IBM MQ V9.3 機能と構成

4. チャネル

2023年09月

日本アイ・ビー・エム　システムズ・エンジニアリング（株）

トピック

- チャネル接続
  - MQシステムの構成
  - リモート・システムとの通信
- キュー・マネージャー間接続
  - キュー・マネージャー間接続の構成要素
  - キュー
  - メッセージ・チャネル
  - リモートのキューへの送信
  - リモート・キューの定義
  - チャネルのトリガリング
  - 転送メッセージのバッチ化
  - チャネルのバッチ・インターバル
  - FASTチャネル
  - デッド・レター・キュー
  - 文字コード変換
  - チャネルEXIT
  - チャネルのハートビート

(続き)

- メッセージの暗号化
- チャネルのメッセージ圧縮
- ローカル・アドレスの指定

- MQクライアント
  - MQクライアント接続
  - クライアント自動再接続
  - 接続情報の与え方
  - MQSERVER環境変数
  - チャネル定義テーブル
  - プログラム内に記述
  - チャネル数の制御
  - 非同期メッセージ送信
  - メッセージ先読み
- その他チャネル
  - MQ Telemetryサービス、MQTTチャネル
  - AMQPチャネル

チャネル接続

MQシステムの構成

■ サーバー環境 / クライアント環境

◆ MQは2種類のコードを提供

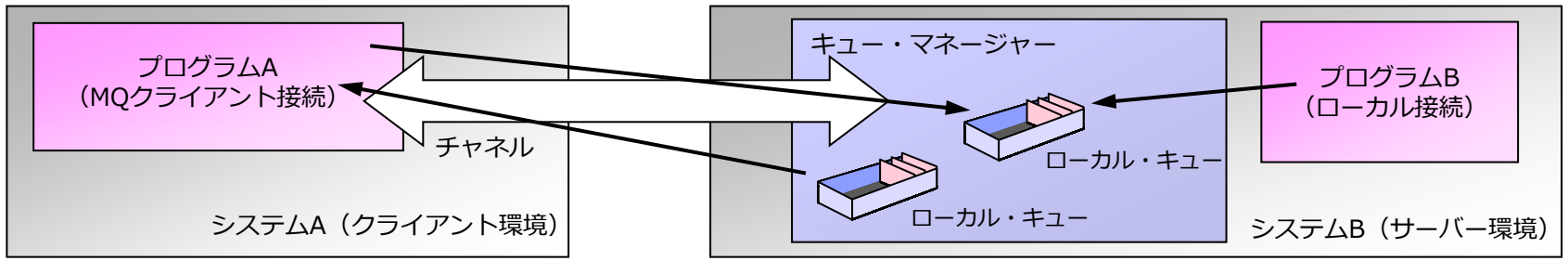
- サーバー・フィーチャー、クライアント・フィーチャー

◆ サーバー環境

- サーバー・フィーチャーを導入した環境
- キュー・マネージャーを作成し、稼働させることが可能

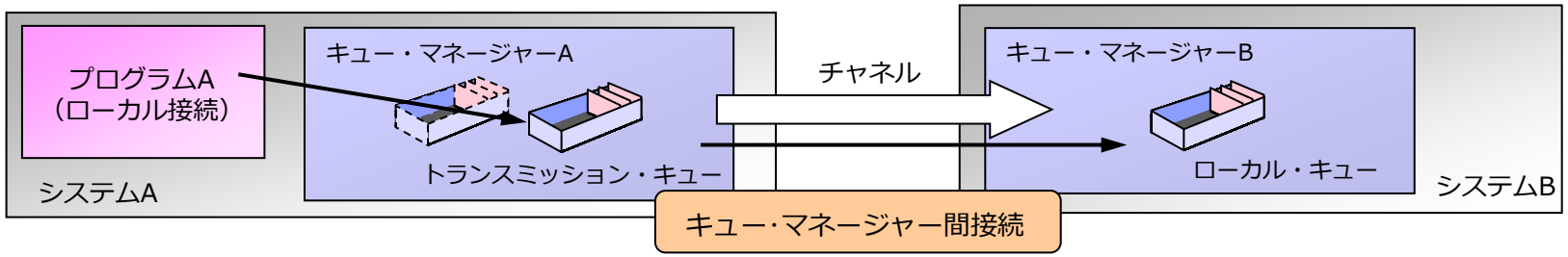
◆ クライアント環境

- クライアント・フィーチャーを導入した環境
- リモートのキュー・マネージャーに接続してキューを読み書きするアプリケーションを稼働させることが可能
  - MQクライアント接続
- キュー・マネージャーを作成することはできない
  - 前提H/Wスペックの緩和、システム管理 / 運用コストが小さい

