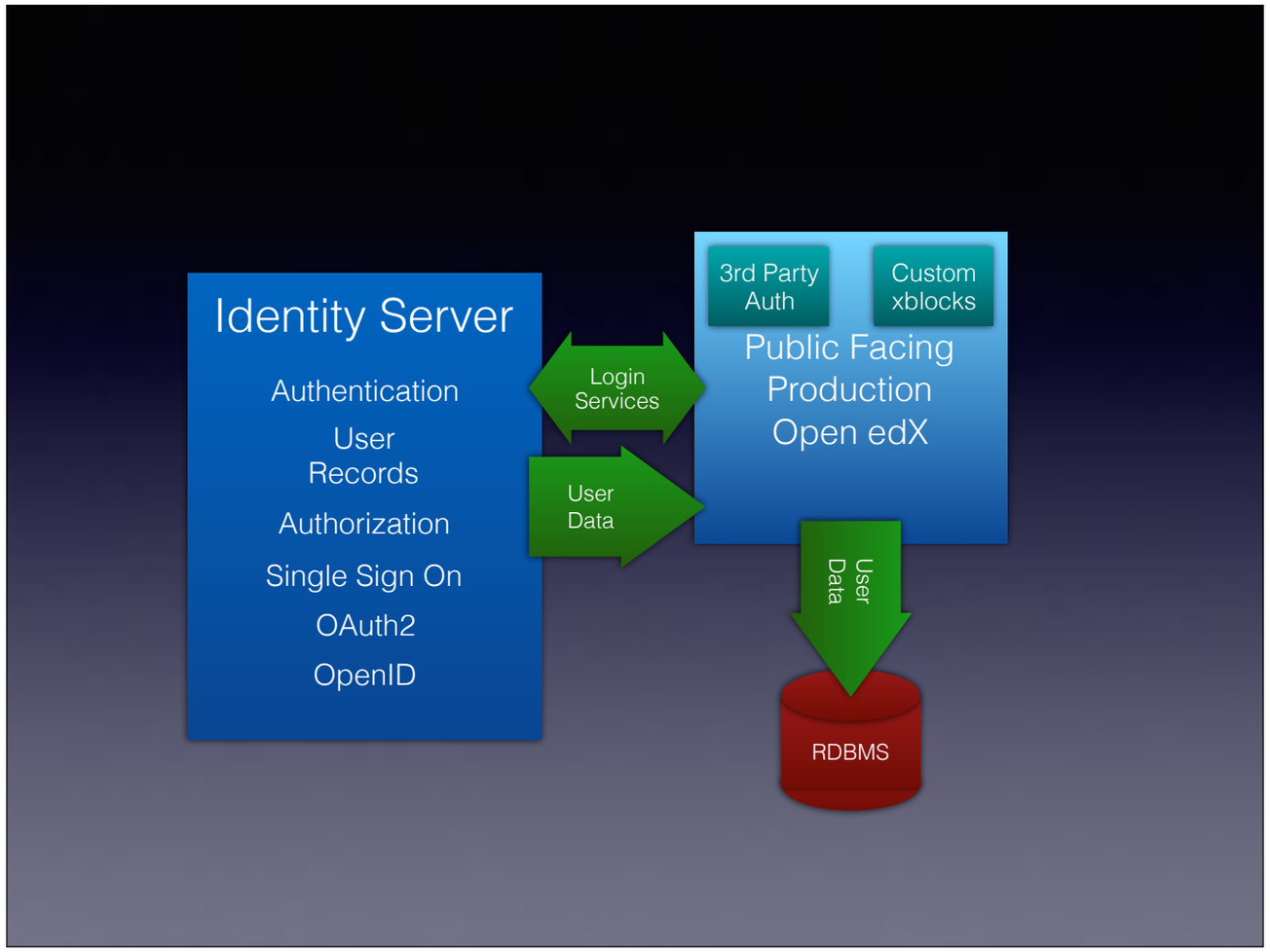
Login authentication authorization for Open edX at GK is delegated to an identity server. [click] When a user attempts to login to Open edX, that user is redirected to our identity server and prompted to authorize with their Global Knowledge credentials. Once authenticated with identity server the user is redirected back to Open edX with their user data, and link is created between Open edX and the Identity Server for that user's account. [click]



Login authentication authorization for Open edX at GK is delegated to an identity server. [click] When a user attempts to login to Open edX, that user is redirected to our identity server and prompted to authorize with their Global Knowledge credentials. Once authenticated with identity server the user is redirected back to Open edX with their user data, and link is created between Open edX and the Identity Server for that user's account. [click]



Login authentication authorization for Open edX at GK is delegated to an identity server. [click] When a user attempts to login to Open edX, that user is redirected to our identity server and prompted to authorize with their Global Knowledge credentials. Once authenticated with identity server the user is redirected back to Open edX with their user data, and link is created between Open edX and the Identity Server for that user's account. [click]



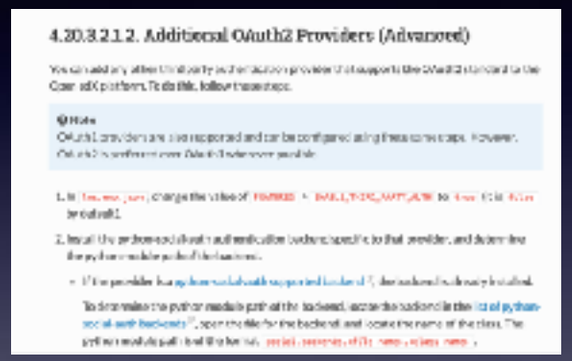
Login authentication authorization for Open edX at GK is delegated to an identity server. [click] When a user attempts to login to Open edX, that user is redirected to our identity server and prompted to authorize with their Global Knowledge credentials. Once authenticated with identity server the user is redirected back to Open edX with their user data, and link is created between Open edX and the Identity Server for that user's account. [click]

Single Sign On

Global Knowledge’s Single-Sign-On implementation depends on two technologies that Open edX provides support for. Third party authorization via additional OAuth2 providers [click], and the transfer of data during third party authorization to create and link accounts using the Open ID standard for claims[click].

Single Sign On

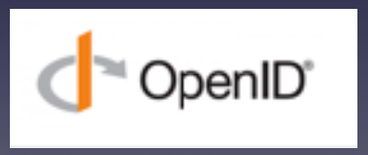
- Open edX Third Party Authentication for Additional OAuth2 Providers



Global Knowledge's Single-Sign-On implementation depends on two technologies that Open edX provides support for. Third party authorization via additional OAuth2 providers [click], and the transfer of data during third party authorization to create and link accounts using the Open ID standard for claims[click].

Single Sign On

- Open edX Third Party Authentication for Additional OAuth2 Providers
- Leverage the OpenID Standard for Transferring User Data During Account Linking and Auth



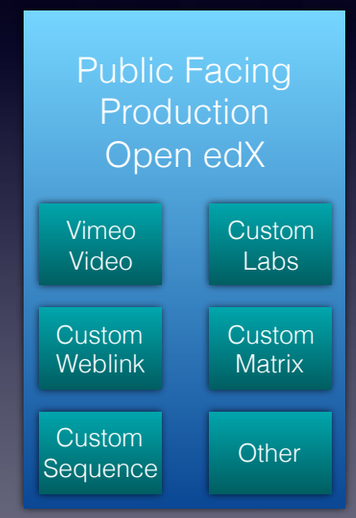
Global Knowledge's Single-Sign-On implementation depends on two technologies that Open edX provides support for. Third party authorization via additional OAuth2 providers [click], and the transfer of data during third party authorization to create and link accounts using the Open ID standard for claims[click].

Not least of all is Global Knowledge’s use of custom xblocks for integration of Open edX with Global Knowledge’s learning assets. Open edX’s support for the standard for portable learning units as xblocks, allows for these integrations. [click]

Custom xblocks

Not least of all is Global Knowledge’s use of custom xblocks for integration of Open edX with Global Knowledge’s learning assets. Open edX’s support for the standard for portable learning units as xblocks, allows for these integrations. [click]

Custom xblocks



The custom xblock implementations at Global Knowledge are; Vimeo video [click], custom labs [click], custom weblink [click], custom matrix [click], custom sequences. The xblock standard also leaves the possibility for countless other integration as Global Knowledge continues to add and improve course content.

Custom xblocks

- Vimeo Video

The custom xblock implementations at Global Knowledge are; Vimeo video [click], custom labs [click], custom weblink [click], custom matrix [click], custom sequences. The xblock standard also leaves the possibility for countless other integration as Global Knowledge continues to add and improve course content.

Custom xblocks

- Vimeo Video
- Custom Labs

The custom xblock implementations at Global Knowledge are; Vimeo video [click], custom labs [click], custom weblink [click], custom matrix [click], custom sequences. The xblock standard also leaves the possibility for countless other integration as Global Knowledge continues to add and improve course content.

Custom xblocks

- Vimeo Video
- Custom Labs
- Custom Weblink

The custom xblock implementations at Global Knowledge are; Vimeo video [click], custom labs [click], custom weblink [click], custom matrix [click], custom sequences. The xblock standard also leaves the possibility for countless other integration as Global Knowledge continues to add and improve course content.

Custom xblocks

- Vimeo Video
- Custom Labs
- Custom Weblink
- Custom Matrix

The custom xblock implementations at Global Knowledge are; Vimeo video [click], custom labs [click], custom weblink [click], custom matrix [click], custom sequences. The xblock standard also leaves the possibility for countless other integration as Global Knowledge continues to add and improve course content.

Custom xblocks

- Vimeo Video
- Custom Labs
- Custom Weblink
- Custom Matrix
- Custom Sequences

The custom xblock implementations at Global Knowledge are; Vimeo video [click], custom labs [click], custom weblink [click], custom matrix [click], custom sequences. The xblock standard also leaves the possibility for countless other integration as Global Knowledge continues to add and improve course content.

