Services

#WWDC14

Optimizing In-App Purchases

Using StoreKit

Session 303
James Wilson
Software Engineering

StoreKit

StoreKit Features

StoreKit Features

In-App Purchases

- Consumables and non-consumable
- Subscriptions

StoreKit Features

In-App Purchases

- Consumables and non-consumable
- Subscriptions

Store Product Sheet

StoreKit Features

In-App Purchases

- Consumables and non-consumable
- Subscriptions

Store Product Sheet

Receipt renewal

What’s New

What's New

StoreKit product sheet supports affiliate program

What's New

StoreKit product sheet supports affiliate program

New transaction state—Deferred

- Ask to Buy feature of Family Sharing

Deferred Transaction State

SKPaymentTransactionStateDeferred

Deferred Transaction State

SKPaymentTransactionStateDeferred

The payment is neither purchased nor failed, yet

Deferred Transaction State

SKPaymentTransactionStateDeferred

The payment is neither purchased nor failed, yet

- Further update will be received

Deferred Transaction State

SKPaymentTransactionStateDeferred

The payment is neither purchased nor failed, yet

- Further update will be received
- Indeterminate time

Deferred Transaction State

SKPaymentTransactionStateDeferred

The payment is neither purchased nor failed, yet

- Further update will be received
- Indeterminate time

Must allow the user to continue to use the app

Deferred Transaction State

SKPaymentTransactionStateDeferred

NEW

The payment is neither purchased nor failed, yet

- Further update will be received
- Indeterminate time

Must allow the user to continue to use the app

- Repurchasing the item is allowed

Deferred Transaction State

SKPaymentTransactionStateDeferred

The payment is neither purchased nor failed, yet

- Further update will be received
- Indeterminate time

Must allow the user to continue to use the app

- Repurchasing the item is allowed
- Let StoreKit handle the interaction

Deferred Transaction State

Ask to Buy

NEW

Child

Parent

Deferred Transaction State

Ask to Buy

NEW

Child

Attempts In-App Purchase

Parent

Deferred Transaction State

Ask to Buy

NEW

Child

Attempts In-App Purchase

Parent

Notified of request

Deferred Transaction State

Ask to Buy

NEW

`SKPaymentTransactionStateDeferred`

Child

Attempts In-App Purchase

Parent

Notified of request

Deferred Transaction State

Ask to Buy

NEW

Child

Parent

Deferred Transaction State

Ask to Buy

NEW

Child

Parent

Approves or declines

Deferred Transaction State

Ask to Buy

NEW

Child

Transaction updated

Parent

Approves or declines

Deferred Transaction State

Ask to Buy

NEW

`SKPaymentTransactionStatePurchased`

`SKPaymentTransactionStateFailed`

Child

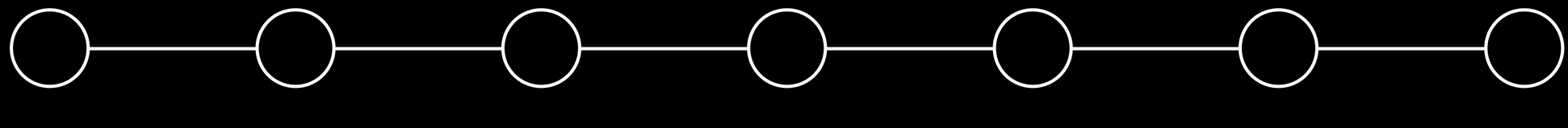
Transaction updated