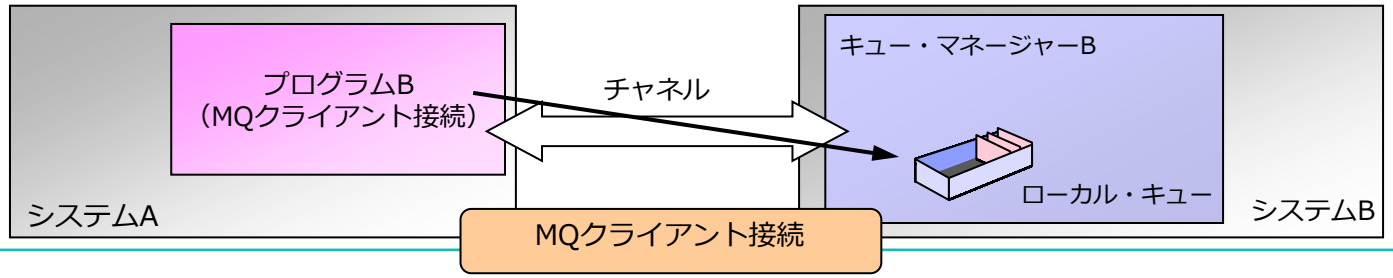


リモート・システムとの通信

- キュー・マネージャー間接続
  - アプリケーションは同一マシンのキュー・マネージャーとローカル接続してキューにメッセージを読み書きする
  - チャネルがキュー・マネージャー間のメッセージの転送を保証
  - 非同期通信
    - リモートのキュー・マネージャーが稼働していなくても、アプリケーションはメッセージの送信が可能



- MQクライアント接続
  - アプリケーションはリモートのキュー・マネージャーとMQクライアント接続してキューにメッセージを読み書きする
  - 同期通信
    - リモートのキュー・マネージャーが稼働していないと、アプリケーションはメッセージを送信できない



