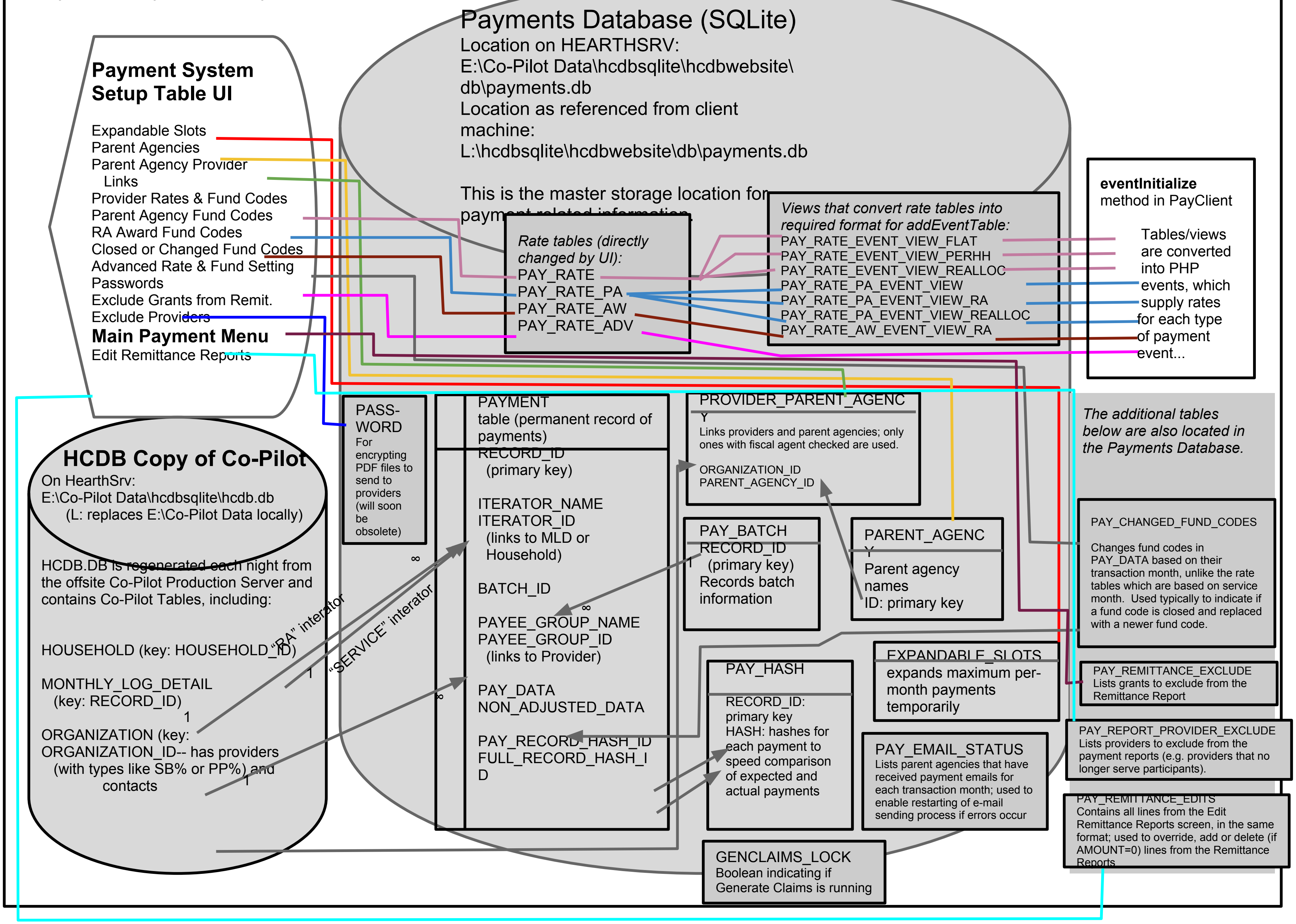
Payment System Data Flow
(as of 8/28/2013)

Payment System Key Database Tables



Payment System Setup Table UI

Expandable Slots
Parent Agencies
Parent Agency Provider Links
Provider Rates & Fund Codes
Parent Agency Fund Codes
RA Award Fund Codes
Closed or Changed Fund Codes
Advanced Rate & Fund Setting
Passwords
Exclude Grants from Remit.
Exclude Providers
Main Payment Menu
Edit Remittance Reports

Payments Database (SQLite)

Location on HEARTHSRV:
E:\Co-Pilot Data\hcdbsqlite\hcdbwebsite\db\payments.db
Location as referenced from client machine:
L:\hcdbsqlite\hcdbwebsite\db\payments.db

This is the master storage location for payment related information.

Rate tables (directly changed by UI):
PAY_RATE
PAY_RATE_PA
PAY_RATE_AW
PAY_RATE_ADV

Views that convert rate tables into required format for addEventTable:
PAY_RATE_EVENT_VIEW_FLAT
PAY_RATE_EVENT_VIEW_PERHH
PAY_RATE_EVENT_VIEW_REALLOC
PAY_RATE_PA_EVENT_VIEW
PAY_RATE_PA_EVENT_VIEW_RA
PAY_RATE_PA_EVENT_VIEW_REALLOC
PAY_RATE_AW_EVENT_VIEW_RA

eventInitialize method in PayClient

Tables/views are converted into PHP events, which supply rates for each type of payment event...

PASS-WORD
For encrypting PDF files to send to providers (will soon be obsolete)

PAYMENT table (permanent record of payments)
RECORD_ID (primary key)
ITERATOR_NAME
ITERATOR_ID (links to MLD or Household)
BATCH_ID
PAYEE_GROUP_NAME
PAYEE_GROUP_ID (links to Provider)
PAY_DATA
NON_ADJUSTED_DATA
PAY_RECORD_HASH_ID
FULL_RECORD_HASH_ID

PROVIDER_PARENT_AGENCY
Links providers and parent agencies; only ones with fiscal agent checked are used.
ORGANIZATION_ID
PARENT_AGENCY_ID

PAY_BATCH
RECORD_ID (primary key)
Records batch information

PARENT_AGENCY
Parent agency names
ID: primary key

PAY_HASH
RECORD_ID: primary key
HASH: hashes for each payment to speed comparison of expected and actual payments

EXPANDABLE_SLOTS
expands maximum per-month payments temporarily

PAY_EMAIL_STATUS
Lists parent agencies that have received payment emails for each transaction month; used to enable restarting of e-mail sending process if errors occur

GENCLAIMS_LOCK
Boolean indicating if Generate Claims is running

The additional tables below are also located in the Payments Database.

PAY_CHANGED_FUND_CODES

Changes fund codes in PAY_DATA based on their transaction month, unlike the rate tables which are based on service month. Used typically to indicate if a fund code is closed and replaced with a newer fund code.

PAY_REMITTANCE_EXCLUDE
Lists grants to exclude from the Remittance Report

PAY_REPORT_PROVIDER_EXCLUDE
Lists providers to exclude from the payment reports (e.g. providers that no longer serve participants).