Custom xblocks

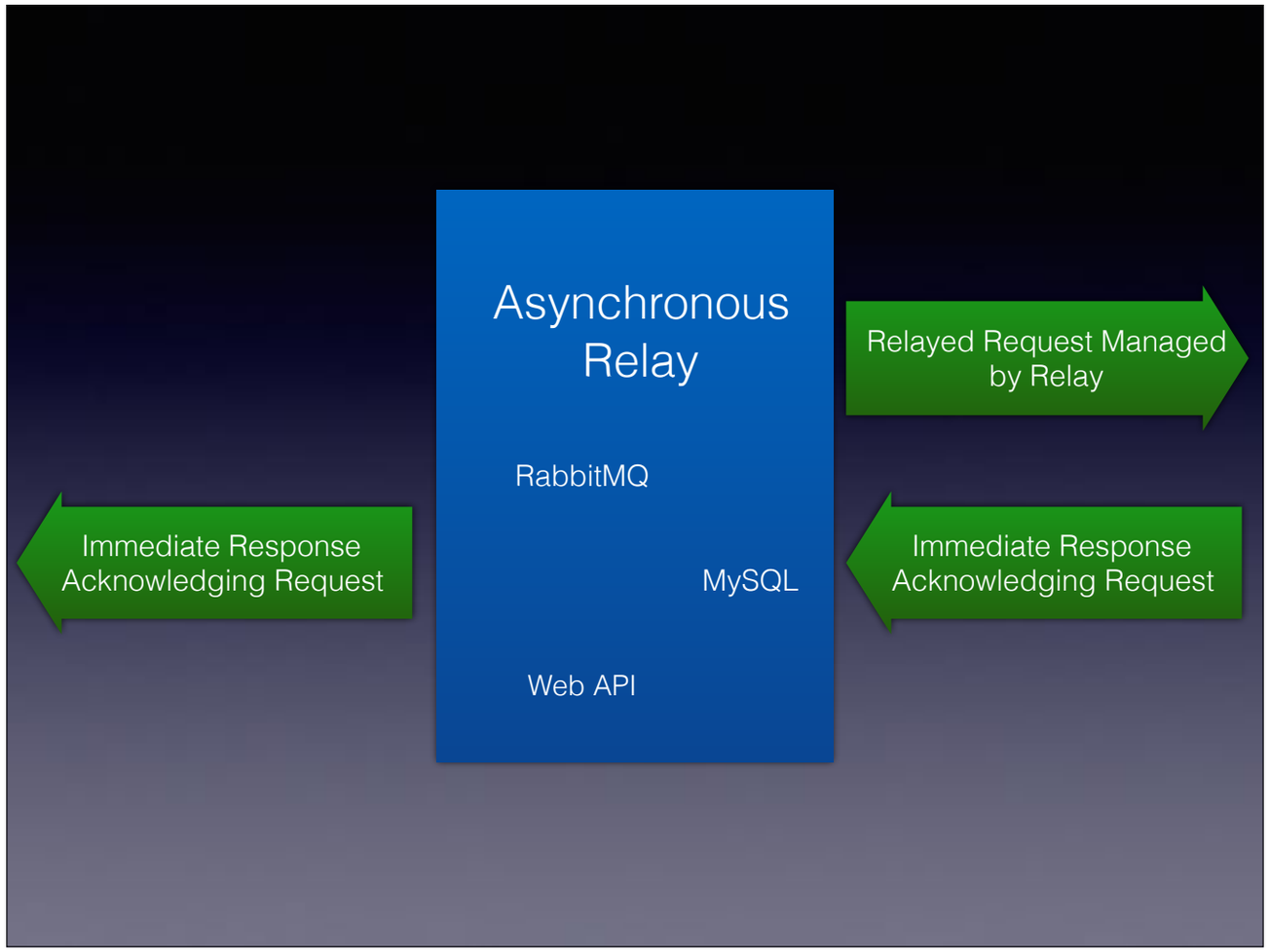
- Vimeo Video
- Custom Labs
- Custom Weblink
- Custom Matrix
- Custom Sequences
- Other

The custom xblock implementations at Global Knowledge are; Vimeo video [click], custom labs [click], custom weblink [click], custom matrix [click], custom sequences. The xblock standard also leaves the possibility for countless other integration as Global Knowledge continues to add and improve course content.

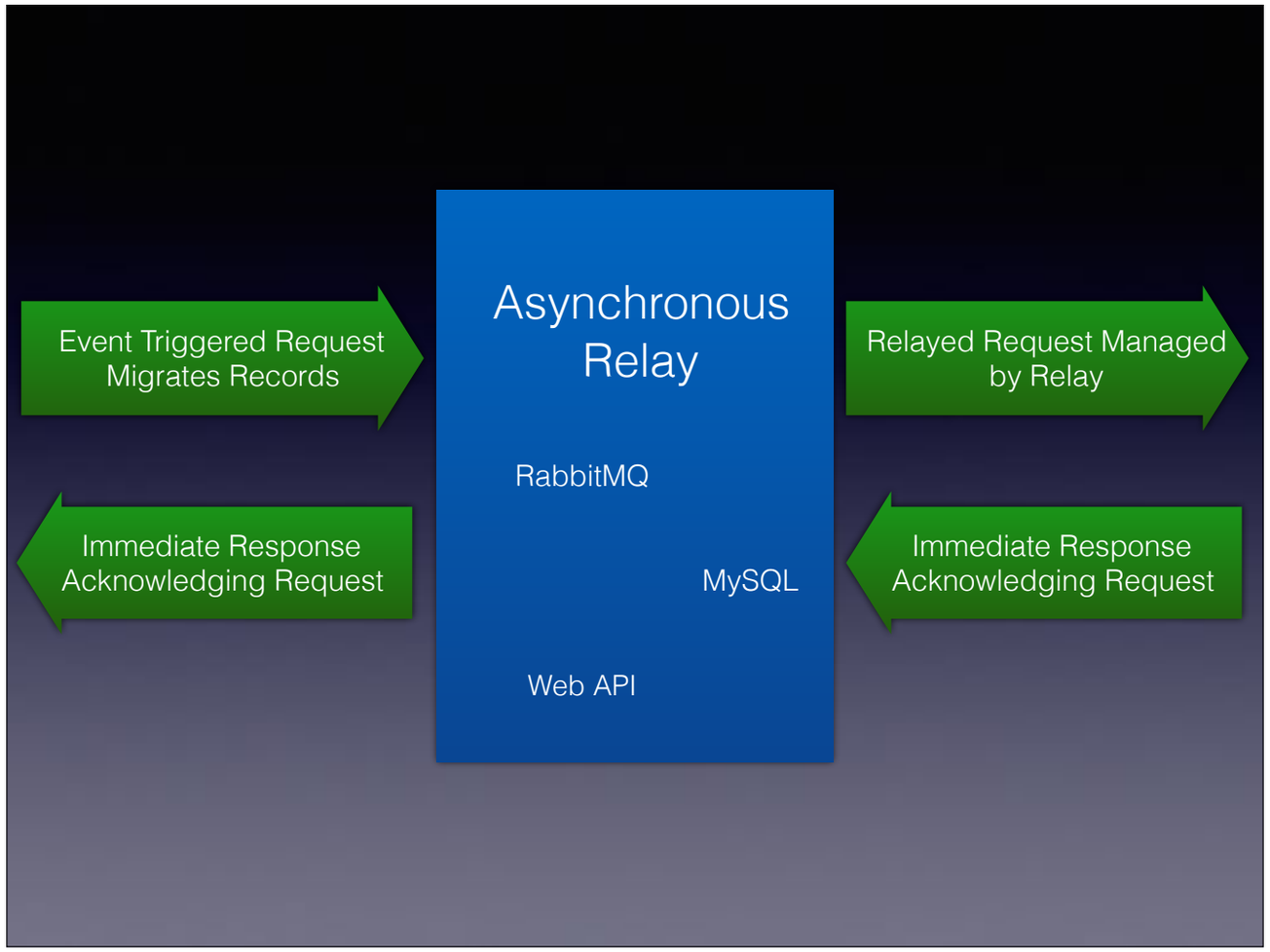
Questions
&
Comments

Thank you for your time and the opportunity to present here at Open edX 2018. I'll now open up the session for questions and comments..