キュー・マネージャー間接続

キュー・マネージャー間接続の構成要素

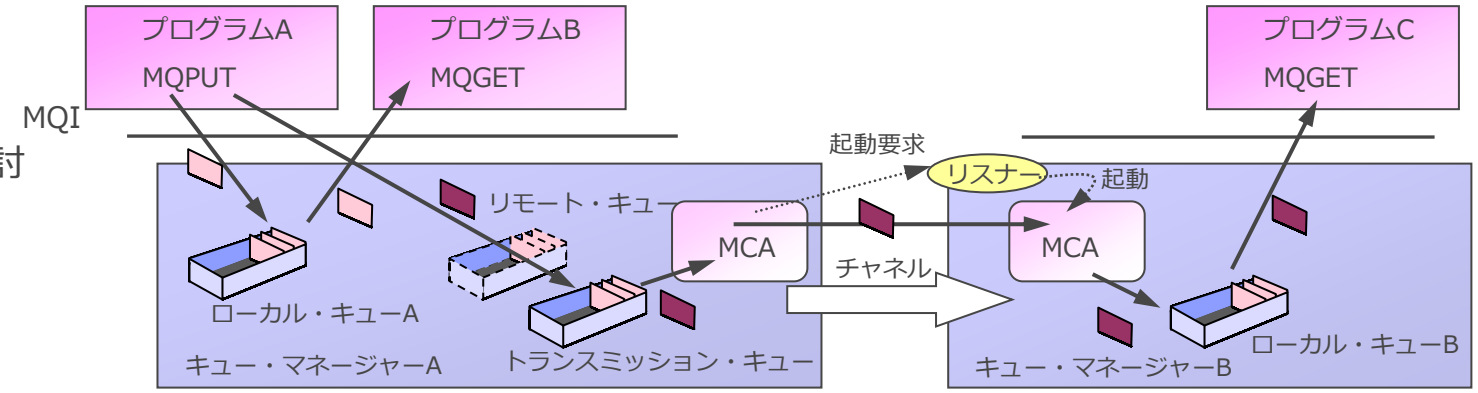
構成要素

◆ キュー・マネージャー

- キュー、チャネルなどの管理
- 基本的に１筐体毎に１つ作成
  - パフォーマンス / 運用要件により追加を検討

◆ キュー

- メッセージを保持する論理的な箱
- １キュー･マネージャーに複数作成



◆ メッセージ・チャネル

- キュー・マネージャー間のコミュニケーション・リンク
- 両端でMCA（メッセージ・チャネル・エージェント）が稼働
  - MCAはキューにメッセージを読み書きする通信プログラム
- キュー･マネージャー間上り下りで１対

◆ リスナー

- リモートのキュー・マネージャーやMQクライアント・プログラムからのチャネル接続要求を受け付けるためのプログラム
- 送信チャネルからの接続要求を受けて受信チャネルを起動

キュー

キューの種類

◆ ローカル・キュー

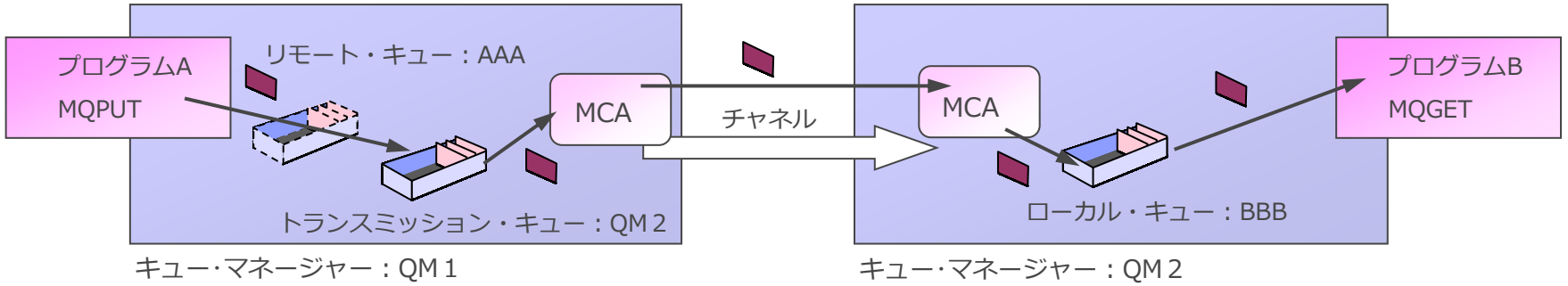
- 実体があり、メッセージを保持することができるキュー

◆ トランスミッション・キュー

- 宛先キュー・マネージャーへの転送メッセージをローカルで一時的に保持する転送用のキュー
  - 特殊な属性をもつローカル・キュー
- 宛先キュー・マネージャー毎に作成

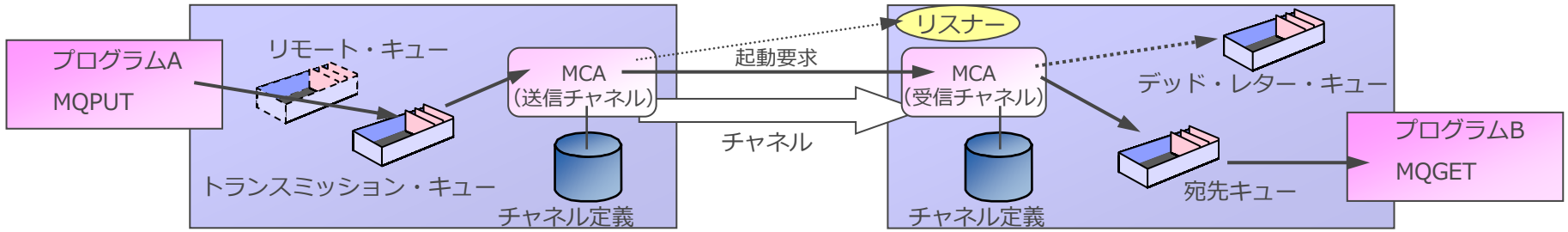
◆ リモート・キュー

- リモートのキュー・マネージャーのキューを指し示す定義
  - 宛先キュー・マネージャー名、宛先キュー名、トランスミッション・キュー名の情報をもつ
- メッセージを書き込むと、トランスミッション・キューにメッセージが書き込まれる