PAY_REMITTANCE_EDITS
Contains all lines from the Edit Remittance Reports screen, in the same format; used to override, add or delete (if AMOUNT=0) lines from the Remittance Reports

HCDB Copy of Co-Pilot

On HearthSrv:
E:\Co-Pilot Data\hcdbsqlite\hcdb.db
(L: replaces E:\Co-Pilot Data locally)

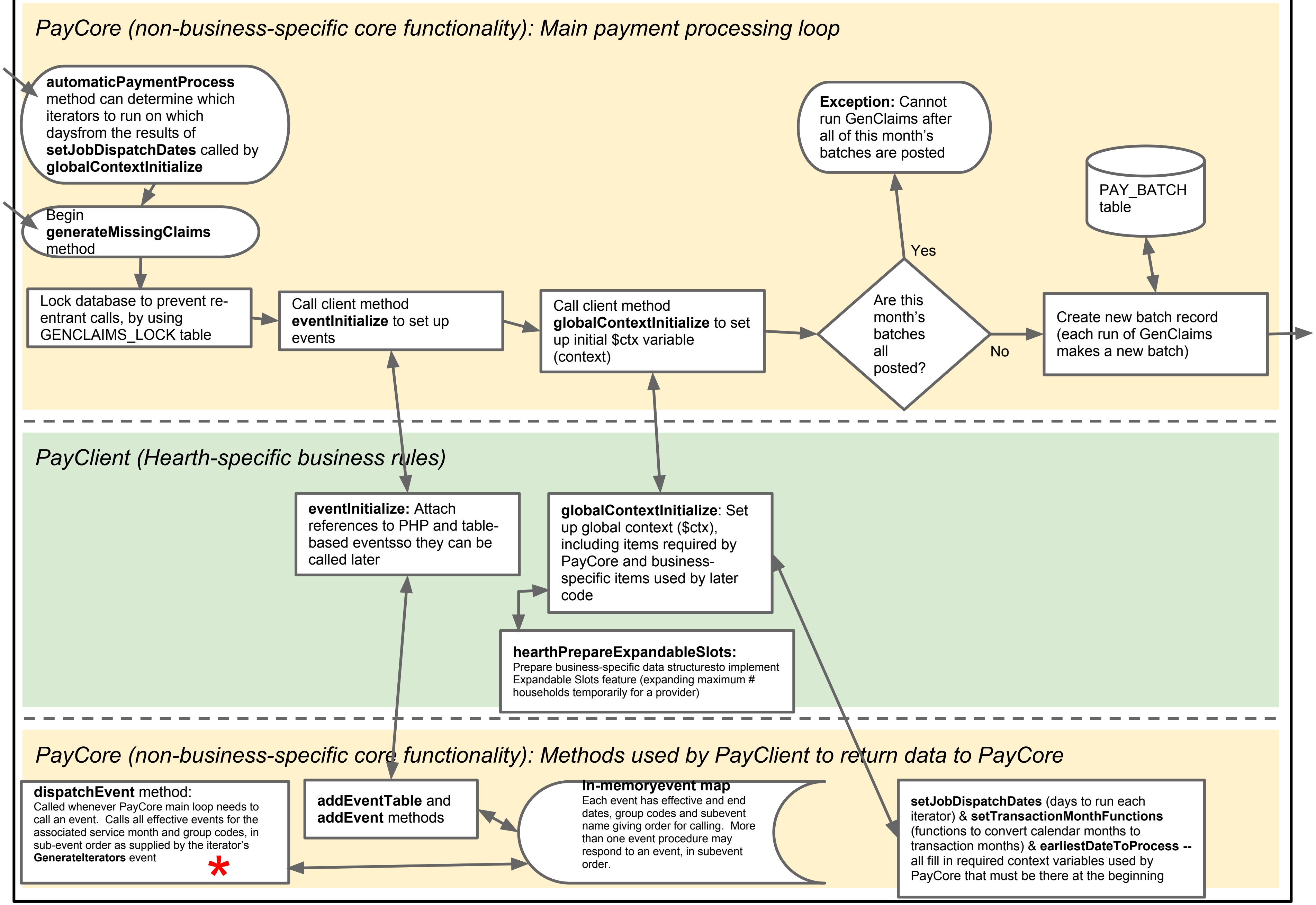
HCDB.DB is regenerated each night from the offsite Co-Pilot Production Server and contains Co-Pilot Tables, including:

HOUSEHOLD (key: HOUSEHOLD_ID)

MONTHLY_LOG_DETAIL (key: RECORD_ID)

ORGANIZATION (key: ORGANIZATION_ID-- has providers (with types like SB% or PP%) and contacts

Payment Process Program Flow (Generate Missing Claims)



PayCore (non-business-specific core functionality): Main payment processing loop

automaticPaymentProcess method can determine which iterators to run on which daysfrom the results of **setJobDispatchDates** called by **globalContextInitialize**

Begin **generateMissingClaims** method

Lock database to prevent re-entrant calls, by using GENCLAIMS_LOCK table

Call client method **eventInitialize** to set up events

Call client method **globalContextInitialize** to set up initial $ctx variable (context)

Are this month's batches all posted?

Yes

Exception: Cannot run GenClaims after all of this month's batches are posted

No

Create new batch record (each run of GenClaims makes a new batch)

PAY_BATCH table

PayClient (Hearth-specific business rules)

eventInitialize: Attach references to PHP and table-based eventsso they can be called later

globalContextInitialize: Set up global context ($ctx), including items required by PayCore and business-specific items used by later code

hearthPrepareExpandableSlots: Prepare business-specific data structuresto implement Expandable Slots feature (expanding maximum # households temporarily for a provider)

PayCore (non-business-specific core functionality): Methods used by PayClient to return data to PayCore

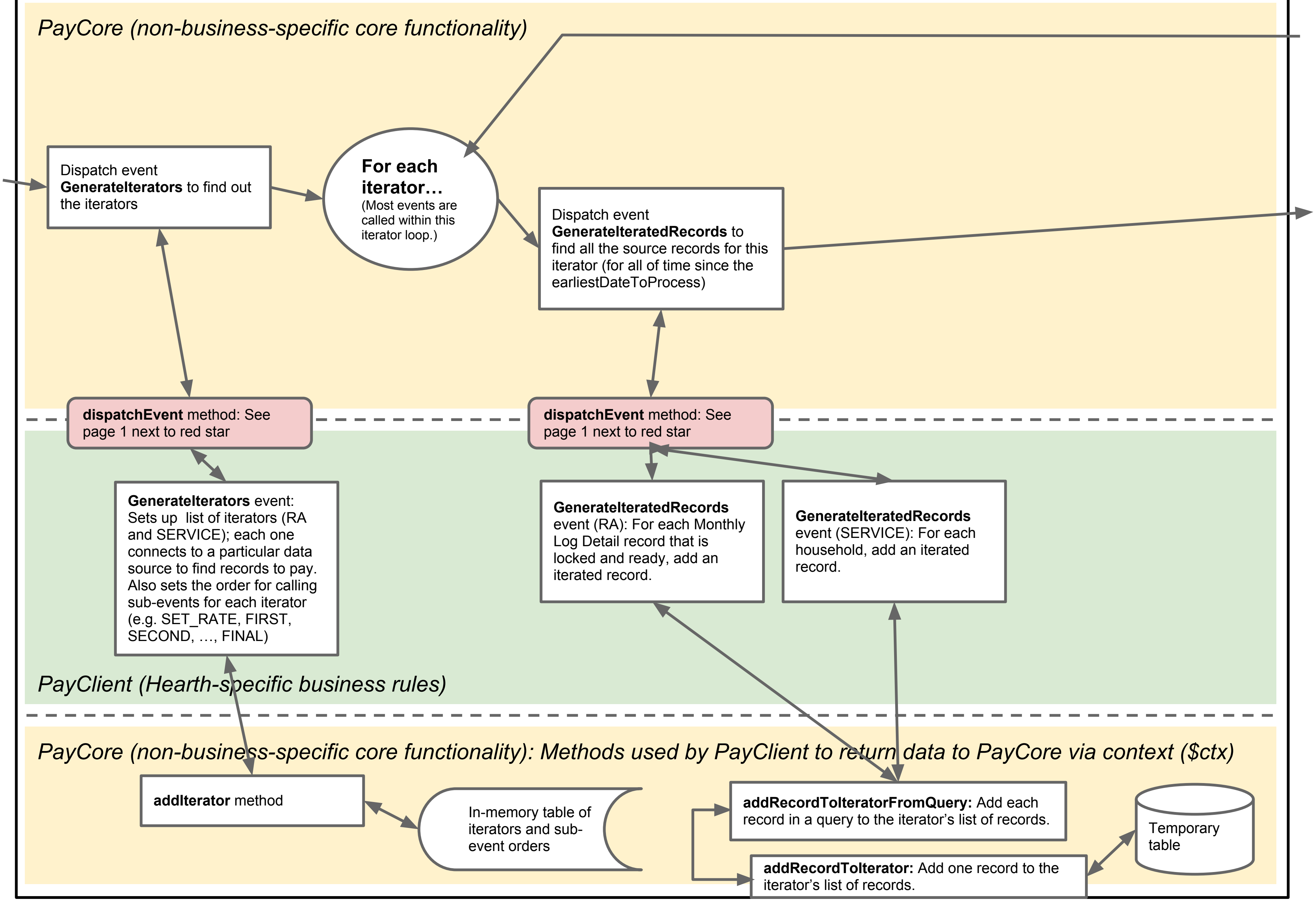
dispatchEvent method: Called whenever PayCore main loop needs to call an event. Calls all effective events for the associated service month and group codes, in sub-event order as supplied by the iterator's **GenerateIterators** event *