Parent

Approves or declines

Optimizing In-App Purchases

In-App Purchase Process



Load In-App Identifiers

Fetch Product Info

Show In-App UI

Make Purchase

Process Transaction

Make Asset Available

Finish Transaction

In-App Purchase Process

Load In-App Identifiers

Fetch Product Info

Show In-App UI

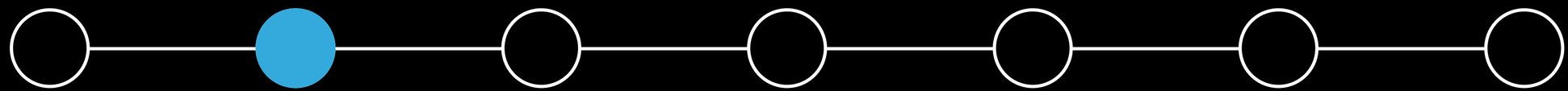
Make Purchase

Process Transaction

Make Asset Available

Finish Transaction

In-App Purchase Process



Load In-App Identifiers

Fetch Product Info

Show In-App UI

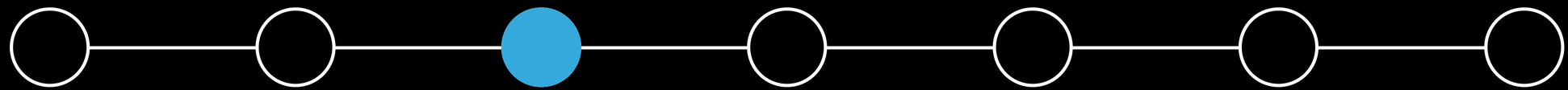
Make Purchase

Process Transaction

Make Asset Available

Finish Transaction

In-App Purchase Process



Load In-App Identifiers

Fetch Product Info

Show In-App UI

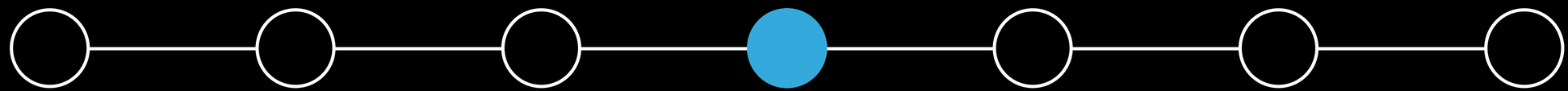
Make Purchase

Process Transaction

Make Asset Available

Finish Transaction

In-App Purchase Process



Load In-App Identifiers

Fetch Product Info

Show In-App UI

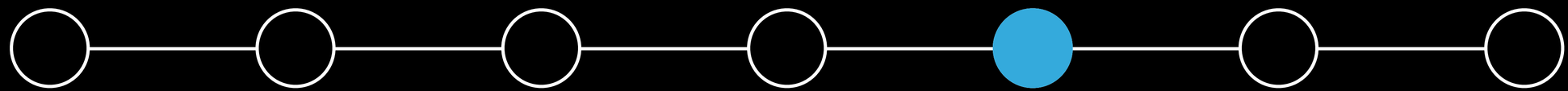
Make Purchase

Process Transaction

Make Asset Available

Finish Transaction

In-App Purchase Process



Load In-App Identifiers

Fetch Product Info

Show In-App UI

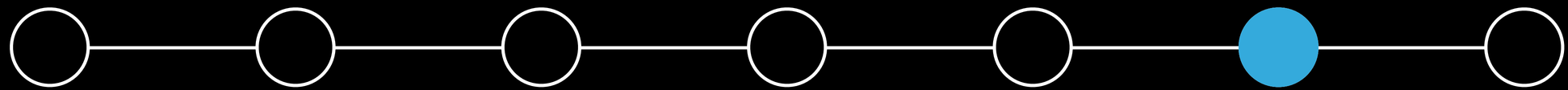
Make Purchase

Process Transaction

Make Asset Available

Finish Transaction

In-App Purchase Process



Load In-App Identifiers

Fetch Product Info

Show In-App UI

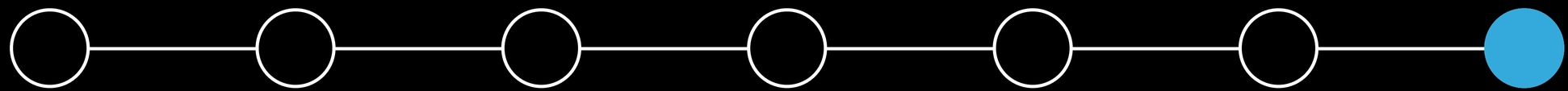
Make Purchase

Process Transaction

Make Asset Available

Finish Transaction

In-App Purchase Process



Load In-App Identifiers

Fetch Product Info

Show In-App UI

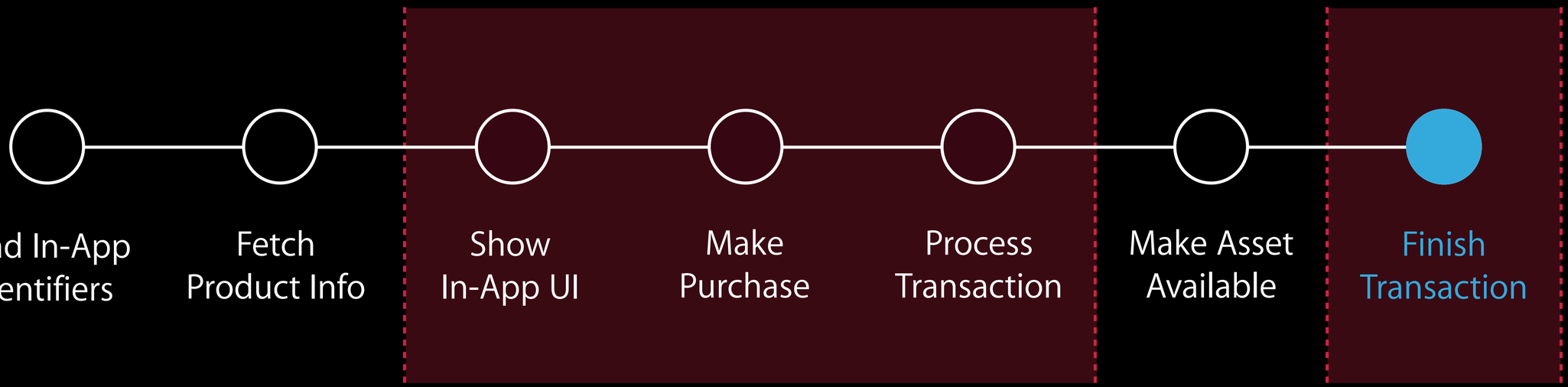
Make Purchase

Process Transaction

Make Asset Available

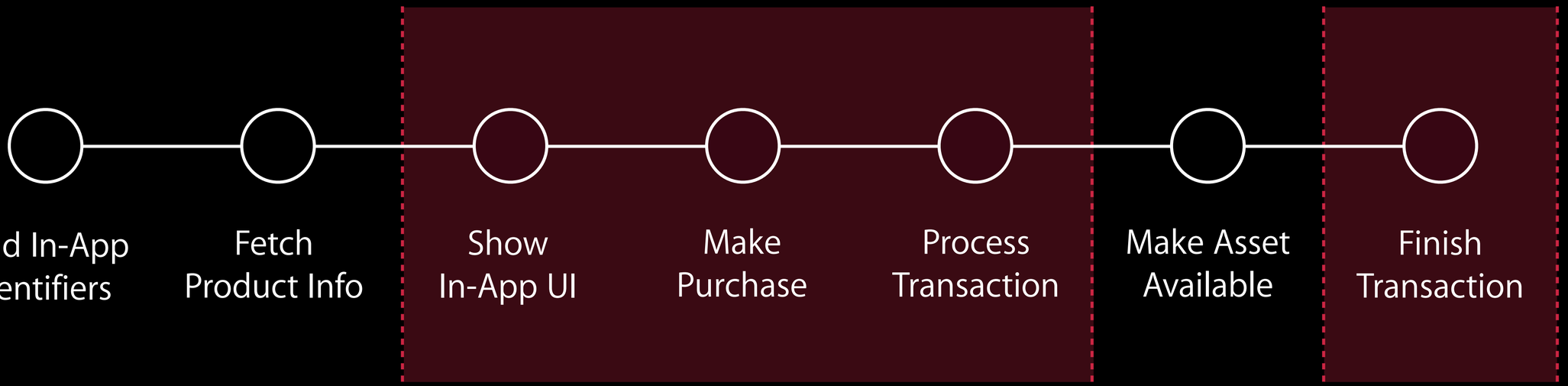
Finish Transaction

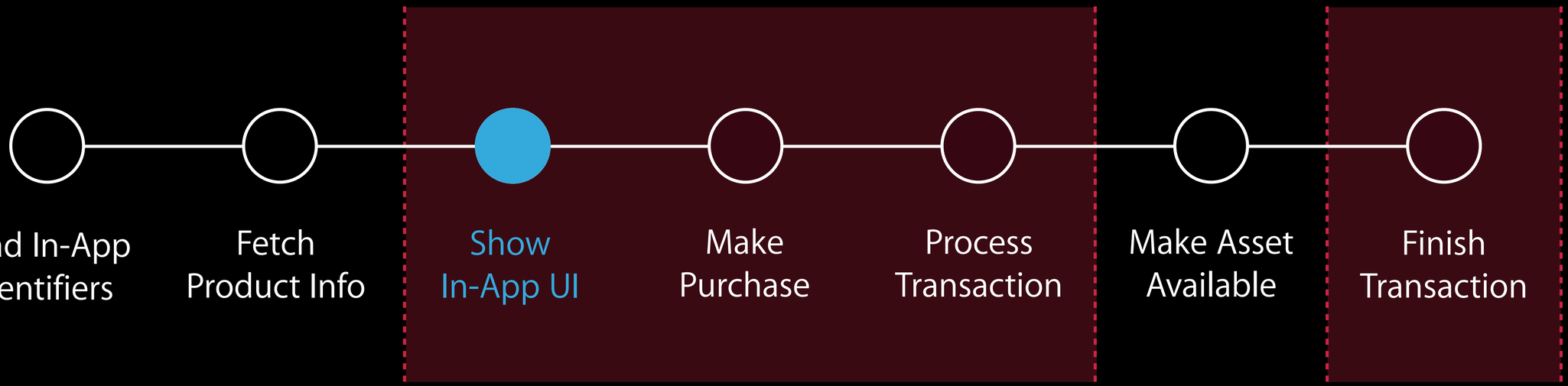
In-App Purchase Process



Danger Zones

User Interaction


Load In-App Identifiers
Fetch Product Info
Show In-App UI
Make Purchase
Process Transaction
Make Asset Available
Finish Transaction


Load In-App Identifiers
Fetch Product Info
Show In-App UI
Make Purchase
Process Transaction
Make Asset Available
Finish Transaction

Product Identifier

Product Identifier

Options for storing the list of product identifiers

Product Identifier

Options for storing the list of product identifiers

- Baked-in product identifier

Product Identifier

Options for storing the list of product identifiers

- Baked-in product identifier
- Fetch from server

Product Identifier

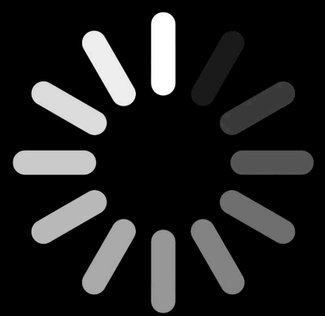
Options for storing the list of product identifiers

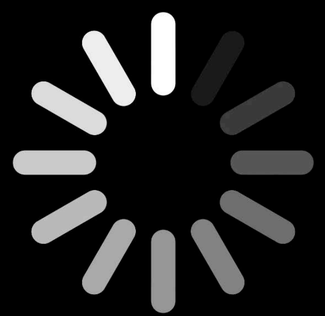
- Baked-in product identifier
- Fetch from server
  - Cache strategy

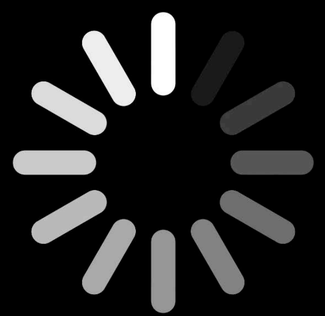
Product Identifier

Options for storing the list of product identifiers