メッセージ・チャネル

- キュー・マネージャー間のコミュニケーション・リンク
  - ◆ 送信側、受信側のそれぞれでチャネル定義を作成（定義名を同じにすること）
  - ◆ 単一方向のメッセージ転送
    - キュー・マネージャー間に双方向のメッセージ転送があれば、2対のチャネル定義が必要
- 受信側ではリスナーを起動しておく必要がある
  - ◆ リスナーは送信チャネルから受信チャネルの起動要求を受け取って、受信チャネルを起動
    - 過去のバージョンと互換性を維持するために、inetdも利用可能 （UNIXの場合）
- チャネルの両端ではMCA(メッセージ・チャネル・エージェント)が稼動
  - ◆ MCAはメッセージをキューに読み書きする通信プログラム
    - 送信側のMCAはトランスミッション・キューからメッセージを読み取り、ネットワークに送信
    - 受信側のMCAは受信したメッセージを宛先キューに書き込む
      - 宛先キューにメッセージが書き込めないとデッド・レター・キューに書き込む
      - デッド・レター・キューにも書き込めないとメッセージを破棄、または、チャネル異常終了



メッセージ・チャネル

チャネルの起動と停止

- チャネルが起動している時のみメッセージ送信が行われる
  - アプリケーションはチャネルの起動 / 停止に関わらず、メッセージの書き込みが可能
- コマンドにより操作
  - START CHANNEL / STOP CHANNEL
- チャネルの自動起動 / 自動停止
  - トランスミッション・キューにメッセージが入るとチャネルを起動（詳細はp20「チャネルのトリガリング」を参照)
  - 指定時間トランスミッション・キューにメッセージがなければ自動的に停止

メッセージ・チャネルのタイプ

- SENDERチャネル（送信側チャネル）
  - メッセージの送信を行い、チャネルの起動、停止が可能
- SERVERチャネル（サーバーチャネル）
  - メッセージの送信を行い、チャネルの起動（フル定義時）、停止が可能
- RECEIVERチャネル（受信側チャネル）
  - メッセージの受信を行い、チャネルの停止が可能（※）
- REQUESTERチャネル（要求側チャネル）
  - メッセージの受信を行い、チャネルの起動、停止が可能（※）

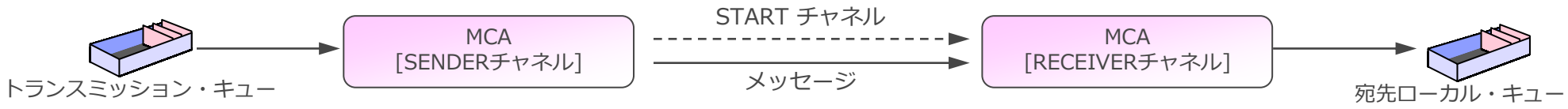
※ 自分だけ停止。送信側がSENDER、フル定義のSERVERの場合、送信側は接続再試行状態になる