addEventTable and **addEvent** methods

In-memoryevent map Each event has effective and end dates, group codes and subevent name giving order for calling. More than one event procedure may respond to an event, in subevent order.

setJobDispatchDates (days to run each iterator) & **setTransactionMonthFunctions** (functions to convert calendar months to transaction months) & **earliestDateToProcess --** all fill in required context variables used by PayCore that must be there at the beginning

Payment Process Program Flow (Generate Missing Claims)



PayCore (non-business-specific core functionality)

Dispatch event **GenerateIterators** to find out the iterators

For each iterator…
(Most events are called within this iterator loop.)

Dispatch event **GenerateIteratedRecords** to find all the source records for this iterator (for all of time since the earliestDateToProcess)

dispatchEvent method: See page 1 next to red star

dispatchEvent method: See page 1 next to red star

GenerateIterators event: Sets up list of iterators (RA and SERVICE); each one connects to a particular data source to find records to pay. Also sets the order for calling sub-events for each iterator (e.g. SET_RATE, FIRST, SECOND, …, FINAL)

GenerateIteratedRecords event (RA): For each Monthly Log Detail record that is locked and ready, add an iterated record.

GenerateIteratedRecords event (SERVICE): For each household, add an iterated record.

PayClient (Hearth-specific business rules)

PayCore (non-business-specific core functionality): Methods used by PayClient to return data to PayCore via context ($ctx)

addIterator method

In-memory table of iterators and sub-event orders

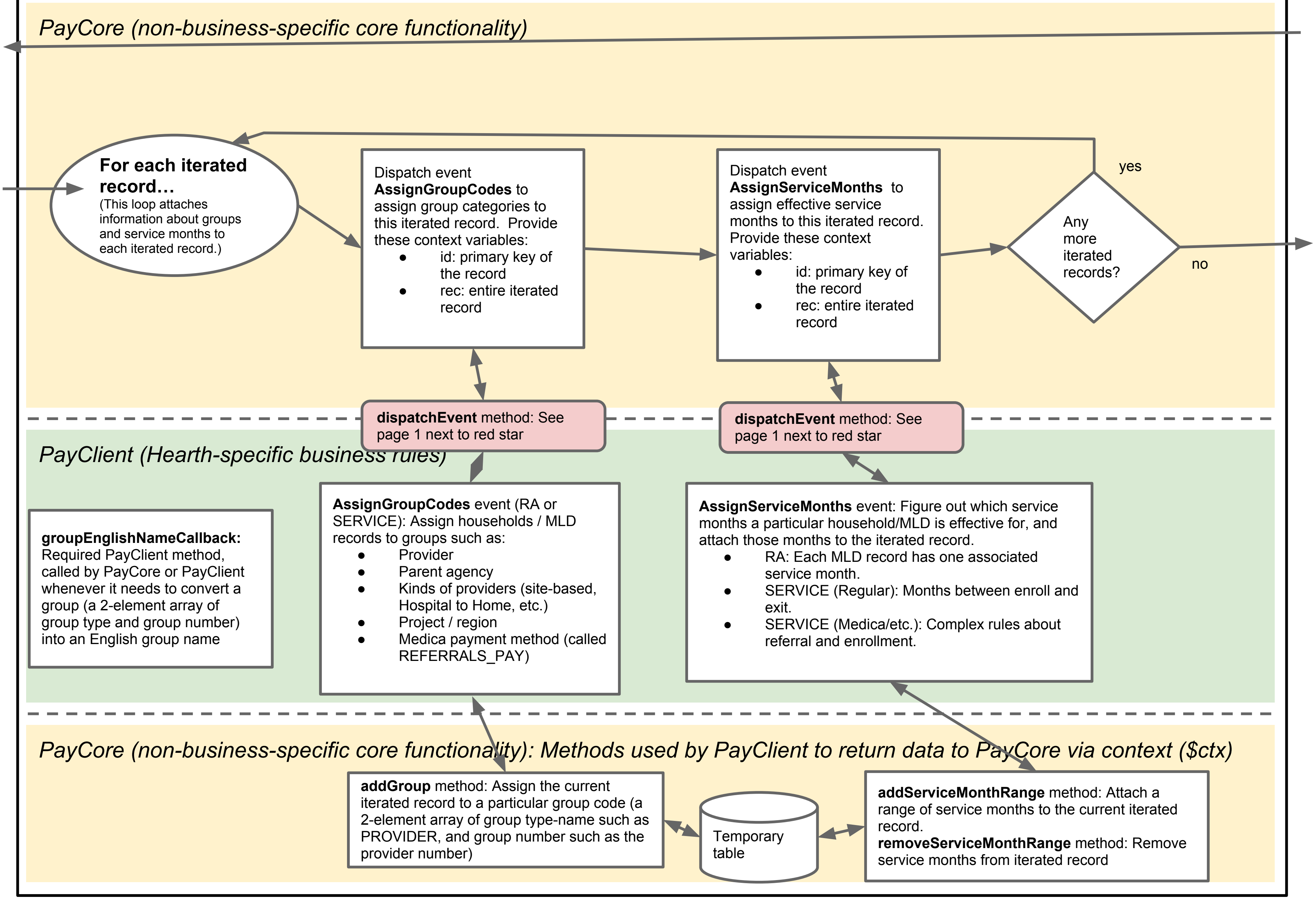
addRecordToIteratorFromQuery: Add each record in a query to the iterator's list of records.

addRecordToIterator: Add one record to the iterator's list of records.

Temporary table

Payment Process Program Flow (Generate Missing Claims)

Page 3 of 10



PayCore (non-business-specific core functionality)

For each iterated record…
(This loop attaches information about groups and service months to each iterated record.)

Dispatch event **AssignGroupCodes** to assign group categories to this iterated record. Provide these context variables:

- id: primary key of the record
- rec: entire iterated record

Dispatch event **AssignServiceMonths** to assign effective service months to this iterated record. Provide these context variables:

- id: primary key of the record
- rec: entire iterated record

Any more iterated records?

dispatchEvent method: See page 1 next to red star

dispatchEvent method: See page 1 next to red star

PayClient (Hearth-specific business rules)

groupEnglishNameCallback: Required PayClient method, called by PayCore or PayClient whenever it needs to convert a group (a 2-element array of group type and group number) into an English group name

AssignGroupCodes event (RA or SERVICE): Assign households / MLD records to groups such as:

- Provider
- Parent agency
- Kinds of providers (site-based, Hospital to Home, etc.)
- Project / region
- Medica payment method (called REFERRALS_PAY)

AssignServiceMonths event: Figure out which service months a particular household/MLD is effective for, and attach those months to the iterated record.