- Baked-in product identifier
- Fetch from server
  - Cache strategy
  - Reliability







Not the way to start an In-App Purchase

Product Information

Fetch the product information using StoreKit

```
SKProductsRequest* request = [[SKProductsRequest alloc]
                                    initWithProductIdentifiers:identifierSet];
```

Product Information

Fetch the product information using StoreKit

```
SKProductsRequest* request = [[SKProductsRequest alloc]
                                    initWithProductIdentifiers:identifierSet];
```

Anticipate the presentation

Product Information

Fetch the product information using StoreKit

```
SKProductsRequest* request = [[SKProductsRequest alloc]
                                    initWithProductIdentifiers:identifierSet];
```

Anticipate the presentation

- Fetch product info just-ahead-of-time

SKProduct Object

SKProduct properties

- Localized title and description
- Price and locale
- Content size and version (hosted)

Price and Currency

Price and Currency

1.234,56 €

Price and Currency

1.234,56 €

£1,234.56

Price and Currency

1.234,56 €

£1,234.56

€1,234.56

Price and Currency

1.234,56 €

£1,234.56

€1,234.56

1.234,56 kn

Price and Currency

1.234,56 €

£1,234.56

€1,234.56

1.234,56 kn

R$ 1.234,56

Price and Currency

1.234,56 €

£1,234.56

€1,234.56

1.234,56 kn

R$ 1.234,56

฿1,234.56

Price and Currency

1.234,56 €

£1,234.56

€1,234.56

1.234,56 kn

R$ 1.234,56

฿1,234.56

$1,234.56

Price and Currency

Showing localized price

Price and Currency

Showing localized price

```
NSNumberFormatter *numberFormatter = [[NSNumberFormatter alloc] init];
```

Price and Currency

Showing localized price

```
NSNumberFormatter *numberFormatter = [[NSNumberFormatter alloc] init];
[numberFormatter setNumberStyle:NSNumberFormatterCurrencyStyle];
```

Price and Currency

Showing localized price

```
NSNumberFormatter *numberFormatter = [[NSNumberFormatter alloc] init];
[numberFormatter setNumberStyle:NSNumberFormatterCurrencyStyle];
[numberFormatter setLocale:product.priceLocale];
```

Price and Currency

Showing localized price

```
NSNumberFormatter *numberFormatter = [[NSNumberFormatter alloc] init];
[numberFormatter setNumberStyle:NSNumberFormatterCurrencyStyle];
[numberFormatter setLocale:product.priceLocale];
NSString *formattedString = [numberFormatter stringFromNumber:product.price];
```

Price and Currency

Showing localized price

```
NSNumberFormatter *numberFormatter = [[NSNumberFormatter alloc] init];
[numberFormatter setNumberStyle:NSNumberFormatterCurrencyStyle];
[numberFormatter setLocale:product.priceLocale];
NSString *formattedString = [numberFormatter stringFromNumber:product.price];
```

Do not perform currency conversion!