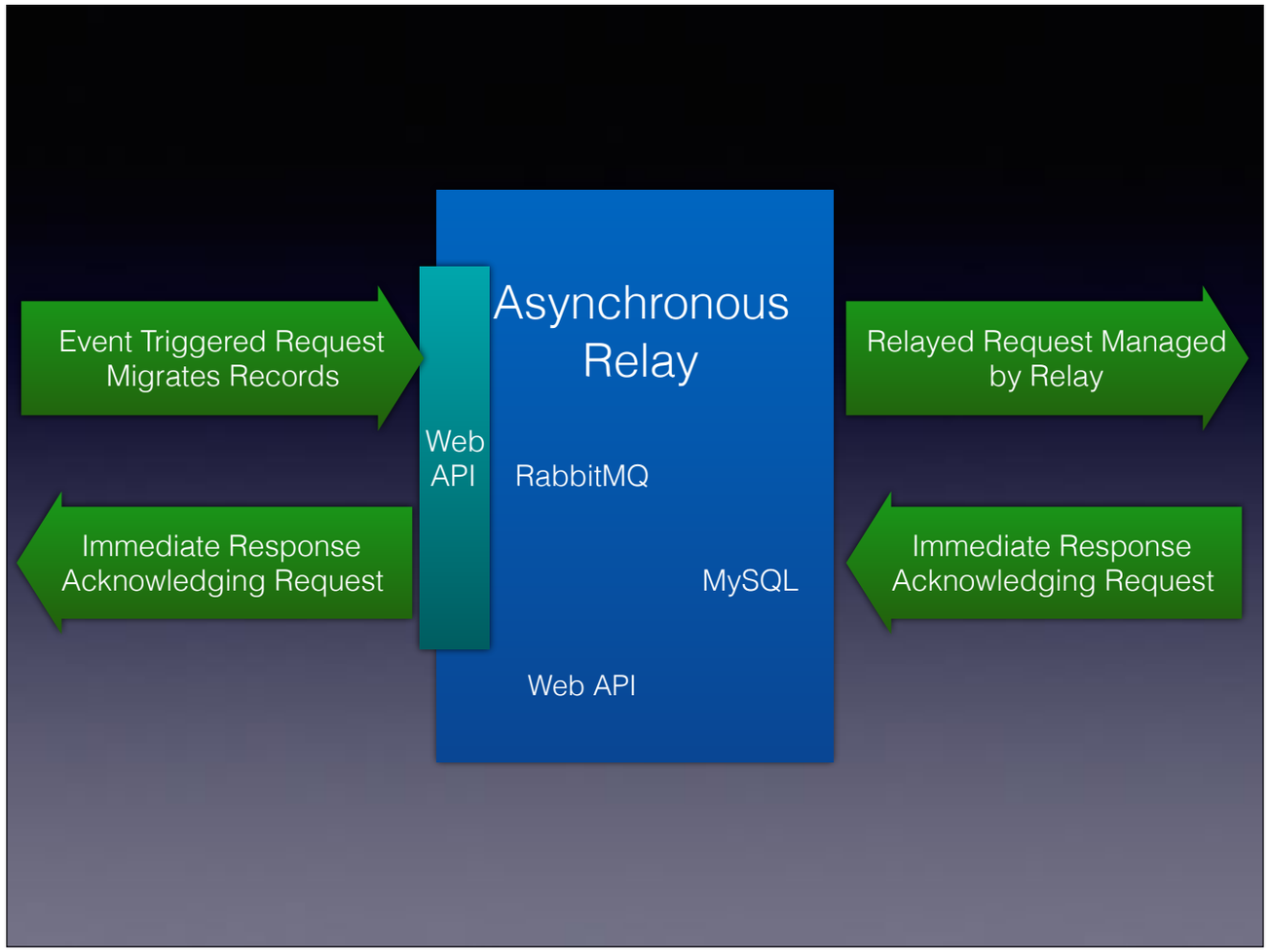
Additional Slides



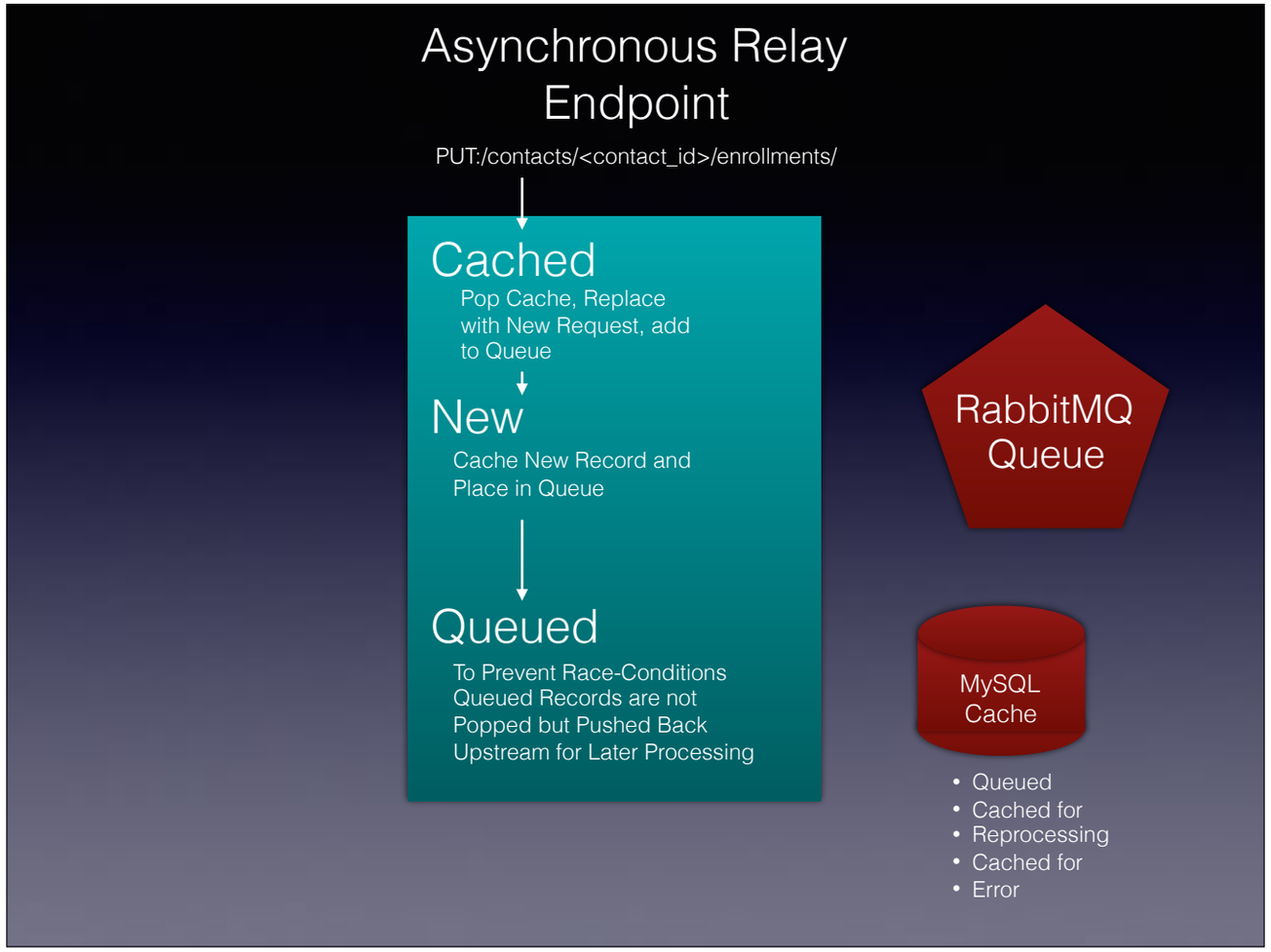
Between the back office and the private production instance of Open edX exists an Asynchronous Message Relay. The Asynchronous relay ensures a timely response to the back office push of data to Open edX, allowing these back office systems to continue without blocking for a response. The asynchronous relay then manages the final push of the data to Open edX, cycling the request through a queue and cash that leaves a record trail of the number of attempts needed to push the data, or the final error state if the push was unsuccessful.



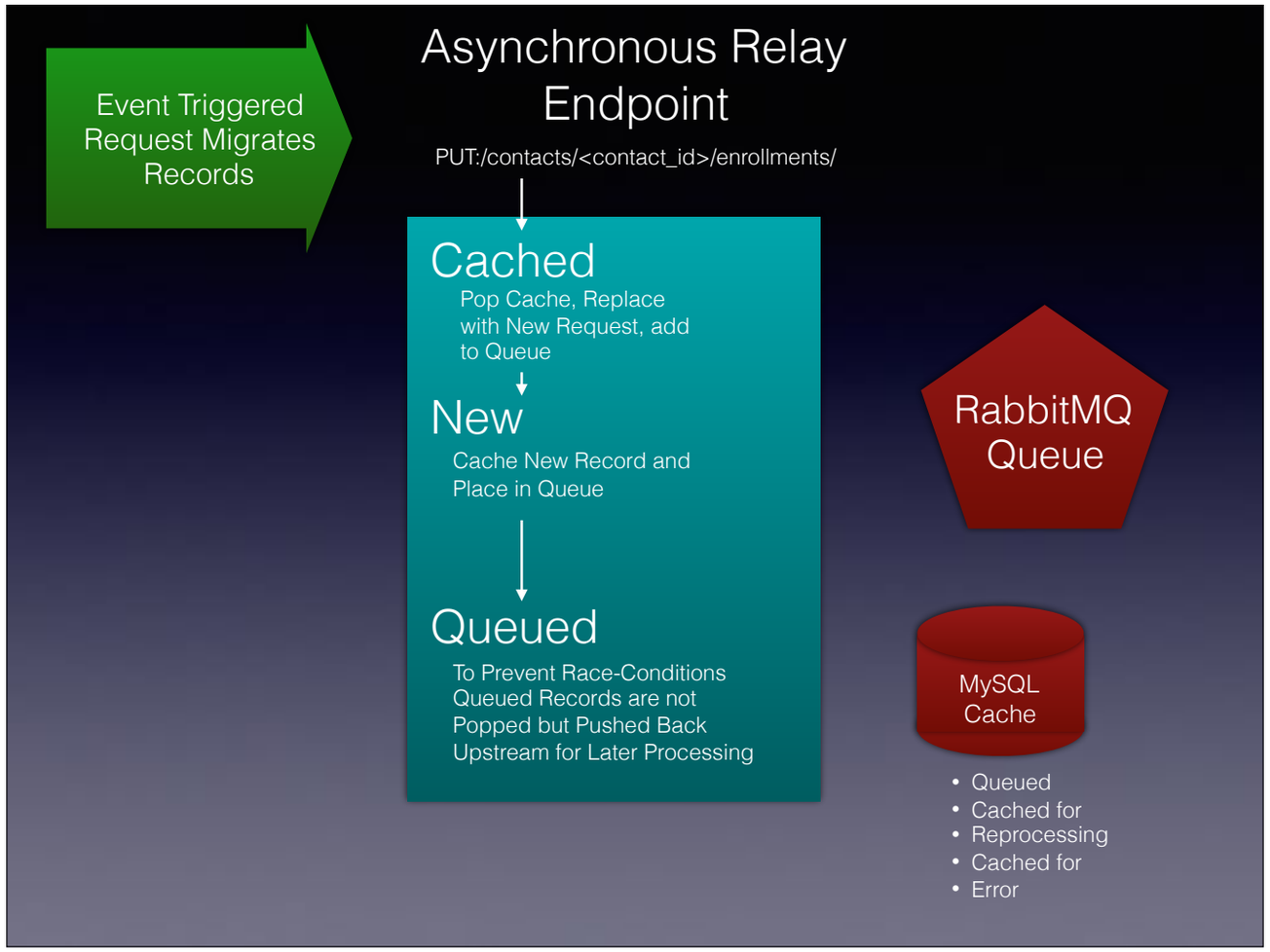
Between the back office and the private production instance of Open edX exists an Asynchronous Message Relay. The Asynchronous relay ensures a timely response to the back office push of data to Open edX, allowing these back office systems to continue without blocking for a response. The asynchronous relay then manages the final push of the data to Open edX, cycling the request through a queue and cash that leaves a record trail of the number of attempts needed to push the data, or the final error state if the push was unsuccessful.



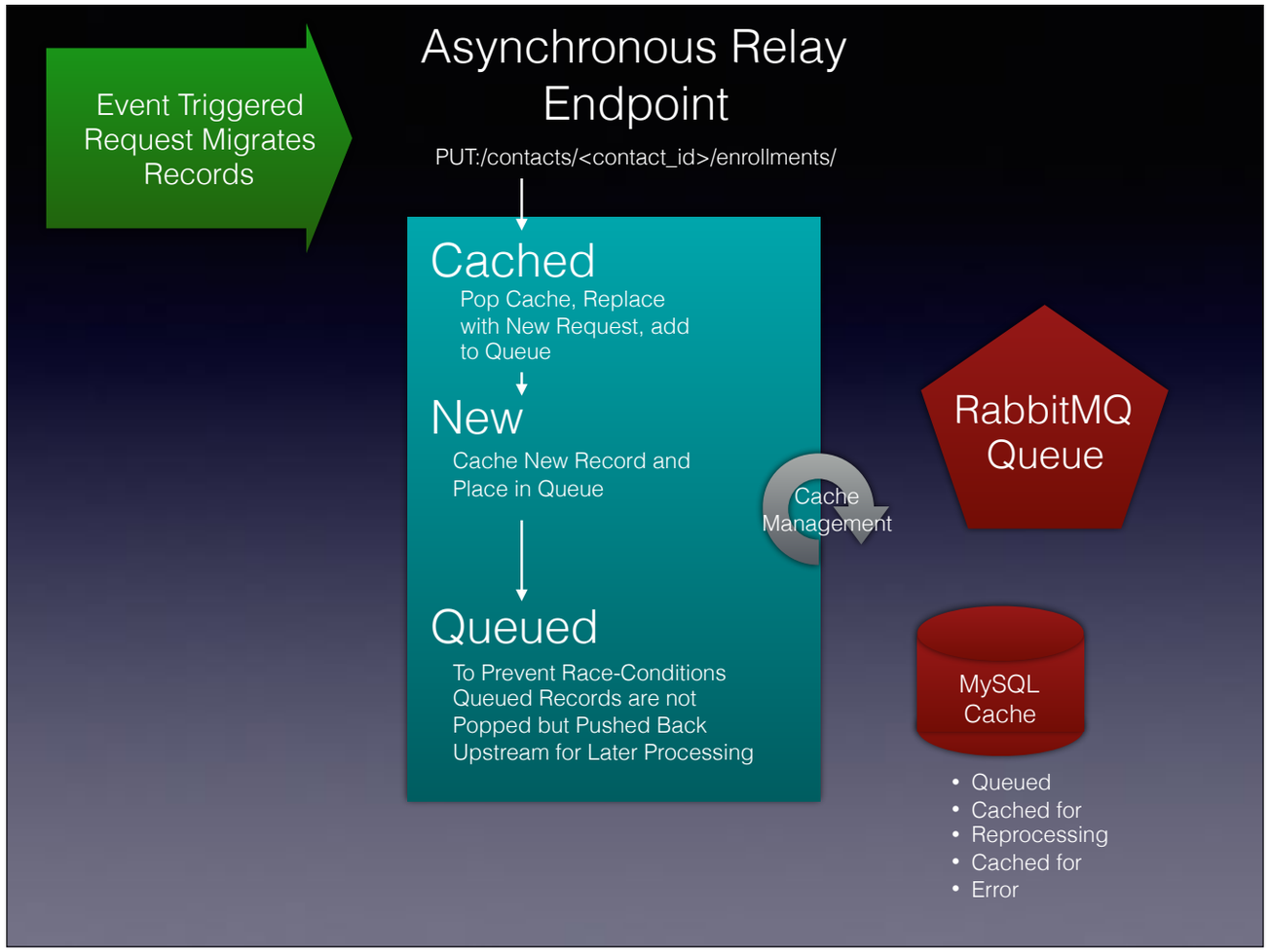
Between the back office and the private production instance of Open edX exists an Asynchronous Message Relay. The Asynchronous relay ensures a timely response to the back office push of data to Open edX, allowing these back office systems to continue without blocking for a response. The asynchronous relay then manages the final push of the data to Open edX, cycling the request through a queue and cash that leaves a record trail of the number of attempts needed to push the data, or the final error state if the push was unsuccessful.