- RA: Each MLD record has one associated service month.
- SERVICE (Regular): Months between enroll and exit.
- SERVICE (Medica/etc.): Complex rules about referral and enrollment.

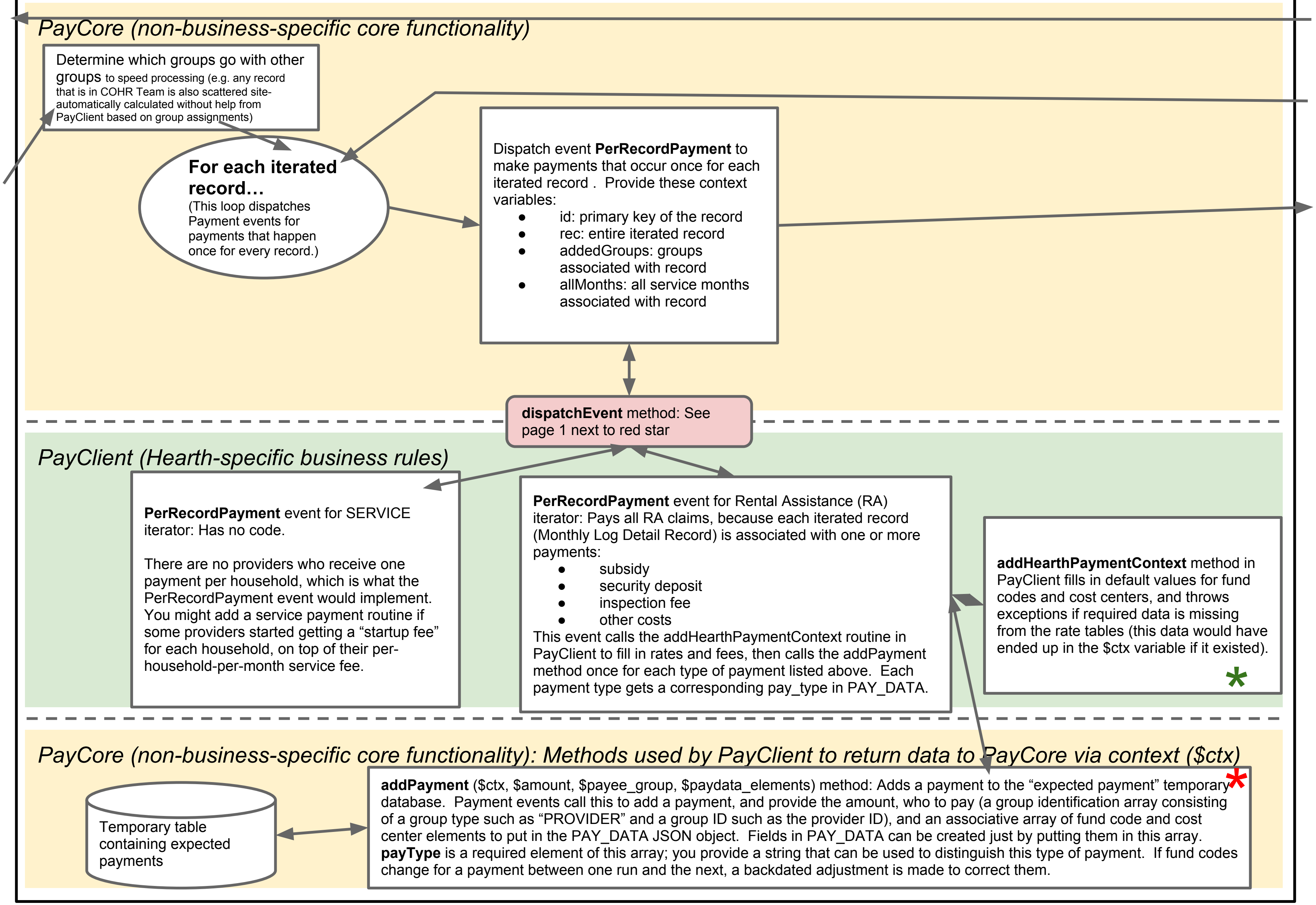
PayCore (non-business-specific core functionality): Methods used by PayClient to return data to PayCore via context ($ctx)

addGroup method: Assign the current iterated record to a particular group code (a 2-element array of group type-name such as PROVIDER, and group number such as the provider number)

Temporary table

addServiceMonthRange method: Attach a range of service months to the current iterated record.
removeServiceMonthRange method: Remove service months from iterated record

Payment Process Program Flow (Generate Missing Claims)



PayCore (non-business-specific core functionality)

Determine which groups go with other groups to speed processing (e.g. any record that is in COHR Team is also scattered site- automatically calculated without help from PayClient based on group assignments)

For each iterated record…
(This loop dispatches Payment events for payments that happen once for every record.)

Dispatch event **PerRecordPayment** to make payments that occur once for each iterated record . Provide these context variables:

- id: primary key of the record
- rec: entire iterated record
- addedGroups: groups associated with record
- allMonths: all service months associated with record

dispatchEvent method: See page 1 next to red star

PayClient (Hearth-specific business rules)

PerRecordPayment event for SERVICE iterator: Has no code.

There are no providers who receive one payment per household, which is what the PerRecordPayment event would implement. You might add a service payment routine if some providers started getting a “startup fee” for each household, on top of their per-household-per-month service fee.

PerRecordPayment event for Rental Assistance (RA) iterator: Pays all RA claims, because each iterated record (Monthly Log Detail Record) is associated with one or more payments:

- subsidy
- security deposit
- inspection fee
- other costs

This event calls the addHearthPaymentContext routine in PayClient to fill in rates and fees, then calls the addPayment method once for each type of payment listed above. Each payment type gets a corresponding pay_type in PAY_DATA.

addHearthPaymentContext method in PayClient fills in default values for fund codes and cost centers, and throws exceptions if required data is missing from the rate tables (this data would have ended up in the $ctx variable if it existed).