<参考> メッセージ・チャネル

メッセージ・チャネルのタイプと組み合わせ

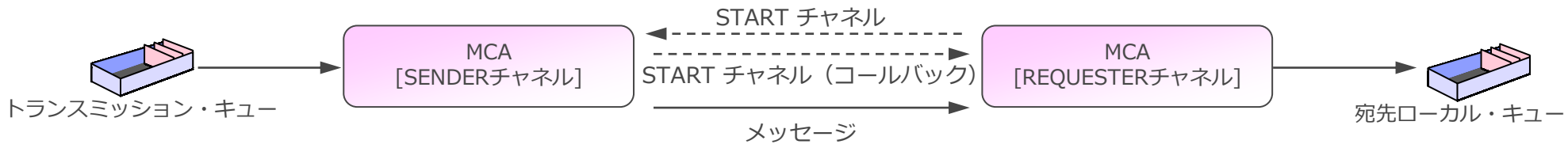
◆ SENDER – RECEIVER

- 最も一般的な組み合わせ

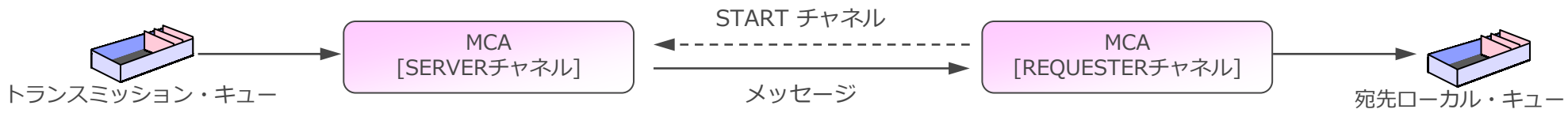


◆ SENDER – REQUESTER

- 両方向からの起動が可能
- ただし、前回停止したチャネルから起動する必要があり、運用上の考慮が必要



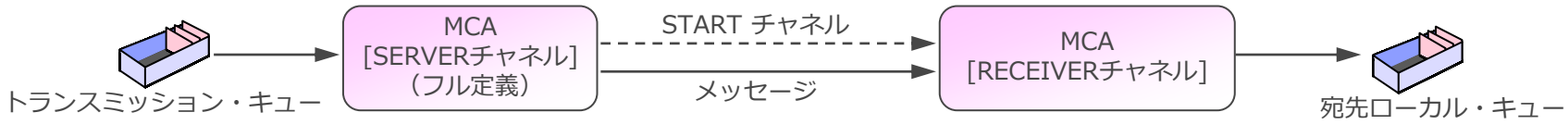
◆ SERVER – REQUESTER



<参考> メッセージ・チャネル

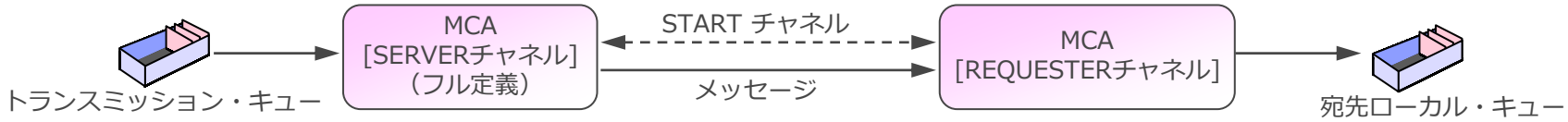
■ メッセージ・チャネルのタイプと組み合わせ（続き）

◆ SERVER（フル定義） – RECEIVER



◆ SERVER（フル定義） – REQUESTER

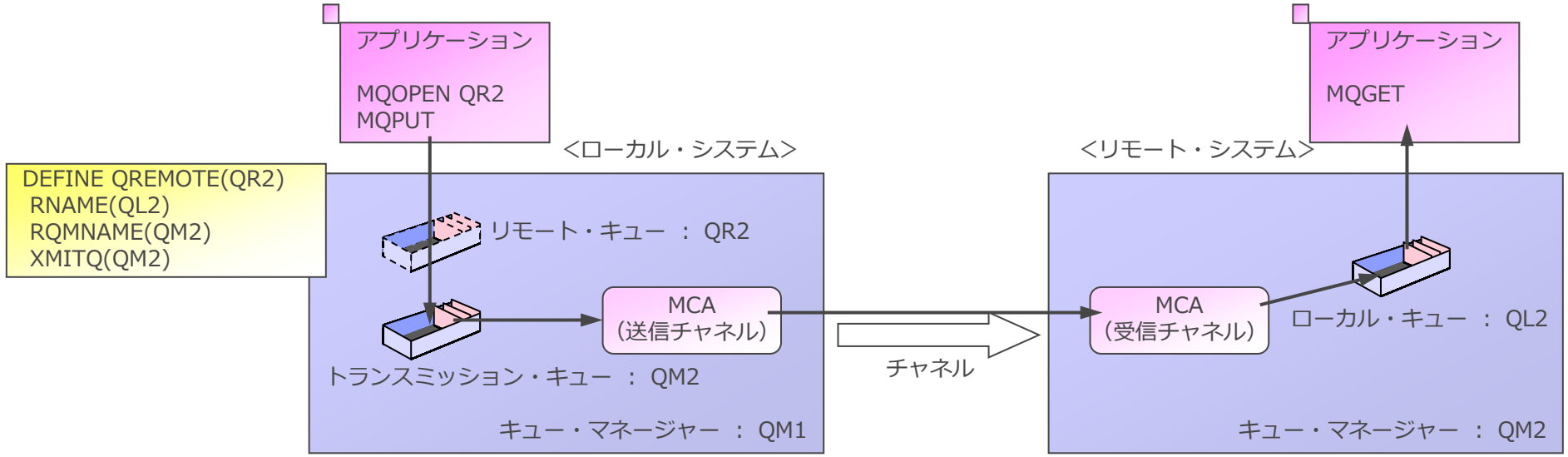
- 両方からの起動が可能
- ただし、前回停止したチャネルから起動する必要があり、運用上の考慮が必要