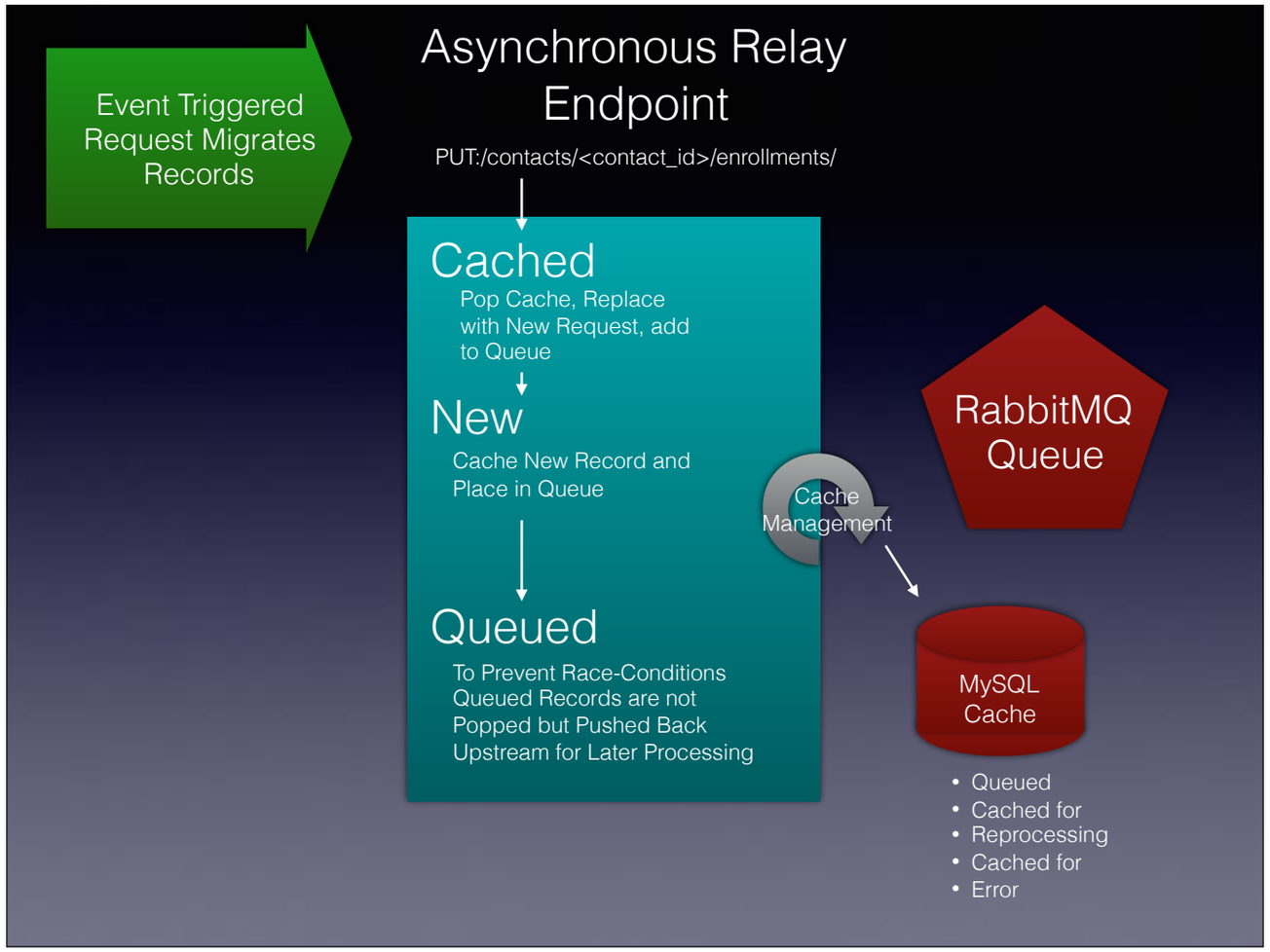
Taking a closer look at the asynchronous relay. The relay manages the complexity of incoming and outgoing requests based on the state of any previously cached request for a specific contact ID. The contact ID request is managed with a MySQL Cache that keeps a copy of each request. The possible states of a request in the cache are; queued for request to Open edX, cached for reprocessing because of a non-fatal error (409 response from edX), and cached for a fatal error (403 response from edX). This relay detail could be the subject of a talk on it's own, but is only here for context, and will likely spur questions at the end.



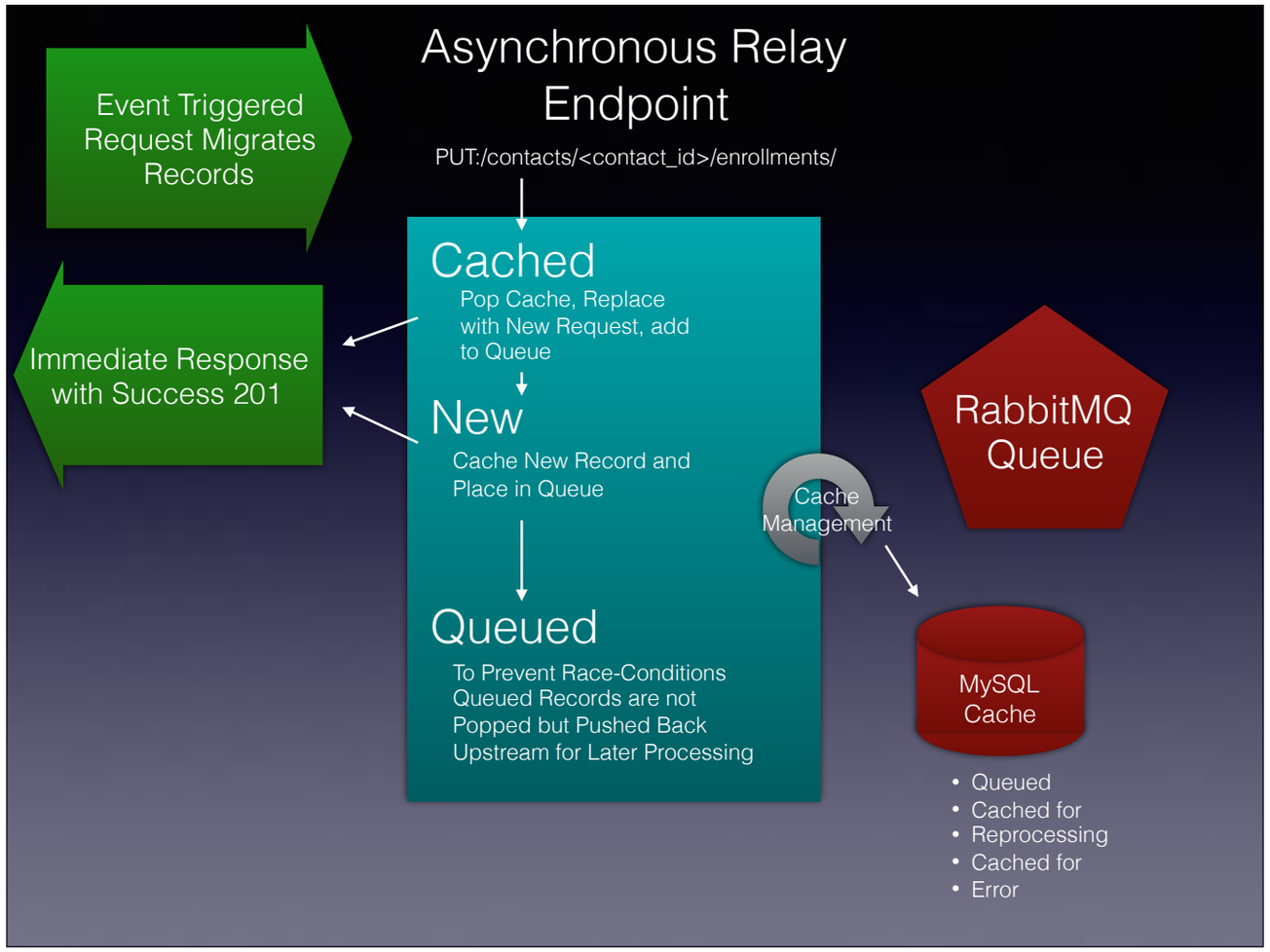
Taking a closer look at the asynchronous relay. The relay manages the complexity of incoming and outgoing requests based on the state of any previously cached request for a specific contact ID. The contact ID request is managed with a MySQL Cache that keeps a copy of each request. The possible states of a request in the cache are; queued for request to Open edX, cached for reprocessing because of a non-fatal error (409 response from edX), and cached for a fatal error (403 response from edX). This relay detail could be the subject of a talk on it's own, but is only here for context, and will likely spur questions at the end.