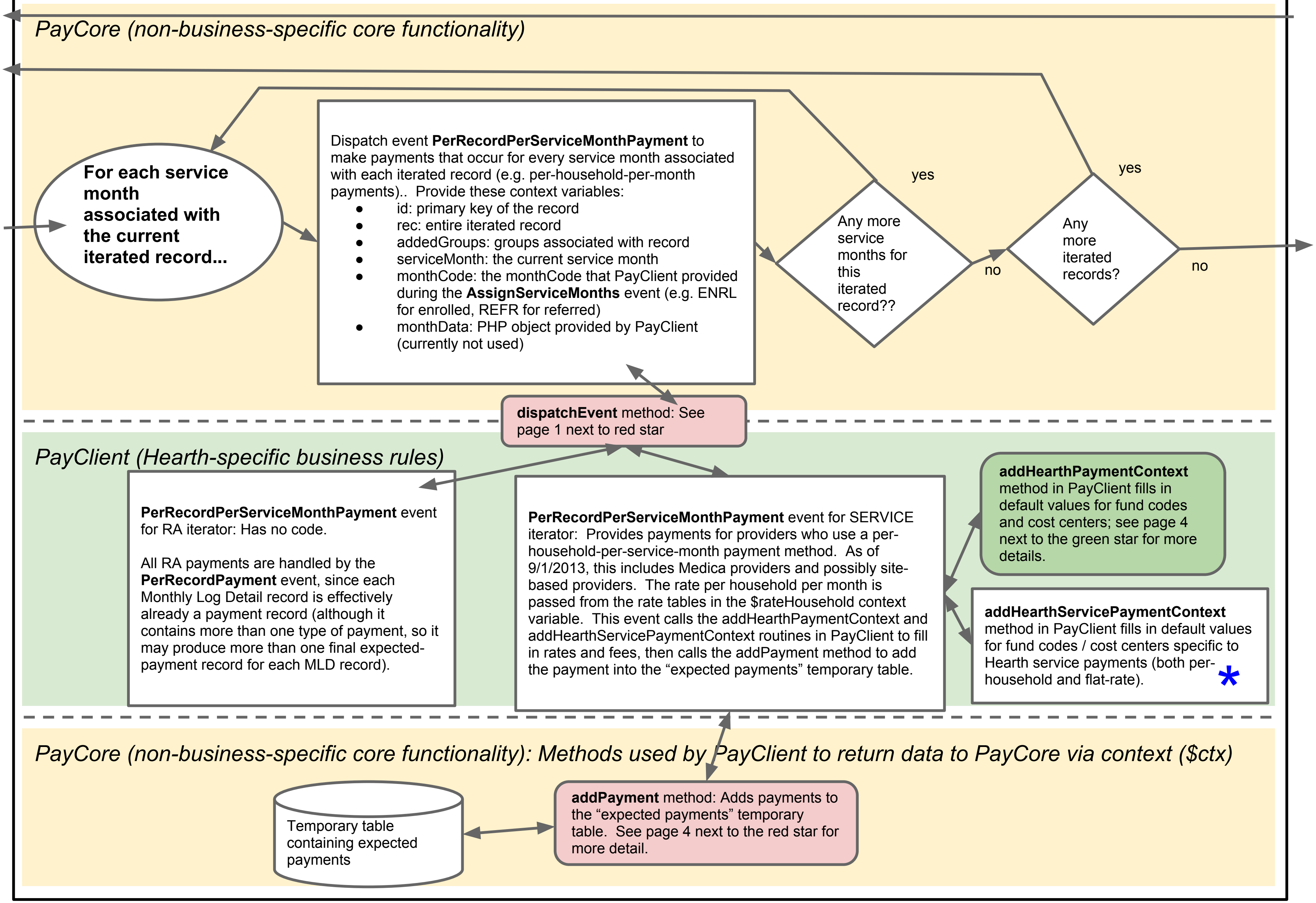
*

PayCore (non-business-specific core functionality): Methods used by PayClient to return data to PayCore via context ($ctx)

Temporary table containing expected payments

addPayment ($ctx, $amount, $payee_group, $paydata_elements) method: Adds a payment to the “expected payment” temporary database. Payment events call this to add a payment, and provide the amount, who to pay (a group identification array consisting of a group type such as “PROVIDER” and a group ID such as the provider ID), and an associative array of fund code and cost center elements to put in the PAY_DATA JSON object. Fields in PAY_DATA can be created just by putting them in this array. **payType** is a required element of this array; you provide a string that can be used to distinguish this type of payment. If fund codes change for a payment between one run and the next, a backdated adjustment is made to correct them. *

Payment Process Program Flow (Generate Missing Claims)



PayCore (non-business-specific core functionality)

For each service month associated with the current iterated record...

Dispatch event **PerRecordPerServiceMonthPayment** to make payments that occur for every service month associated with each iterated record (e.g. per-household-per-month payments).. Provide these context variables:

- id: primary key of the record
- rec: entire iterated record
- addedGroups: groups associated with record
- serviceMonth: the current service month
- monthCode: the monthCode that PayClient provided during the **AssignServiceMonths** event (e.g. ENRL for enrolled, REFR for referred)
- monthData: PHP object provided by PayClient (currently not used)

Any more service months for this iterated record??

Any more iterated records?

dispatchEvent method: See page 1 next to red star

PayClient (Hearth-specific business rules)

PerRecordPerServiceMonthPayment event for RA iterator: Has no code.

All RA payments are handled by the **PerRecordPayment** event, since each Monthly Log Detail record is effectively already a payment record (although it contains more than one type of payment, so it may produce more than one final expected-payment record for each MLD record).

PerRecordPerServiceMonthPayment event for SERVICE iterator: Provides payments for providers who use a per-household-per-service-month payment method. As of 9/1/2013, this includes Medica providers and possibly site-based providers. The rate per household per month is passed from the rate tables in the $rateHousehold context variable. This event calls the addHearthPaymentContext and addHearthServicePaymentContext routines in PayClient to fill in rates and fees, then calls the addPayment method to add the payment into the “expected payments” temporary table.

addHearthPaymentContext method in PayClient fills in default values for fund codes and cost centers; see page 4 next to the green star for more details.

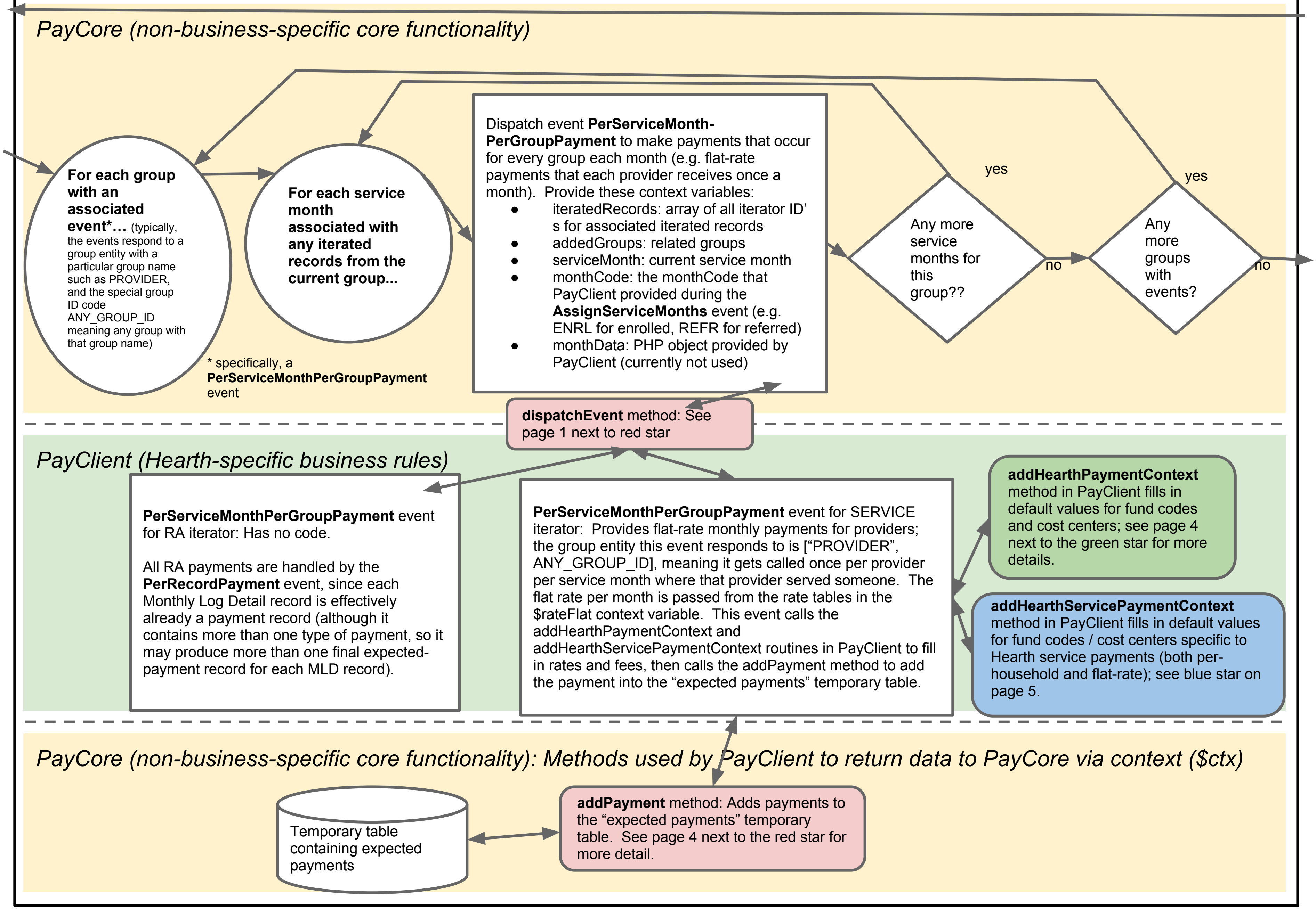
addHearthServicePaymentContext method in PayClient fills in default values for fund codes / cost centers specific to Hearth service payments (both per-household and flat-rate). *

PayCore (non-business-specific core functionality): Methods used by PayClient to return data to PayCore via context ($ctx)

Temporary table containing expected payments

addPayment method: Adds payments to the “expected payments” temporary table. See page 4 next to the red star for more detail.