Handling Errors

Handling Errors

Not all errors are equal

Handling Errors

Not all errors are equal

Check the error code

Handling Errors

Not all errors are equal

Check the error code

- Don’t show an error alert unless necessary

Handling Errors

Not all errors are equal

Check the error code

- Don’t show an error alert unless necessary
- User canceling a payment will result in an error

Handling Errors

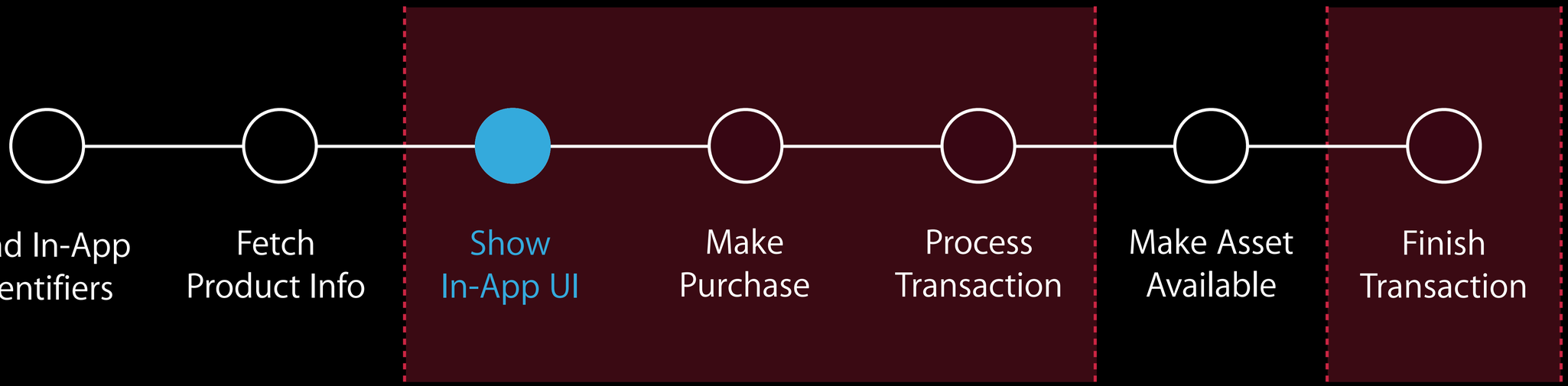
Not all errors are equal

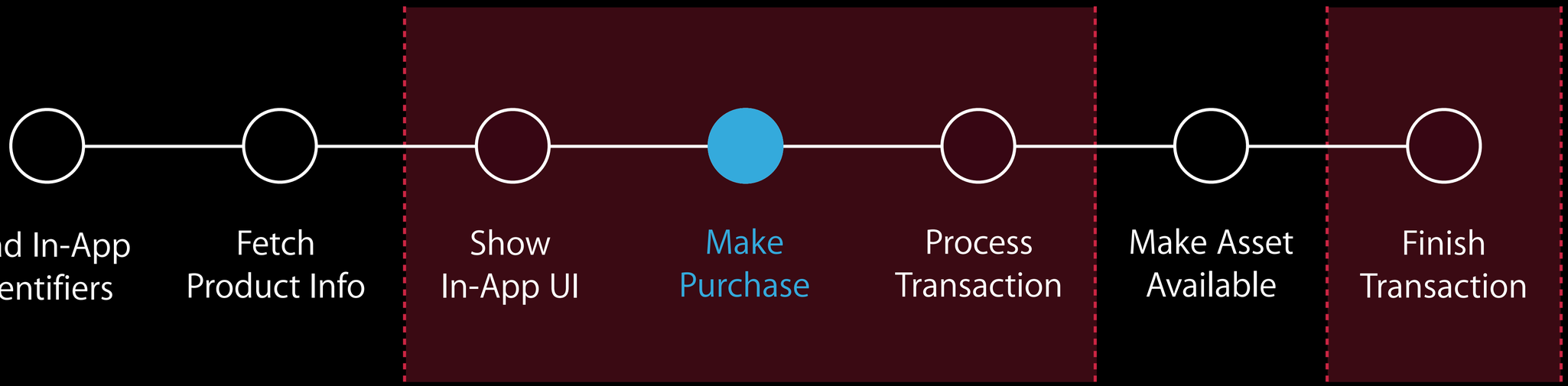
Check the error code

- Don’t show an error alert unless necessary
- User canceling a payment will result in an error

Let StoreKit handle the transaction flow as much as possible

Making the Purchase


Load In-App Identifiers
Fetch Product Info
Show In-App UI
Make Purchase
Process Transaction
Make Asset Available
Finish Transaction


Load In-App Identifiers
Fetch Product Info
Show In-App UI
Make Purchase
Process Transaction
Make Asset Available
Finish Transaction

The Payment Queue

Observe it, always

The Payment Queue

Observe it, always

The center of your In-App Purchase implementation

- The only source of truth for state

The Payment Queue

Observe it, always

The center of your In-App Purchase implementation

- The only source of truth for state

Rely on the queue, and only the queue

- For transactions in progress
- Payment status updates
- Download status

The Payment Queue

Observe it, always

The center of your In-App Purchase implementation

- The only source of truth for state

Rely on the queue, and only the queue

- For transactions in progress
- Payment status updates
- Download status

Any and all transactions in the queue are valid and real

On Launch

Start observing the payment queue

On Launch

Start observing the payment queue

```
[[SKPaymentQueue defaultQueue] addTransactionObserver:yourObserver];
```

Example

Fetch product info

Get information about your products from the store

Example

Fetch product info

Get information about your products from the store

```
NSSet* identifierSet = [NSSet setWithArray:productIdentifiers];
```

Example

Fetch product info

Get information about your products from the store

```
NSSet* identifierSet = [NSSet setWithArray:productIdentifiers];

SKProductsRequest* request = [[SKProductsRequest alloc]
                  initWithProductIdentifiers: identifierSet];
```

Example

Fetch product info

Get information about your products from the store

```
NSSet* identifierSet = [NSSet setWithArray:productIdentifiers];

SKProductsRequest* request = [[SKProductsRequest alloc]
                initWithProductIdentifiers: identifierSet];

request.delegate = self;
```

Example

Fetch product info

Get information about your products from the store

```
NSSet* identifierSet = [NSSet setWithArray:productIdentifiers];

SKProductsRequest* request = [[SKProductsRequest alloc]
                initWithProductIdentifiers: identifierSet];

request.delegate = self;
[request start];
```

Example

Add payment to queue

Start the payment transaction

Example

Add payment to queue

Start the payment transaction

```
SKPayment* payment = [SKPayment paymentWithProduct:product];
```

Example

Add payment to queue

Start the payment transaction

```
SKPayment* payment = [SKPayment paymentWithProduct:product];
[[SKPaymentQueue defaultQueue] addPayment:payment];
```

Example

Handle events

Example

Handle events

```
- (void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions
```

Example

Handle events

```
- (void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions
{
   for (SKPaymentTransaction* transaction in transactions)
```

Example

Handle events

```
- (void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions
{
    for (SKPaymentTransaction* transaction in transactions)
    {
        switch(transaction.transactionState) {
```

Example

Handle events

```
- (void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions
{
    for (SKPaymentTransaction* transaction in transactions)
    {
        switch(transaction.transactionState) {
            case SKPaymentTransactionStatePurchased:
```