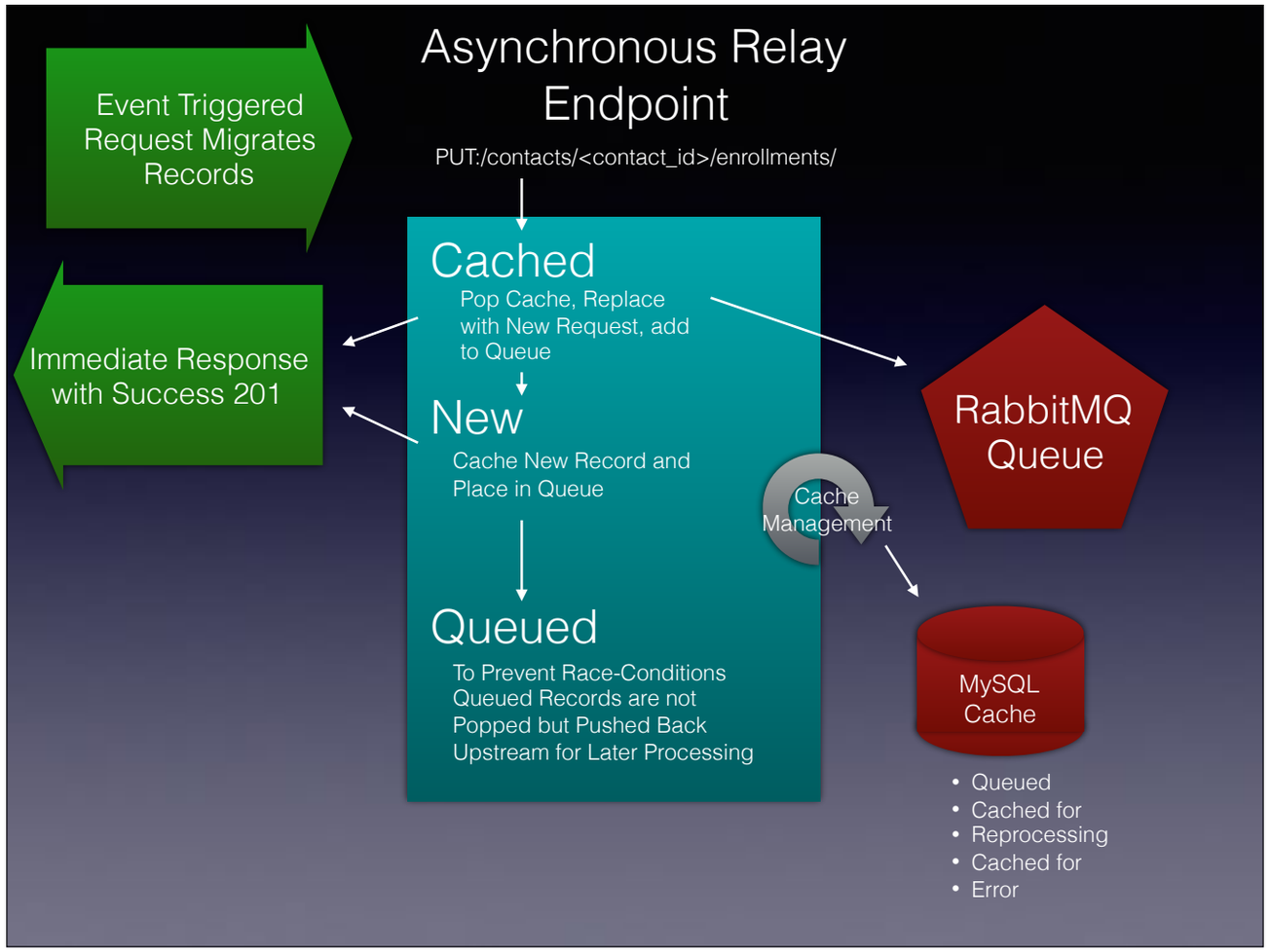
Taking a closer look at the asynchronous relay. The relay manages the complexity of incoming and outgoing requests based on the state of any previously cached request for a specific contact ID. The contact ID request is managed with a MySQL Cache that keeps a copy of each request. The possible states of a request in the cache are; queued for request to Open edX, cached for reprocessing because of a non-fatal error (409 response from edX), and cached for a fatal error (403 response from edX). This relay detail could be the subject of a talk on it's own, but is only here for context, and will likely spur questions at the end.



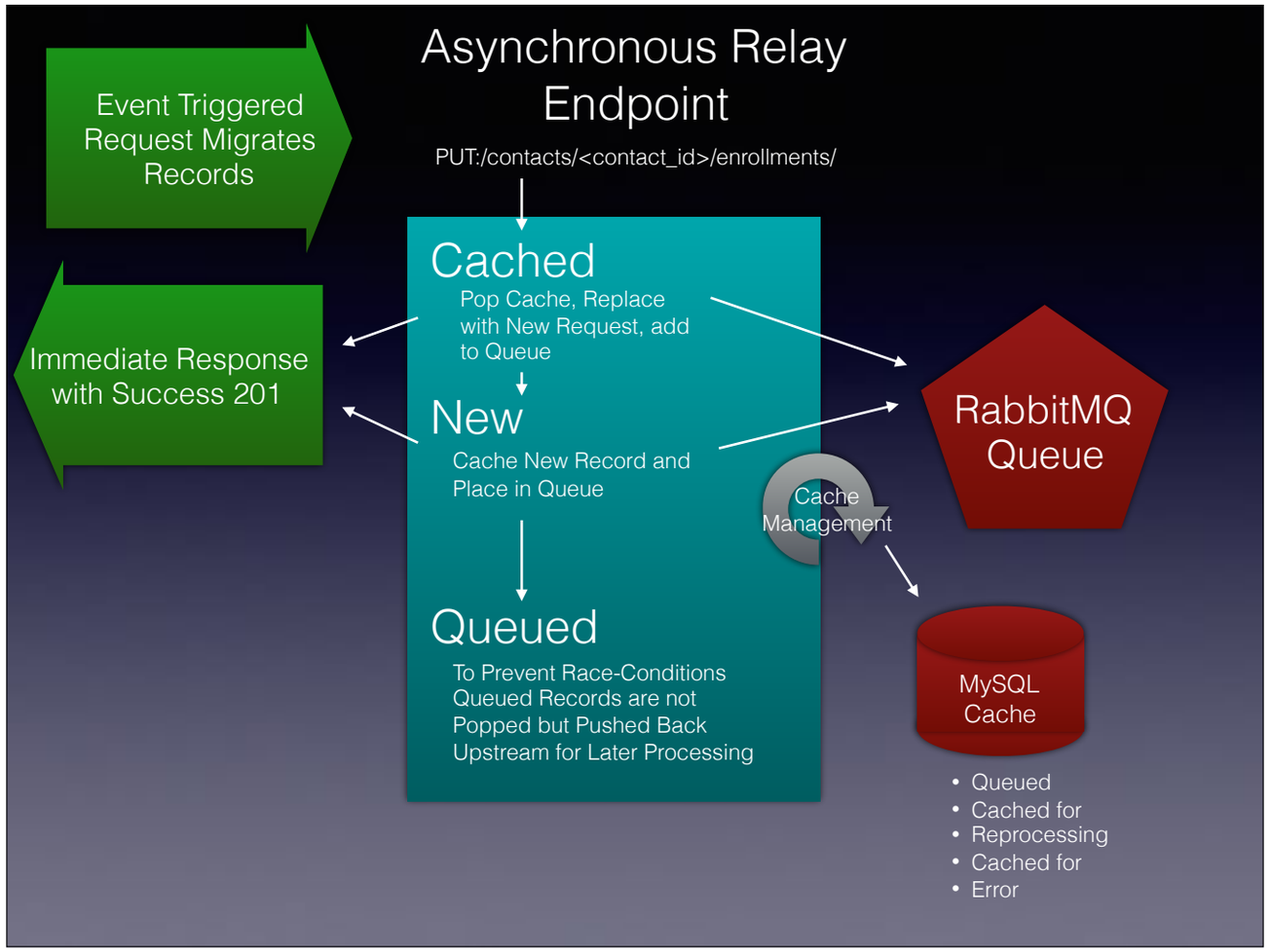
Taking a closer look at the asynchronous relay. The relay manages the complexity of incoming and outgoing requests based on the state of any previously cached request for a specific contact ID. The contact ID request is managed with a MySQL Cache that keeps a copy of each request. The possible states of a request in the cache are; queued for request to Open edX, cached for reprocessing because of a non-fatal error (409 response from edX), and cached for a fatal error (403 response from edX). This relay detail could be the subject of a talk on it's own, but is only here for context, and will likely spur questions at the end.



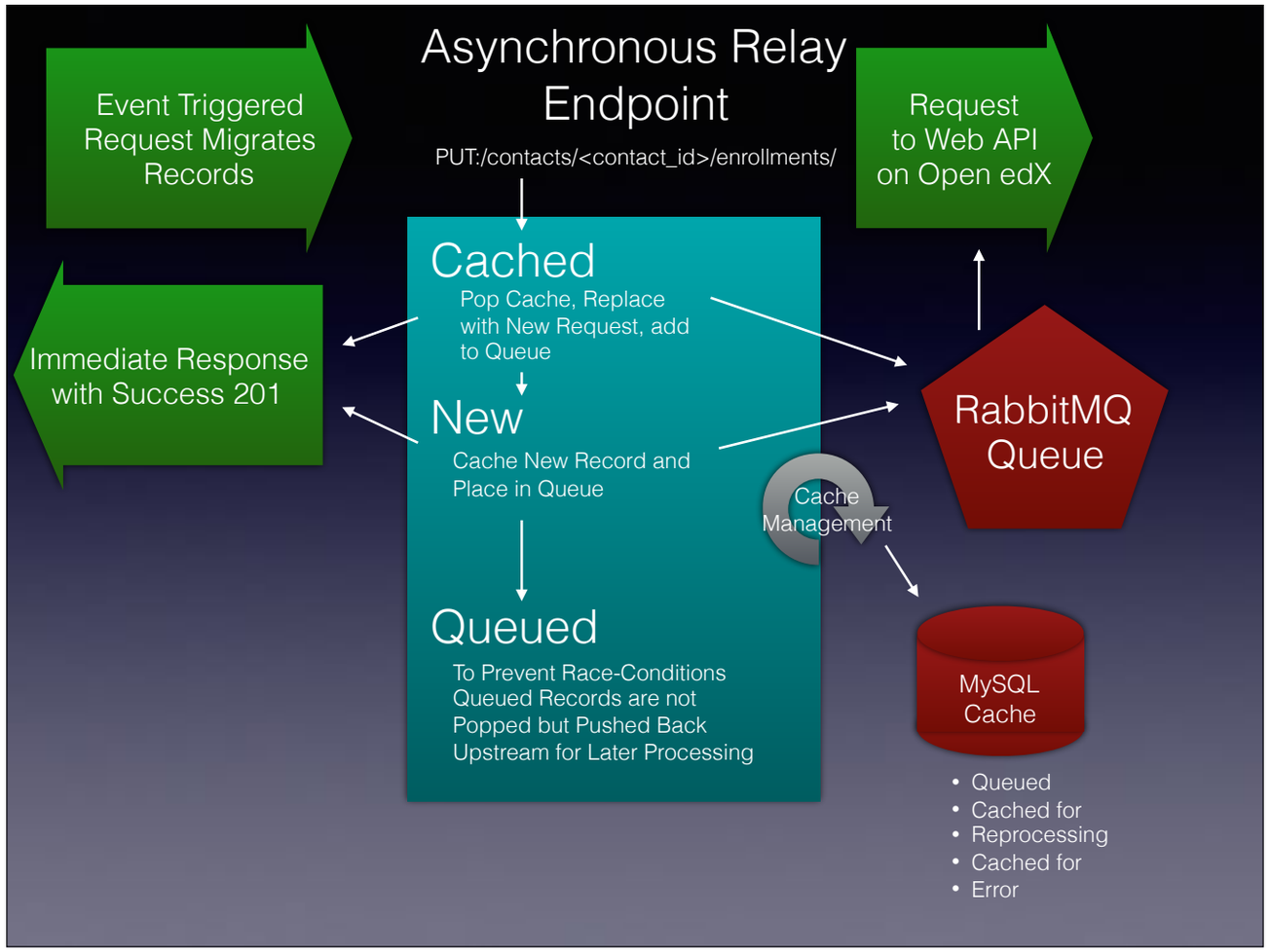
Taking a closer look at the asynchronous relay. The relay manages the complexity of incoming and outgoing requests based on the state of any previously cached request for a specific contact ID. The contact ID request is managed with a MySQL Cache that keeps a copy of each request. The possible states of a request in the cache are; queued for request to Open edX, cached for reprocessing because of a non-fatal error (409 response from edX), and cached for a fatal error (403 response from edX). This relay detail could be the subject of a talk on it's own, but is only here for context, and will likely spur questions at the end.