Payment Process Program Flow (Generate Missing Claims)



PayCore (non-business-specific core functionality)

For each group with an associated event*… (typically, the events respond to a group entity with a particular group name such as PROVIDER, and the special group ID code ANY_GROUP_ID meaning any group with that group name)

* specifically, a **PerServiceMonthPerGroupPayment** event

For each service month associated with any iterated records from the current group...

Dispatch event **PerServiceMonth-PerGroupPayment** to make payments that occur for every group each month (e.g. flat-rate payments that each provider receives once a month). Provide these context variables:

- iteratedRecords: array of all iterator ID’s for associated iterated records
- addedGroups: related groups
- serviceMonth: current service month
- monthCode: the monthCode that PayClient provided during the **AssignServiceMonths** event (e.g. ENRL for enrolled, REFR for referred)
- monthData: PHP object provided by PayClient (currently not used)

Any more service months for this group??

Any more groups with events?

dispatchEvent method: See page 1 next to red star

PayClient (Hearth-specific business rules)

PerServiceMonthPerGroupPayment event for RA iterator: Has no code.

All RA payments are handled by the **PerRecordPayment** event, since each Monthly Log Detail record is effectively already a payment record (although it contains more than one type of payment, so it may produce more than one final expected-payment record for each MLD record).

PerServiceMonthPerGroupPayment event for SERVICE iterator: Provides flat-rate monthly payments for providers; the group entity this event responds to is [“PROVIDER”, ANY_GROUP_ID], meaning it gets called once per provider per service month where that provider served someone. The flat rate per month is passed from the rate tables in the $rateFlat context variable. This event calls the addHearthPaymentContext and addHearthServicePaymentContext routines in PayClient to fill in rates and fees, then calls the addPayment method to add the payment into the “expected payments” temporary table.

addHearthPaymentContext method in PayClient fills in default values for fund codes and cost centers; see page 4 next to the green star for more details.

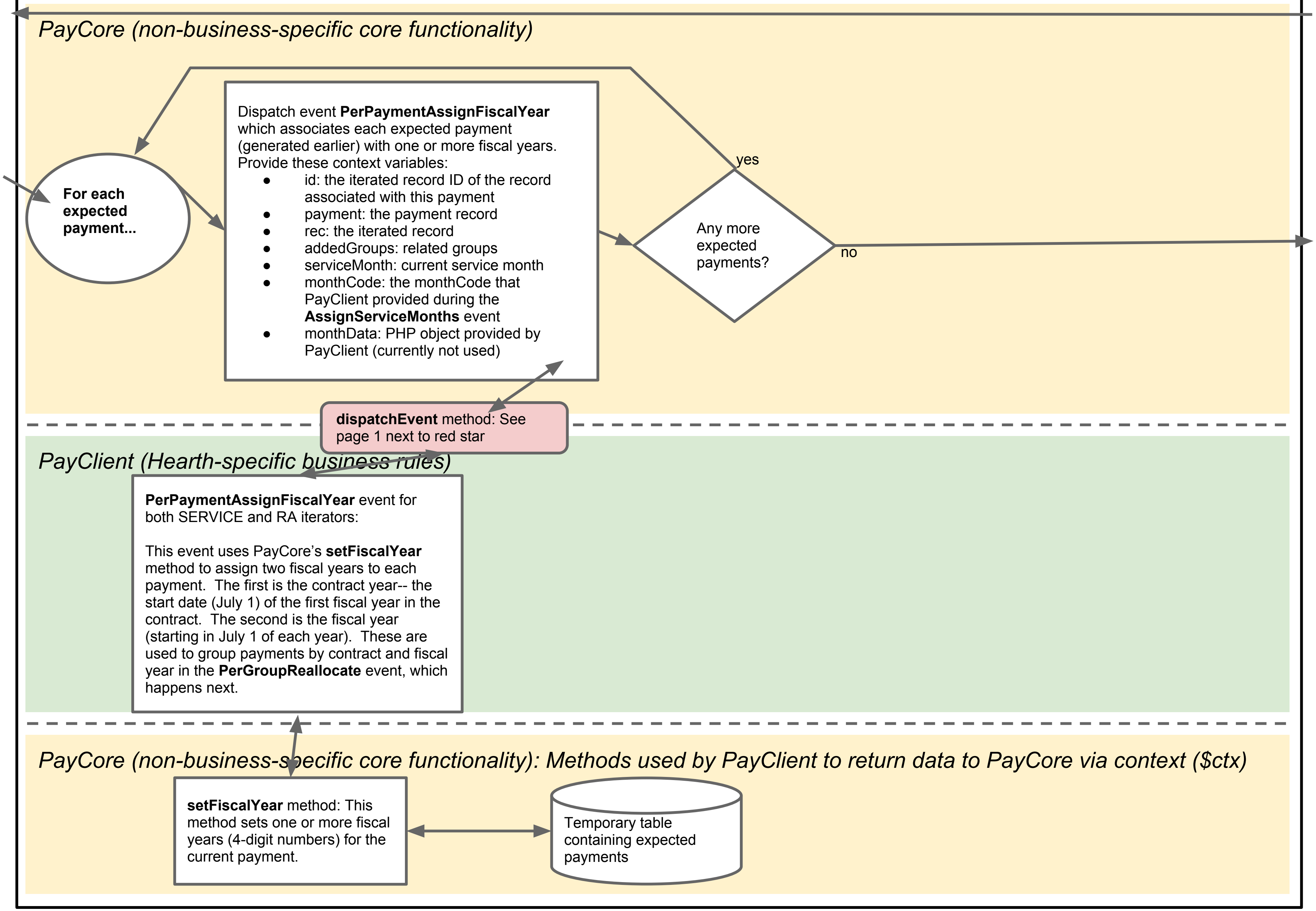
addHearthServicePaymentContext method in PayClient fills in default values for fund codes / cost centers specific to Hearth service payments (both per-household and flat-rate); see blue star on page 5.

PayCore (non-business-specific core functionality): Methods used by PayClient to return data to PayCore via context ($ctx)

Temporary table containing expected payments

addPayment method: Adds payments to the “expected payments” temporary table. See page 4 next to the red star for more detail.

Payment Process Program Flow (Generate Missing Claims)



PayCore (non-business-specific core functionality)

Dispatch event **PerPaymentAssignFiscalYear** which associates each expected payment (generated earlier) with one or more fiscal years. Provide these context variables:

- id: the iterated record ID of the record associated with this payment
- payment: the payment record
- rec: the iterated record
- addedGroups: related groups
- serviceMonth: current service month
- monthCode: the monthCode that PayClient provided during the **AssignServiceMonths** event
- monthData: PHP object provided by PayClient (currently not used)

dispatchEvent method: See page 1 next to red star

PayClient (Hearth-specific business rules)

PerPaymentAssignFiscalYear event for both SERVICE and RA iterators:

This event uses PayCore’s **setFiscalYear** method to assign two fiscal years to each payment. The first is the contract year-- the start date (July 1) of the first fiscal year in the contract. The second is the fiscal year (starting in July 1 of each year). These are used to group payments by contract and fiscal year in the **PerGroupReallocate** event, which happens next.