Example

Handle events

```
- (void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions
{
    for (SKPaymentTransaction* transaction in transactions)
    {
        switch(transaction.transactionState) {
            case SKPaymentTransactionStatePurchased:
                NSURL* receiptURL = [[NSBundle mainBundle] appStoreReceiptURL];
                NSData* receipt = [NSData dataWithContentsOfURL:receiptURL];
                // Process the transaction
        }
    }
```

Don’t Do This

```
case SKPaymentTransactionStatePurchased:
    // Get the local state for this transaction
```

Don’t Do This

```
case SKPaymentTransactionStatePurchased:
    // Get the local state for this transaction
    SKPayment *payment = myCachedPayments[transaction.payment.productIdentifier];
```

Don’t Do This

```
case SKPaymentTransactionStatePurchased:
    // Get the local state for this transaction
    SKPayment *payment = myCachedPayments[transaction.payment.productIdentifier];
    if (!payment)
    {
```

Don’t Do This

```
case SKPaymentTransactionStatePurchased:
    // Get the local state for this transaction
    SKPayment *payment = myCachedPayments[transaction.payment.productIdentifier];
    if (!payment)
    {
        // No idea where this transaction came from!
        // Ignore it
        continue;
    }
```

Don’t Do This

```
case SKPaymentTransactionStatePurchased:
    // Get the local state for this transaction
    SKPayment *payment = myCachedPayments[transaction.payment.productIdentifier];
    if (!payment)
    {
        // No idea where this transaction came from!
        // Ignore it
        continue;
    }
```

Tracking your own state or payment cache is unnecessary

Why Not?

Why Not?

Because, what if…

- You crash
- Purchase is disrupted
- Or your app didn’t even start the purchase

Why Not?

Because, what if…

- You crash
- Purchase is disrupted
- Or your app didn’t even start the purchase

The transaction is just as valid

- Process it always

Example

Handling deferred transaction

```
- (void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions
{
    for (SKPaymentTransaction* transaction in transactions)
    {
        switch(transaction.transactionState) {









        }
    }
}
```

Example

Handling deferred transaction

```
- (void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions
{
    for (SKPaymentTransaction* transaction in transactions)
    {
        switch(transaction.transactionState) {
            case SKPaymentTransactionStateDeferred:




        }
    }
}
```

Example

Handling deferred transaction

```
- (void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions
{
    for (SKPaymentTransaction* transaction in transactions)
    {
        switch(transaction.transactionState) {
            case SKPaymentTransactionStateDeferred:




        }
    }
}
```

Example

Handling deferred transaction

```
- (void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions
{
    for (SKPaymentTransaction* transaction in transactions)
    {
        switch(transaction.transactionState) {
            case SKPaymentTransactionStateDeferred:
               // Allow the user to continue to use the app


        }
    }
}
```

Example

Handling deferred transaction

```
- (void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions
{
    for (SKPaymentTransaction* transaction in transactions)
    {
        switch(transaction.transactionState) {
            case SKPaymentTransactionStateDeferred:
               // Allow the user to continue to use the app
               // It may be some time before the transaction is updated

        }
    }
}
```

Example

Handling deferred transaction

```
- (void)paymentQueue:(SKPaymentQueue *)queue
 updatedTransactions:(NSArray *)transactions
{
    for (SKPaymentTransaction* transaction in transactions)
    {
        switch(transaction.transactionState) {
            case SKPaymentTransactionStateDeferred:
                // Allow the user to continue to use the app
                // It may be some time before the transaction is updated
                // Do not get stuck in a modal “Purchasing…” state!
        }
    }
}
```

Example

Finish the transaction

Example

Finish the transaction

Always finish the transaction

Example

Finish the transaction

Always finish the transaction

```
- (void)finishTransaction:(SKPaymentTransaction *)transaction
```

Example

Finish the transaction

Always finish the transaction

```
- (void)finishTransaction:(SKPaymentTransaction *)transaction
```

Tells the store that your app has finished processing the transaction

- The transaction will be removed from the queue

SKPaymentQueue Tips

SKPaymentQueue Tips

```
@property(nonatomic, readonly) NSArray *transactions;
```

- No need for you to keep track of transactions in-flight, trust the queue

SKPaymentQueue Tips

```
@property(nonatomic, readonly) NSArray *transactions;
```