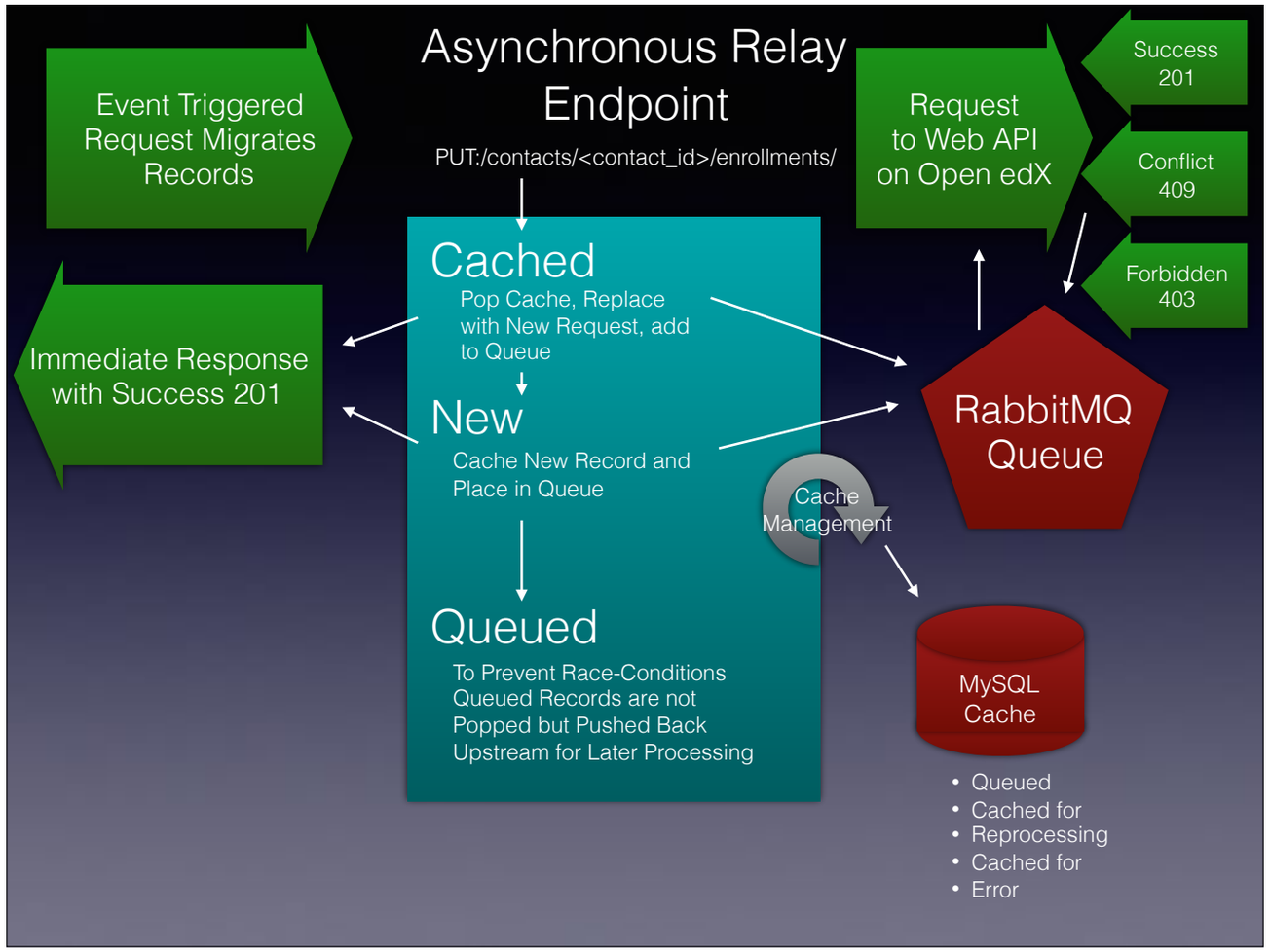
Taking a closer look at the asynchronous relay. The relay manages the complexity of incoming and outgoing requests based on the state of any previously cached request for a specific contact ID. The contact ID request is managed with a MySQL Cache that keeps a copy of each request. The possible states of a request in the cache are; queued for request to Open edX, cached for reprocessing because of a non-fatal error (409 response from edX), and cached for a fatal error (403 response from edX). This relay detail could be the subject of a talk on it's own, but is only here for context, and will likely spur questions at the end.



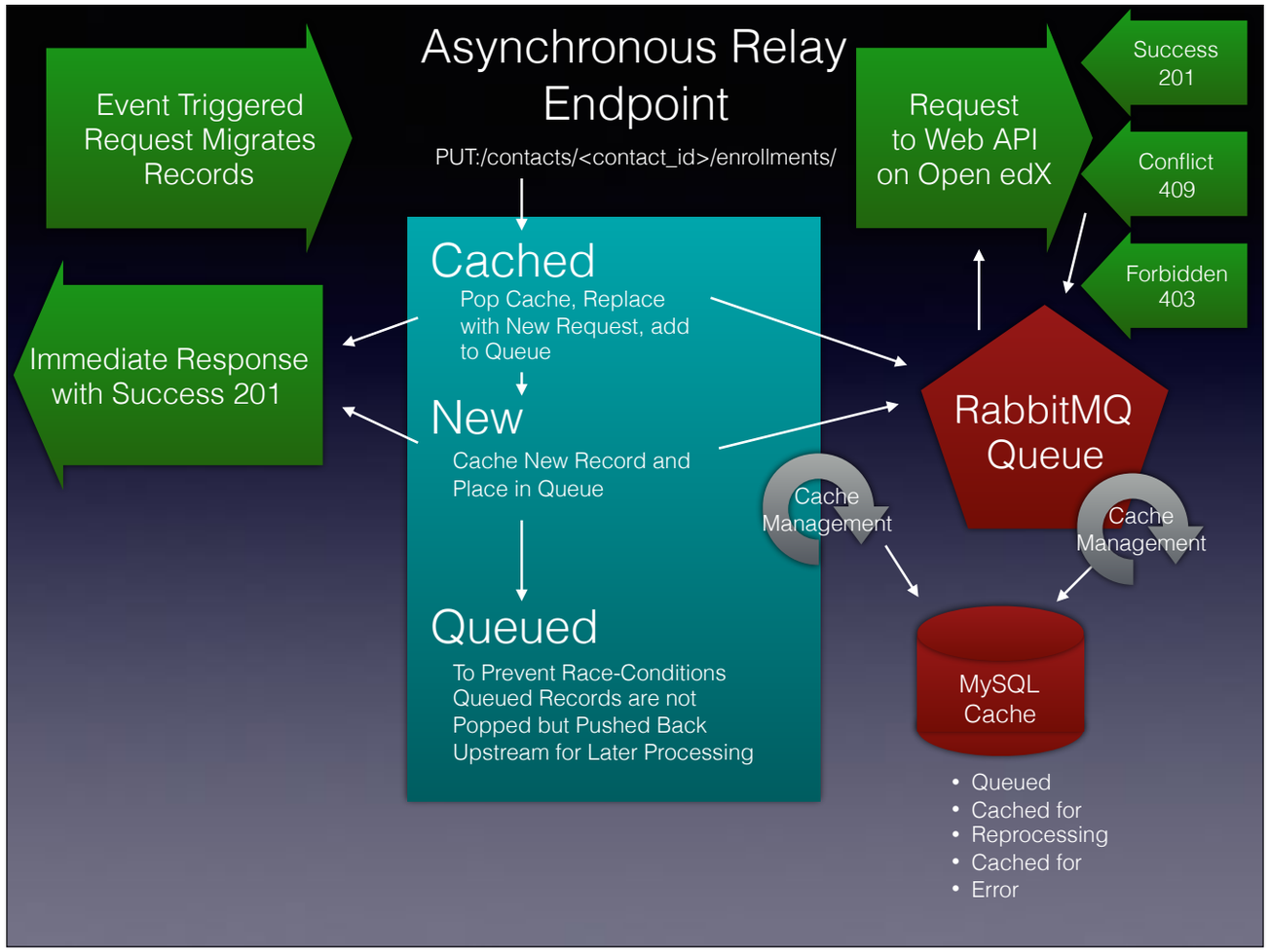
Taking a closer look at the asynchronous relay. The relay manages the complexity of incoming and outgoing requests based on the state of any previously cached request for a specific contact ID. The contact ID request is managed with a MySQL Cache that keeps a copy of each request. The possible states of a request in the cache are; queued for request to Open edX, cached for reprocessing because of a non-fatal error (409 response from edX), and cached for a fatal error (403 response from edX). This relay detail could be the subject of a talk on it's own, but is only here for context, and will likely spur questions at the end.