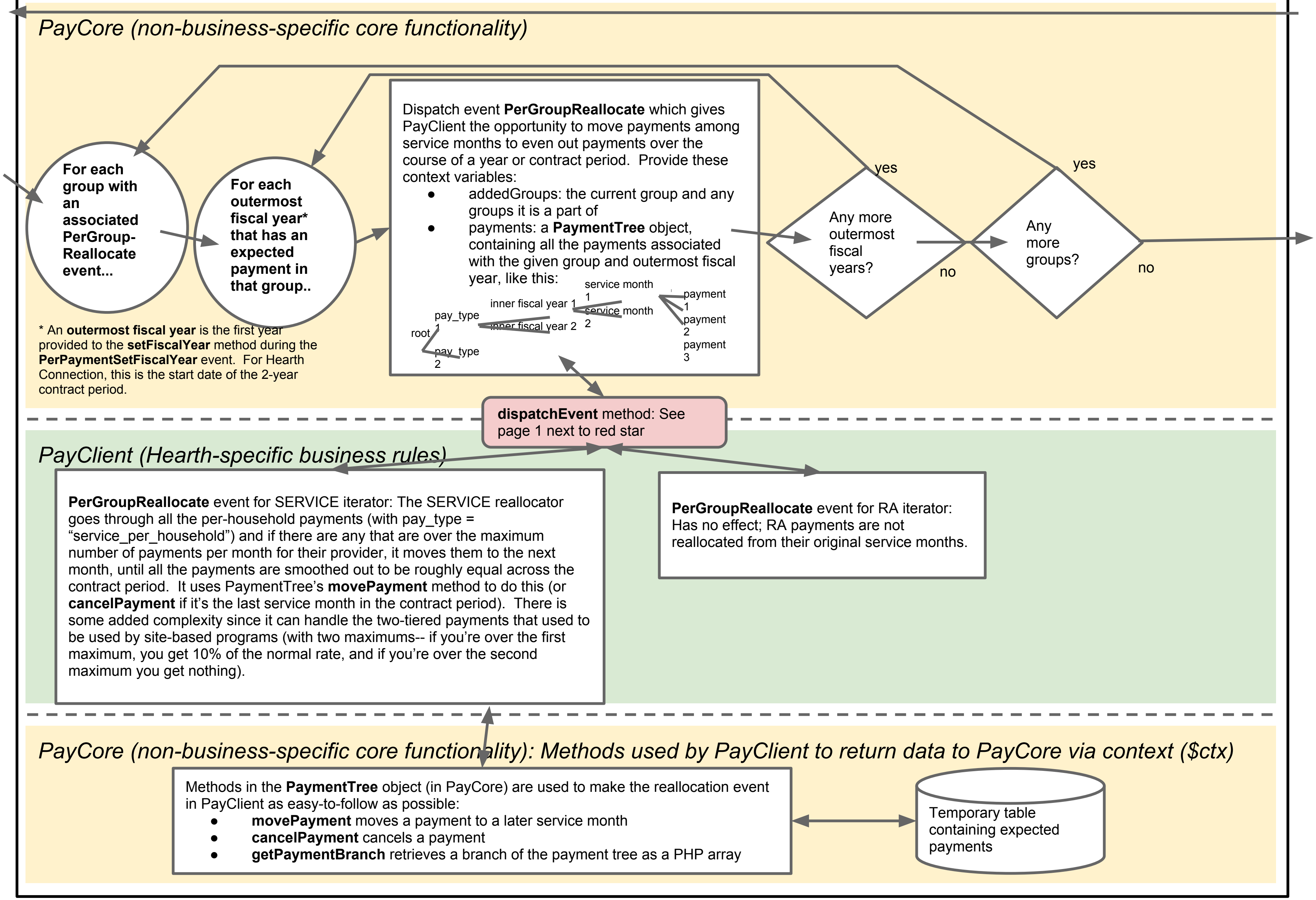
PayCore (non-business-specific core functionality): Methods used by PayClient to return data to PayCore via context ($ctx)

setFiscalYear method: This method sets one or more fiscal years (4-digit numbers) for the current payment.

Temporary table containing expected payments

Payment Process Program Flow (Generate Missing Claims)

Page 8 of 10



PayCore (non-business-specific core functionality)

For each group with an associated PerGroup-Reallocate event...

For each outermost fiscal year* that has an expected payment in that group..

Dispatch event **PerGroupReallocate** which gives PayClient the opportunity to move payments among service months to even out payments over the course of a year or contract period. Provide these context variables:

- addedGroups: the current group and any groups it is a part of
- payments: a **PaymentTree** object, containing all the payments associated with the given group and outermost fiscal year, like this:

* An **outermost fiscal year** is the first year provided to the **setFiscalYear** method during the **PerPaymentSetFiscalYear** event. For Hearth Connection, this is the start date of the 2-year contract period.

dispatchEvent method: See page 1 next to red star

PayClient (Hearth-specific business rules)

PerGroupReallocate event for SERVICE iterator: The SERVICE reallocator goes through all the per-household payments (with pay_type = “service_per_household”) and if there are any that are over the maximum number of payments per month for their provider, it moves them to the next month, until all the payments are smoothed out to be roughly equal across the contract period. It uses PaymentTree’s **movePayment** method to do this (or **cancelPayment** if it’s the last service month in the contract period). There is some added complexity since it can handle the two-tiered payments that used to be used by site-based programs (with two maximums-- if you’re over the first maximum, you get 10% of the normal rate, and if you’re over the second maximum you get nothing).

PerGroupReallocate event for RA iterator: Has no effect; RA payments are not reallocated from their original service months.