- No need for you to keep track of transactions in-flight, trust the queue

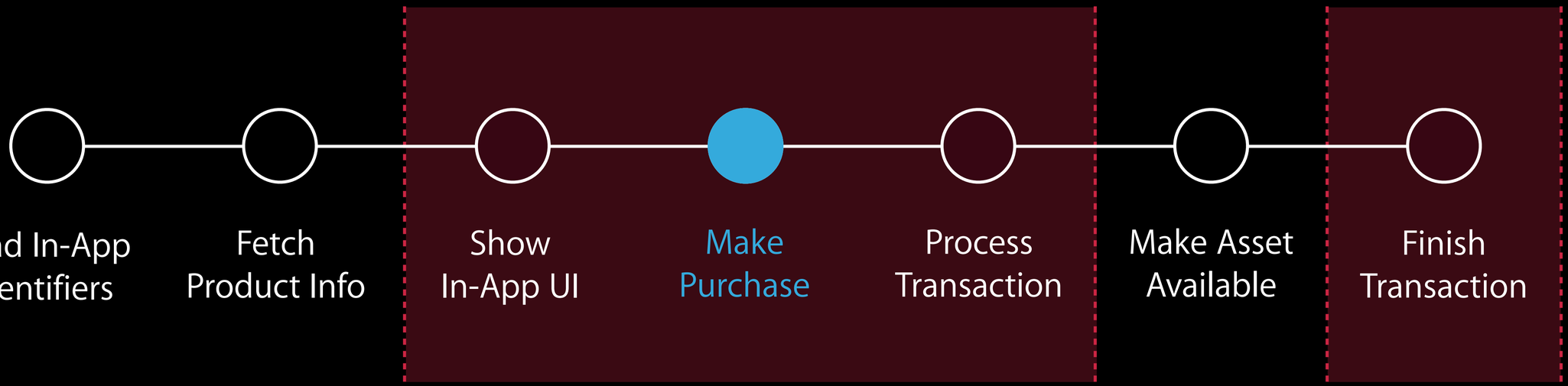
```
+ (BOOL)canMakePayments
```

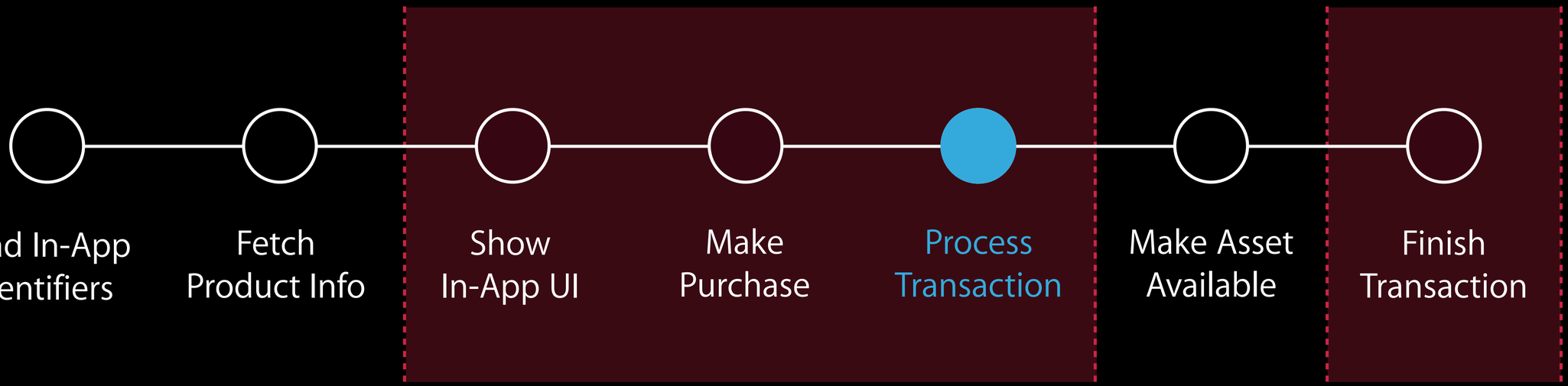
- Known if In-App Purchases have been restricted

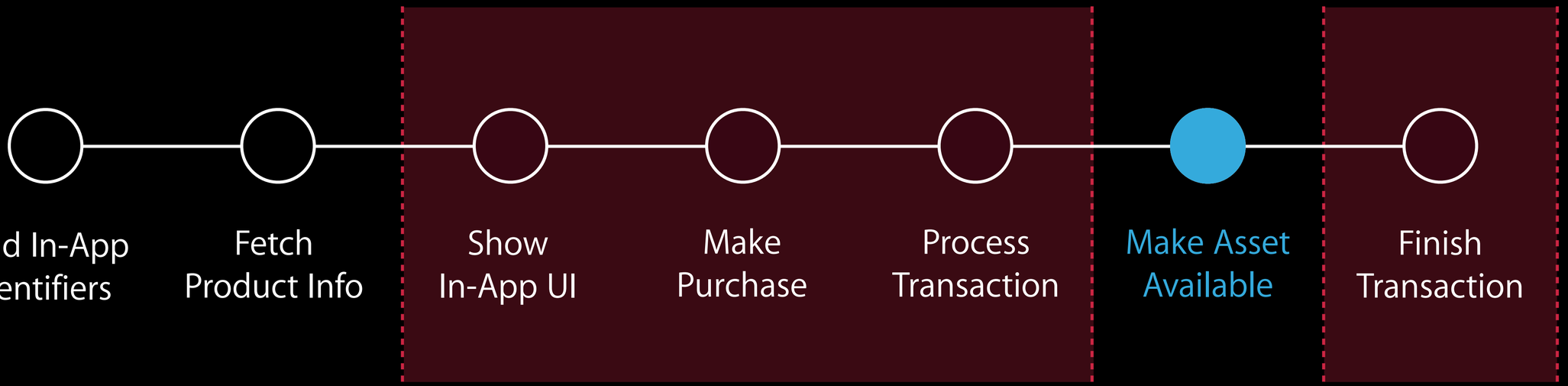
Demo

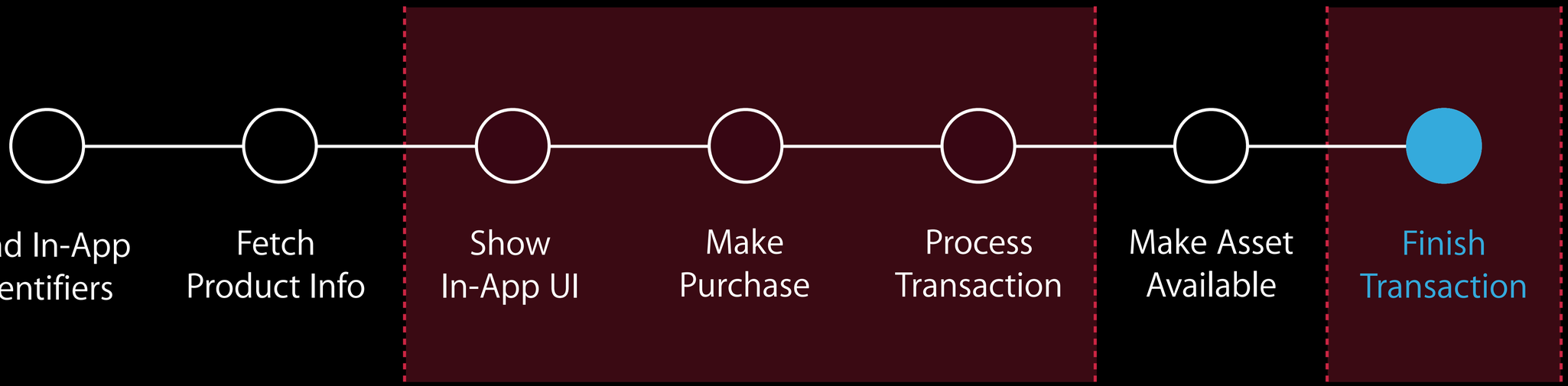
Trouble free In-App Purchase

Post-Sale Experience


Load In-App Identifiers
Fetch Product Info
Show In-App UI
Make Purchase
Process Transaction
Make Asset Available
Finish Transaction


Load In-App Identifiers
Fetch Product Info
Show In-App UI
Make Purchase
Process Transaction
Make Asset Available
Finish Transaction


Load In-App Identifiers
Fetch Product Info
Show In-App UI
Make Purchase
Process Transaction
Make Asset Available
Finish Transaction


Load In-App Identifiers
Fetch Product Info
Show In-App UI
Make Purchase
Process Transaction
Make Asset Available
Finish Transaction

Post-Sale Experience

Post-Sale Experience

Validate the purchase

- Receipt verification on-device or server-to-server

Post-Sale Experience

Validate the purchase

- Receipt verification on-device or server-to-server

Download content

- Hosted In-App Purchase content
- Self-hosted

Post-Sale Experience

Validate the purchase

- Receipt verification on-device or server-to-server

Download content

- Hosted In-App Purchase content
- Self-hosted

Persisting and restoring transactions

Receipt Validation

Receipt Validation

On-device validation

- Unlock features and content within the app

Receipt Validation

On-device validation

- Unlock features and content within the app

Server-to-server validation

- Restrict access to downloadable content

Receipt Validation

Do not use online validation directly from the device!