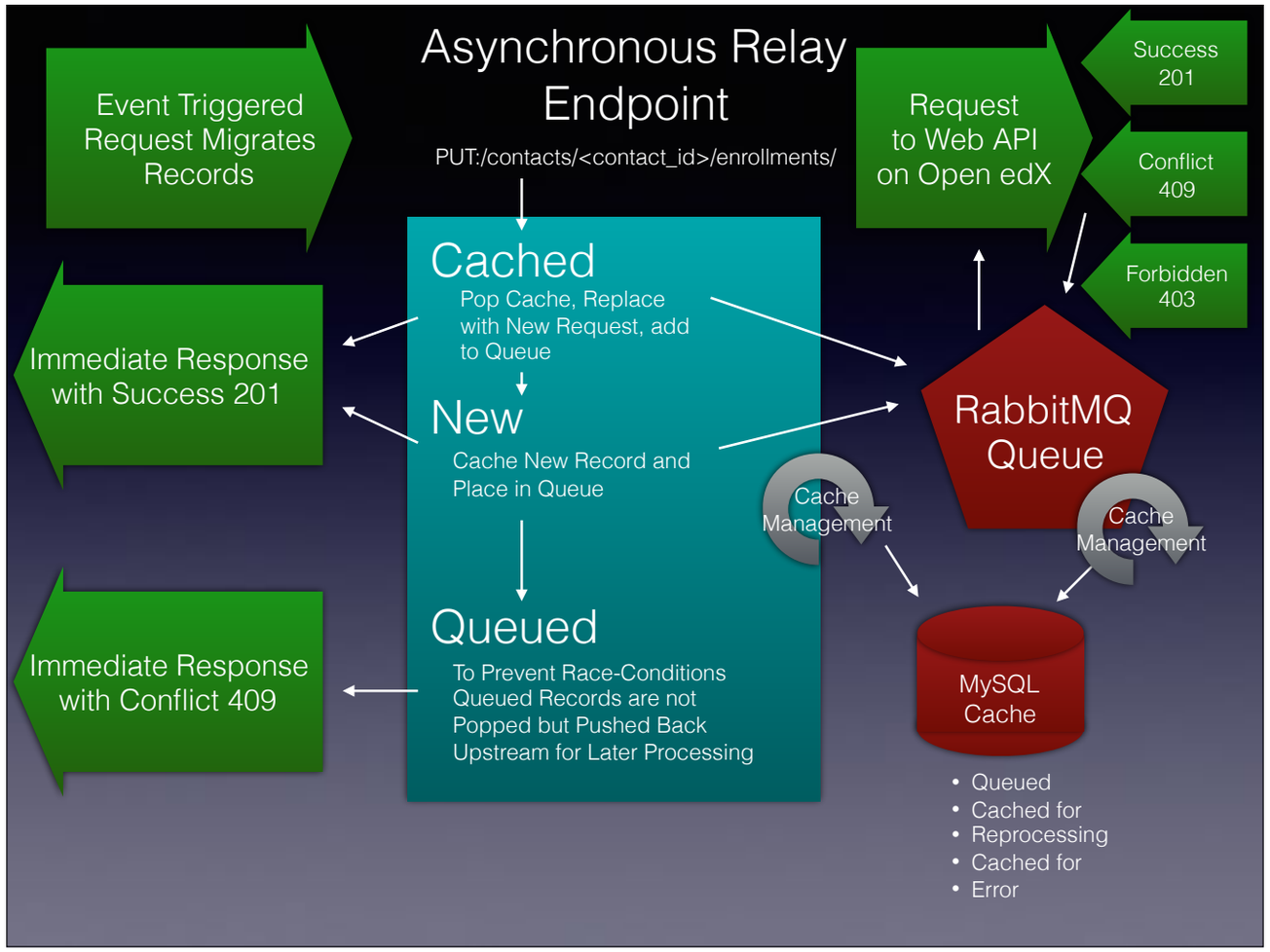
Taking a closer look at the asynchronous relay. The relay manages the complexity of incoming and outgoing requests based on the state of any previously cached request for a specific contact ID. The contact ID request is managed with a MySQL Cache that keeps a copy of each request. The possible states of a request in the cache are; queued for request to Open edX, cached for reprocessing because of a non-fatal error (409 response from edX), and cached for a fatal error (403 response from edX). This relay detail could be the subject of a talk on it's own, but is only here for context, and will likely spur questions at the end.



Taking a closer look at the asynchronous relay. The relay manages the complexity of incoming and outgoing requests based on the state of any previously cached request for a specific contact ID. The contact ID request is managed with a MySQL Cache that keeps a copy of each request. The possible states of a request in the cache are; queued for request to Open edX, cached for reprocessing because of a non-fatal error (409 response from edX), and cached for a fatal error (403 response from edX). This relay detail could be the subject of a talk on it's own, but is only here for context, and will likely spur questions at the end.



Taking a closer look at the asynchronous relay. The relay manages the complexity of incoming and outgoing requests based on the state of any previously cached request for a specific contact ID. The contact ID request is managed with a MySQL Cache that keeps a copy of each request. The possible states of a request in the cache are; queued for request to Open edX, cached for reprocessing because of a non-fatal error (409 response from edX), and cached for a fatal error (403 response from edX). This relay detail could be the subject of a talk on it's own, but is only here for context, and will likely spur questions at the end.



Taking a closer look at the asynchronous relay. The relay manages the complexity of incoming and outgoing requests based on the state of any previously cached request for a specific contact ID. The contact ID request is managed with a MySQL Cache that keeps a copy of each request. The possible states of a request in the cache are; queued for request to Open edX, cached for reprocessing because of a non-fatal error (409 response from edX), and cached for a fatal error (403 response from edX). This relay detail could be the subject of a talk on it's own, but is only here for context, and will likely spur questions at the end.

Environment

Global Knowledges environment is heterogeneous, multiple operating systems, database management systems, programming languages, and environments.

Environment

Linux

Windows

Global Knowledges environment is heterogeneous, multiple operating systems, database management systems, programming languages, and environments.

Environment

Linux

MySQL

SQLServer

Windows

Global Knowledges environment is heterogeneous, multiple operating systems, database management systems, programming languages, and environments.

Environment

.NET

MySQL

Linux

SQLServer

Windows

Python

Global Knowledges environment is heterogeneous, multiple operating systems, database management systems, programming languages, and environments.