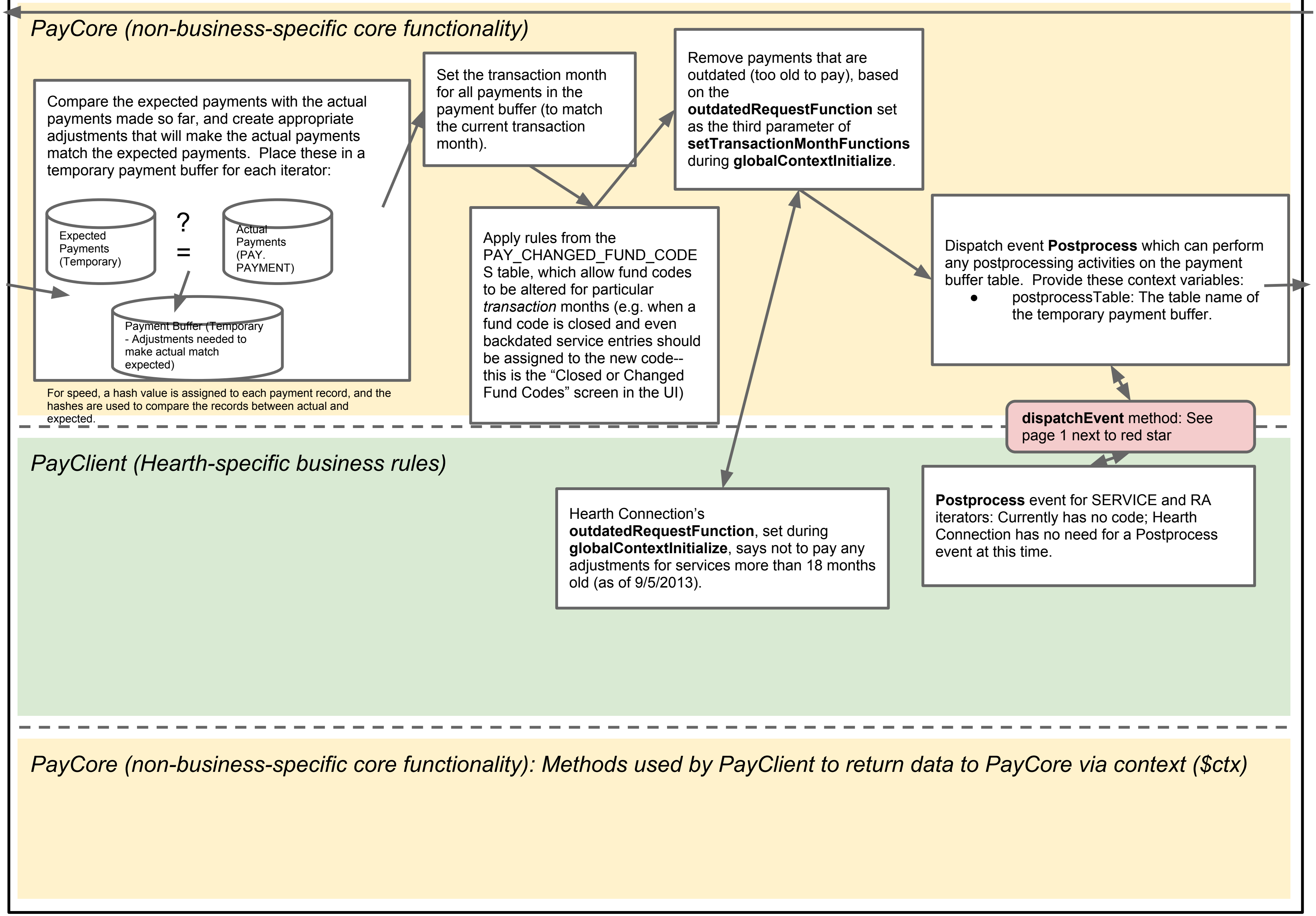
PayCore (non-business-specific core functionality): Methods used by PayClient to return data to PayCore via context ($ctx)

Methods in the **PaymentTree** object (in PayCore) are used to make the reallocation event in PayClient as easy-to-follow as possible:

- **movePayment** moves a payment to a later service month
- **cancelPayment** cancels a payment
- **getPaymentBranch** retrieves a branch of the payment tree as a PHP array

Temporary table containing expected payments

Payment Process Program Flow (Generate Missing Claims)



PayCore (non-business-specific core functionality)

Compare the expected payments with the actual payments made so far, and create appropriate adjustments that will make the actual payments match the expected payments. Place these in a temporary payment buffer for each iterator:

Expected Payments (Temporary)
?
=
Actual Payments (PAY. PAYMENT)
Payment Buffer (Temporary - Adjustments needed to make actual match expected)

For speed, a hash value is assigned to each payment record, and the hashes are used to compare the records between actual and expected.

Set the transaction month for all payments in the payment buffer (to match the current transaction month).

Apply rules from the PAY_CHANGED_FUND_CODES table, which allow fund codes to be altered for particular *transaction* months (e.g. when a fund code is closed and even backdated service entries should be assigned to the new code--this is the “Closed or Changed Fund Codes” screen in the UI)

Remove payments that are outdated (too old to pay), based on the **outdatedRequestFunction** set as the third parameter of **setTransactionMonthFunctions** during **globalContextInitialize**.

Dispatch event **Postprocess** which can perform any postprocessing activities on the payment buffer table. Provide these context variables:

- postprocessTable: The table name of the temporary payment buffer.

dispatchEvent method: See page 1 next to red star

PayClient (Hearth-specific business rules)

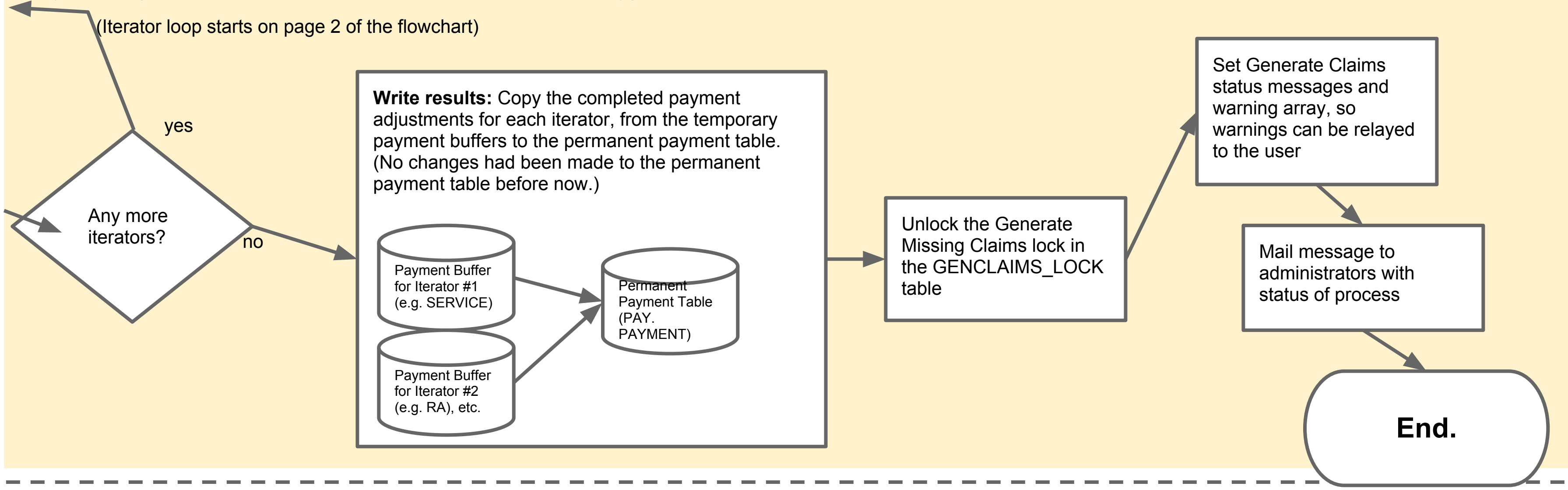
Hearth Connection’s **outdatedRequestFunction**, set during **globalContextInitialize**, says not to pay any adjustments for services more than 18 months old (as of 9/5/2013).

Postprocess event for SERVICE and RA iterators: Currently has no code; Hearth Connection has no need for a Postprocess event at this time.

PayCore (non-business-specific core functionality): Methods used by PayClient to return data to PayCore via context ($ctx)

Payment Process Program Flow (Generate Missing Claims)

PayCore (non-business-specific core functionality)



(Iterator loop starts on page 2 of the flowchart)

Any more iterators?

Write results: Copy the completed payment adjustments for each iterator, from the temporary payment buffers to the permanent payment table. (No changes had been made to the permanent payment table before now.)

Payment Buffer for Iterator #1 (e.g. SERVICE)
Payment Buffer for Iterator #2 (e.g. RA), etc.
Permanent Payment Table (PAY. PAYMENT)

Unlock the Generate Missing Claims lock in the GENCLAIMS_LOCK table

Set Generate Claims status messages and warning array, so warnings can be relayed to the user

Mail message to administrators with status of process

End.

PayClient (Hearth-specific business rules)

PayCore (non-business-specific core functionality): Methods used by PayClient to return data to PayCore via context ($ctx)

Payment Process Program Flow (Generate Missing Claims)

PayCore (non-business-specific core functionality)

PayClient (Hearth-specific business rules)

PayCore (non-business-specific core functionality): Methods used by PayClient to return data to PayCore via context ($ctx)