iOS 6 APIs for receipt validation are deprecated

Downloading Content

Hosted In-App Purchase content

Downloading Content

Hosted In-App Purchase content

- Hosted on Apple's servers

Downloading Content

Hosted In-App Purchase content

- Hosted on Apple's servers
- Scalable and reliable

Downloading Content

Hosted In-App Purchase content

- Hosted on Apple's servers
- Scalable and reliable
- Downloads in background

Downloading Content

Hosted In-App Purchase content

- Hosted on Apple's servers
- Scalable and reliable
- Downloads in background
- Up to 2GB per in-app purchasable product

Hosted Content

Start download

Hosted Content

Start download

```
- (void)paymentQueue:(SKPaymentQueue *)queue
        updatedTransactions:(NSArray *)transactions
for(SKPaymentTransaction* transaction in transactions)
{




}
```

Hosted Content

Start download

```
- (void)paymentQueue:(SKPaymentQueue *)queue
        updatedTransactions:(NSArray *)transactions
for(SKPaymentTransaction* transaction in transactions)
{
   if(transaction.downloads)



}
```

Hosted Content

Start download

```
- (void)paymentQueue:(SKPaymentQueue *)queue
        updatedTransactions:(NSArray *)transactions
for(SKPaymentTransaction* transaction in transactions)
{
   if(transaction.downloads)
        [[SKPaymentQueue defaultQueue] startDownloads:
                                transaction.downloads];
}
```

Hosted Content

Download progress

Hosted Content

Download progress

```
- (void)paymentQueue:(SKPaymentQueue *)queue
      updatedDownloads:(NSArray *)downloads;
```

Hosted Content

Download progress

```
- (void)paymentQueue:(SKPaymentQueue *)queue
      updatedDownloads:(NSArray *)downloads;
```

```
download.progress
download.timeRemaining
```

Hosted Content

Download progress

```
- (void)paymentQueue:(SKPaymentQueue *)queue
      updatedDownloads:(NSArray *)downloads;
```

```
download.progress
download.timeRemaining
download.state
download.error
```

Hosted Content

Download progress

```
- (void)paymentQueue:(SKPaymentQueue *)queue
      updatedDownloads:(NSArray *)downloads;
```

```
download.progress
download.timeRemaining
download.state
download.error
```

```
When download.state is SKDownloadStateFinished
  download.contentURL
```

Self-Hosted Content

Self-hosted downloadable content

- Use background download APIs

Self-Hosted Content

Self-hosted downloadable content

- Use background download APIs
  - Content is downloaded even when your app is not active

Self-Hosted Content

Self-hosted downloadable content

- Use background download APIs
  - Content is downloaded even when your app is not active
  - Using class NSURLConnection APIs has limitations

Self-Hosted Content

Using NSURLSession for downloading content

Self-Hosted Content

Using NSURLSession for downloading content

```
NSURLSessionConfiguration *config = [NSURLSessionConfiguration
                backgroundSessionConfiguration:@"MyBackgroundSession"];
```

Self-Hosted Content

Using NSURLSession for downloading content

```
NSURLSessionConfiguration *config = [NSURLSessionConfiguration
                backgroundSessionConfiguration:@"MyBackgroundSession"];
    NSURLSession *session = [NSURLSession sessionWithConfiguration:config
                                                          delegate:self delegateQueue:queue];
```

Self-Hosted Content

Using NSURLSession for downloading content

```
NSURLSessionConfiguration *config = [NSURLSessionConfiguration
                backgroundSessionConfiguration:@"MyBackgroundSession"];
NSURLSession *session = [NSURLSession sessionWithConfiguration:config
                                                      delegate:self delegateQueue:queue];

NSURLRequest *request = [NSURLRequest requestWithURL:myURL];
```

Self-Hosted Content

Using NSURLSession for downloading content

```
NSURLSessionConfiguration *config = [NSURLSessionConfiguration
                backgroundSessionConfiguration:@"MyBackgroundSession"];
NSURLSession *session = [NSURLSession sessionWithConfiguration:config
                                                      delegate:self delegateQueue:queue];

NSURLRequest *request = [NSURLRequest requestWithURL:myURL];
NSURLSessionDownloadTask *downloadTask = [session
                                                      downloadTaskWithRequest:request];
```

Self-Hosted Content

NSURLSessionDownloadDelegate

Download progress

```
- (void)URLSession:(NSURLSession *)session
      downloadTask:(NSURLSessionDownloadTask *)downloadTask
      didWriteData:(int64_t)bytesWritten
      totalBytesWritten:(int64_t)totalBytesWritten
      totalBytesExpectedToWrite:(int64_t)totalBytesExpectedToWrite
{
    // do something with progress
}
```

Self-Hosted Content

NSURLSession

Reconnect to session on launch

```
- (void)application:(UIApplication *)application
        handleEventsForBackgroundURLSession:(NSString *)identifier
     completionHandler:(void (^)())completionHandler
{
    NSURLSessionConfiguration *config = [NSURLSessionConfiguration
              backgroundSessionConfiguration:identifier];
    NSURLSession *session = [NSURLSession sessionWithConfiguration:config
                                 delegate:self delegateQueue:queue];
    self.completionHandler = completionHandler; // call when done
}
```

Downloading Content

When the content is downloaded, finish the transaction

```
[[SKPaymentQueue defaultQueue] finishTransaction:transaction];
```

Otherwise, the payment will stay in the queue

Restore Transactions

Restore Completed Transactions

Restoring transactions allows the user to restore

- Non-consumable in-app purchases
- Auto-renewing subscriptions

Restore Completed Transactions

Restoring transactions allows the user to restore

- Non-consumable in-app purchases
- Auto-renewing subscriptions

Consumables and non-renewable restrictions

Restore Completed Transactions

Restoring transactions allows the user to restore

- Non-consumable in-app purchases
- Auto-renewing subscriptions

Consumables and non-renewable restrictions

- You must persist the state!