Environment

.NET

MySQL

Linux

SQLServer

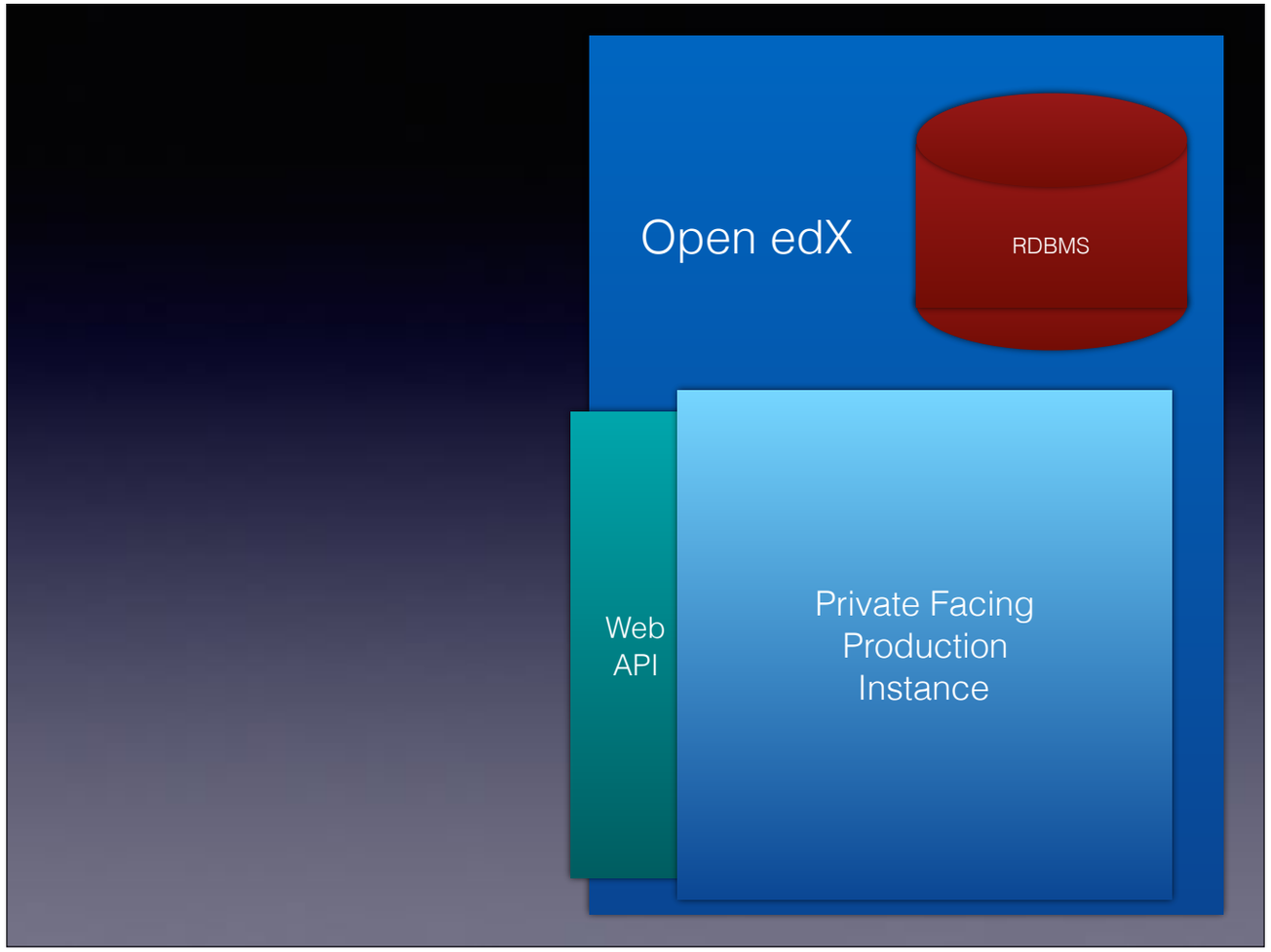
on premise

Windows

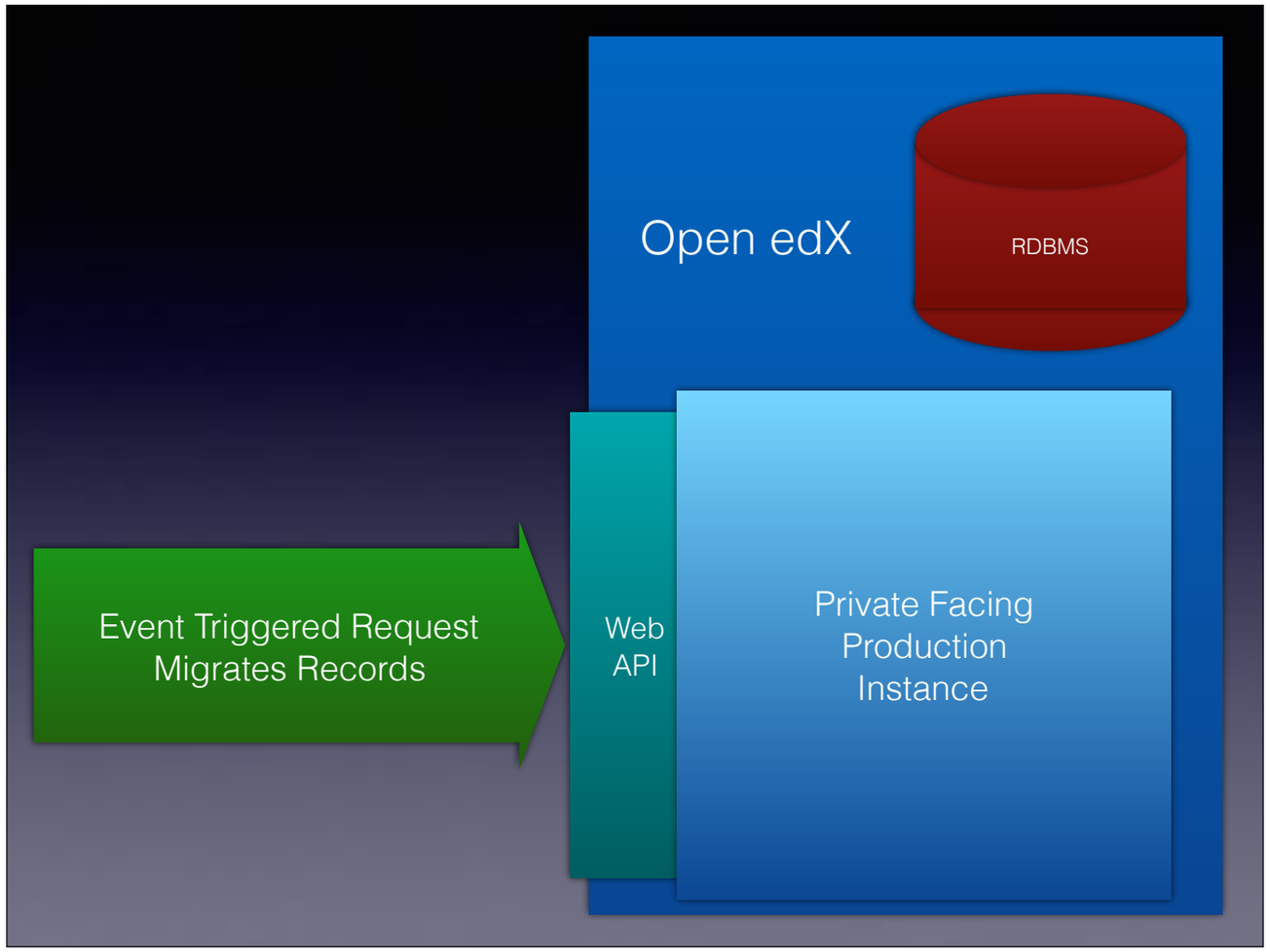
Python

cloud

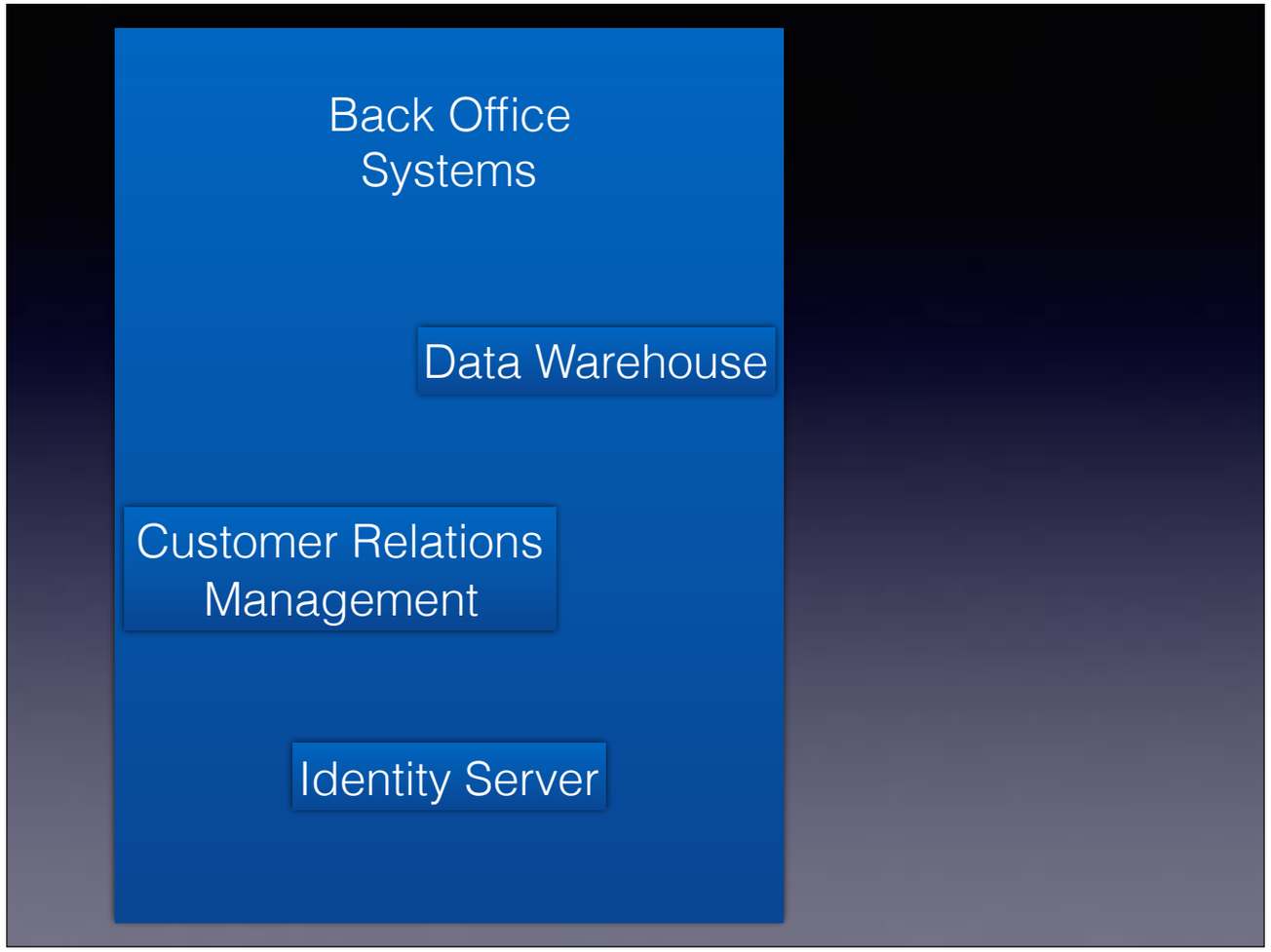
Global Knowledges environment is heterogeneous, multiple operating systems, database management systems, programming languages, and environments.



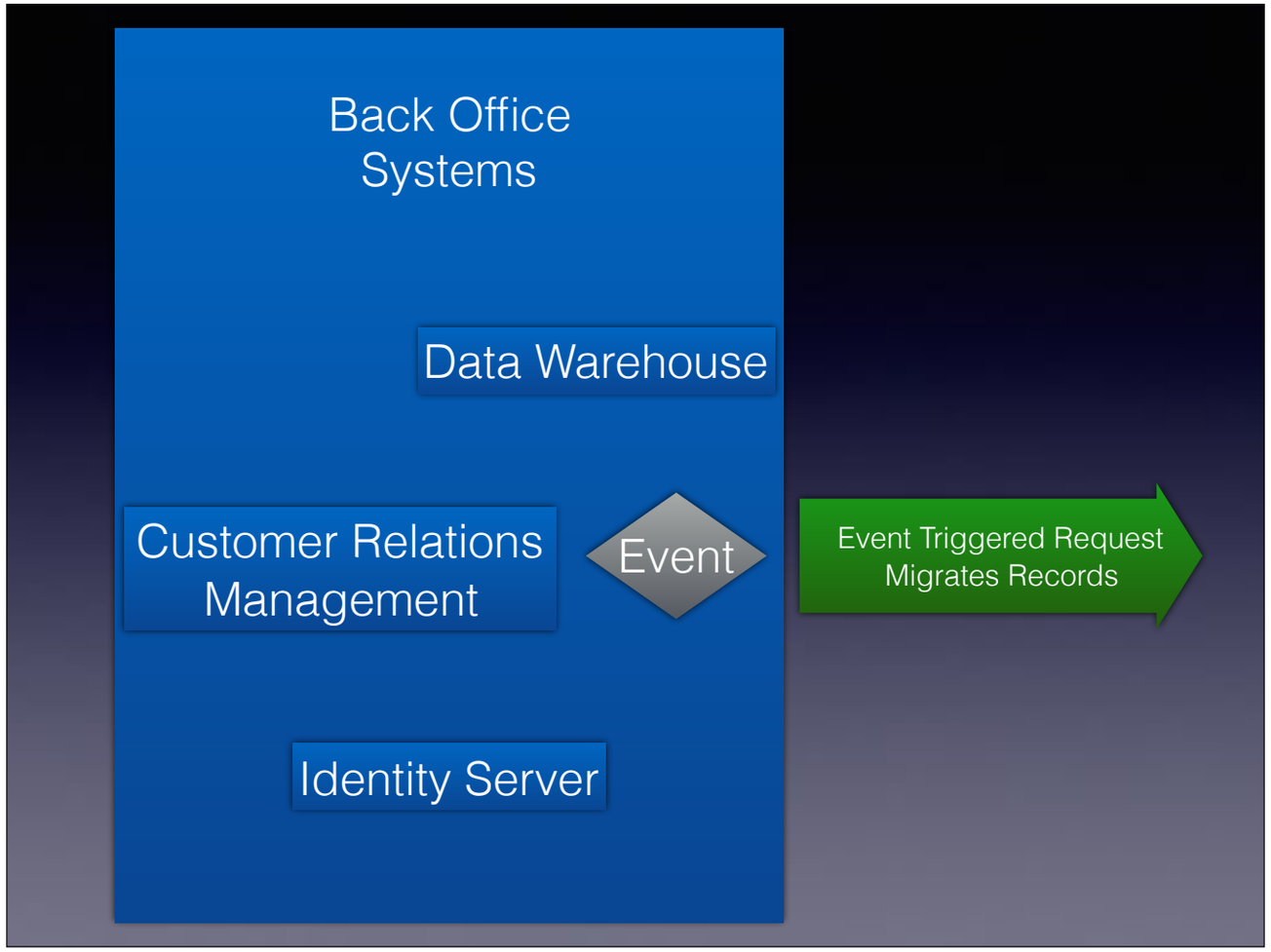
The arriving request enters the private facing production instance of Open edX through a custom Web API that places incoming records into a set of landing tables before creating the Open edX student accounts and enrollments.



The arriving request enters the private facing production instance of Open edX through a custom Web API that places incoming records into a set of landing tables before creating the Open edX student accounts and enrollments.



Back office systems are upstream of Open edX, meaning user data flows from the back office to Open edX, entry of new or the edit of an existing user enrollment in the back office, triggers an [event] that pushes a user enrollment record to Open edX



Back office systems are upstream of Open edX, meaning user data flows from the back office to Open edX, entry of new or the edit of an existing user enrollment in the back office, triggers an [event] that pushes a user enrollment record to Open edX