Restore Completed Transactions

```
[[SKPaymentQueue defaultQueue] restoreCompletedTransactions]
```

Restore Completed Transactions

```
[[SKPaymentQueue defaultQueue] restoreCompletedTransactions]
```

Observe the queue

Restore Completed Transactions

```
[[SKPaymentQueue defaultQueue] restoreCompletedTransactions]
```

Observe the queue

```
- paymentQueue:restoreCompletedTransactionsFailedWithError:
```

Restore Completed Transactions

```
[[SKPaymentQueue defaultQueue] restoreCompletedTransactions]
```

Observe the queue

```
- paymentQueue:restoreCompletedTransactionsFailedWithError:
- paymentQueueRestoreCompletedTransactionsFinished
```

Restore Completed Transactions

```
[[SKPaymentQueue defaultQueue] restoreCompletedTransactions]
```

Observe the queue

```
- paymentQueue:restoreCompletedTransactionsFailedWithError:
- paymentQueueRestoreCompletedTransactionsFinished
```

Inspect the receipt and unlock content and features accordingly

Restore Completed Transactions

```
[[SKPaymentQueue defaultQueue] restoreCompletedTransactions]
```

Restore Completed Transactions

```
[[SKPaymentQueue defaultQueue] restoreCompletedTransactions]
```

Requires a network connection

Restore Completed Transactions

```
[[SKPaymentQueue defaultQueue] restoreCompletedTransactions]
```

Requires a network connection

May cause sign in prompt

Restore Completed Transactions

```
[[SKPaymentQueue defaultQueue] restoreCompletedTransactions]
```

Requires a network connection

May cause sign in prompt

Your app must offer to restore transactions

Restore Completed Transactions

```
[[SKPaymentQueue defaultQueue] restoreCompletedTransactions]
```

Requires a network connection

May cause sign in prompt

Your app must offer to restore transactions

- But do not call try to restore unless requested by the user

Summary

Recipe for trouble-free In-App Purchases

In-App Purchase Process

In-App Purchase Process

In-App Purchase Process

Server fetch of product identifiers

- Cache appropriately
- Avoid delay in presenting products

In-App Purchase Process

In-App Purchase Process

In-App Purchase Process

Fetch only the products you need

Fetch just ahead of time

- Avoid delay in presenting products

In-App Purchase Process

In-App Purchase Process

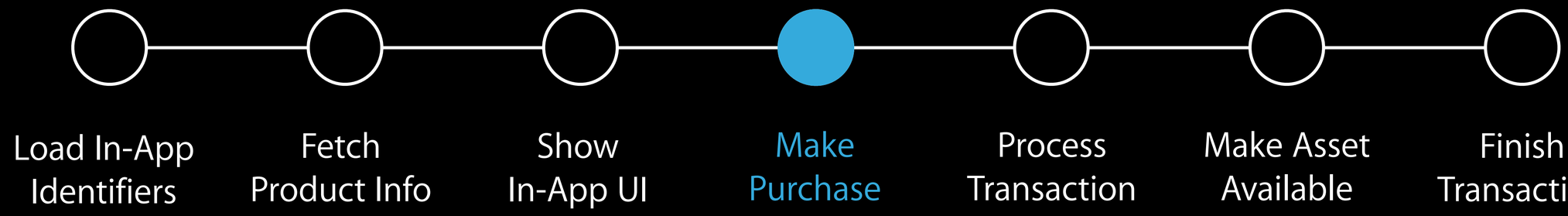
In-App Purchase Process

Take care to ensure proper localization

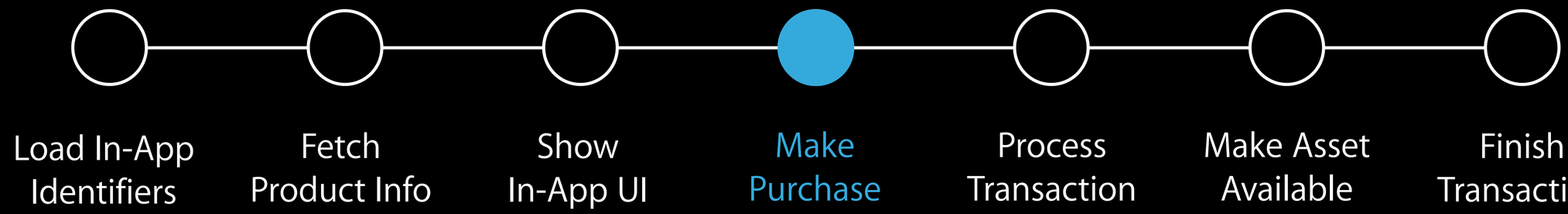
Do not convert currencies

In-App Purchase Process

In-App Purchase Process



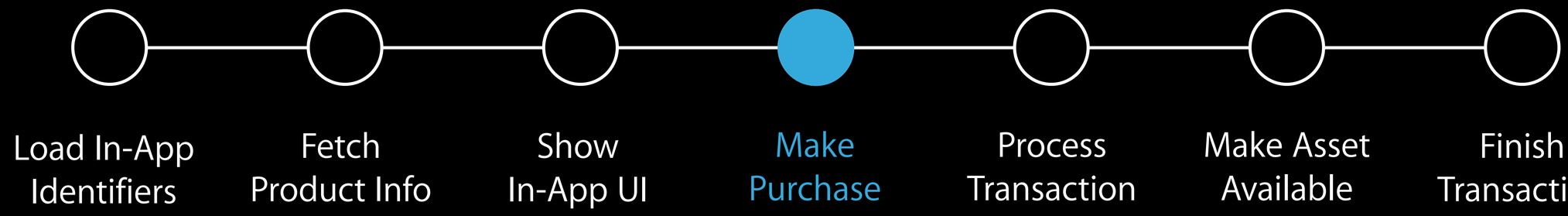
In-App Purchase Process



Add the payment to the queue

Then obey the queue, always

In-App Purchase Process



In-App Purchase Process

In-App Purchase Process

Verify the receipt

Unlock features and content

Avoid deprecated APIs and unsafe verification

In-App Purchase Process

In-App Purchase Process

In-App Purchase Process

Content download

- Host content with Apple
- Use background download APIs

In-App Purchase Process

In-App Purchase Process

In-App Purchase Process

Always finish the transaction!

Keep your queue clean

More Information

Evangelism
evangelism@apple.com

Documentation
In-App Purchases Programming Guide
https://developer.apple.com

Apple Developer Forums
http://devforums.apple.com

Related Sessions

• Preventing Unauthorized Purchases with Receipts	Pacific Heights	Friday 10:15AM
• Designing a Great In-App Purchase Experience	Nob Hill	Wednesday 11:30AM
• Kids and Apps	Nob Hill	Thursday 3:15PM

Labs

• StoreKit and Receipts Lab	Services Lab A	Wednesday 3:15PM
• StoreKit and Receipts Lab	Services Lab A	Friday 10:15AM
• Open Hours	Services Lab A	Friday 2:00PM

WWDC14