※ フル定義のSERVERチャネル定義とは、チャネルの開始可能に十分な属性定義（コネクション名）を定義したもの

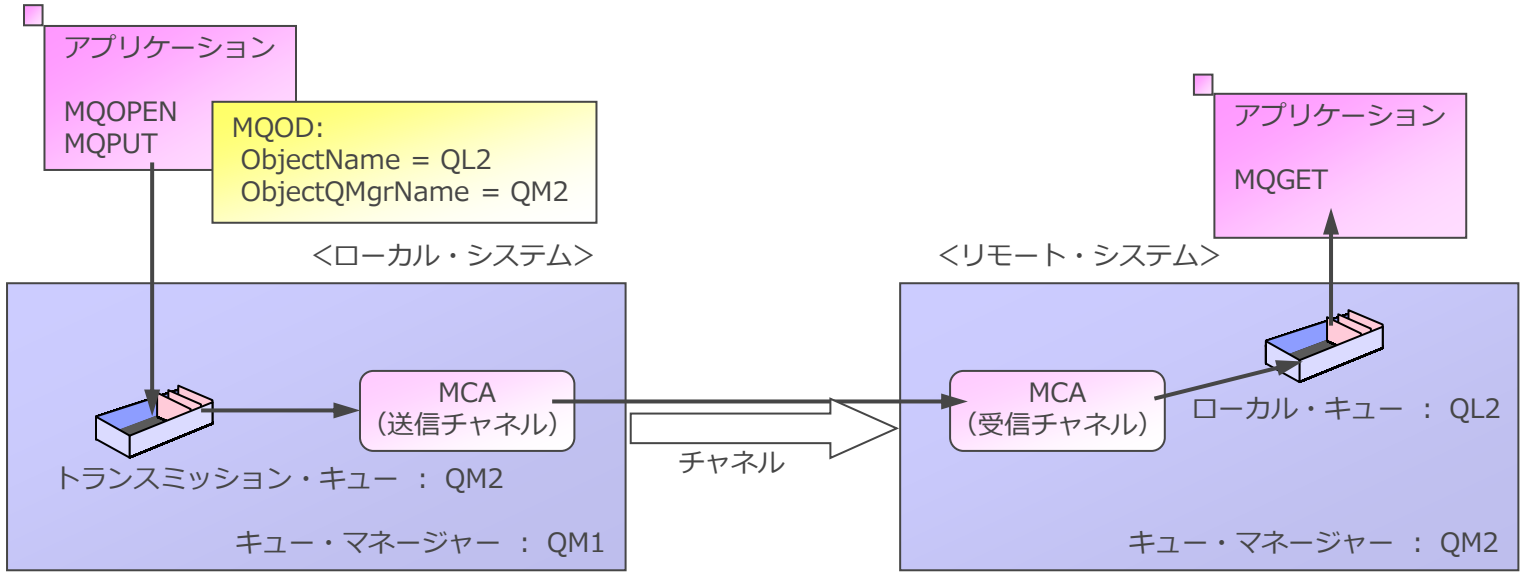
リモートのキューへの送信

- リモート・キューを経由した送信
  - ◆ MQOPEN(MQPUT1)時にMQODにローカル・システムのリモート・キュー名を指定
    - ローカル・キューマネージャー上にリモート・キューを定義
    - アプリケーションは宛先のキュー・マネージャー名とローカル・キュー名を意識しない
    - 定義のみで宛先キューの変更が可能　⇒　柔軟性が高い
  - ◆ アプリケーションは宛先のキュー・マネージャーやキューを意識する必要はない
    - 宛先変更に対する柔軟性が高い



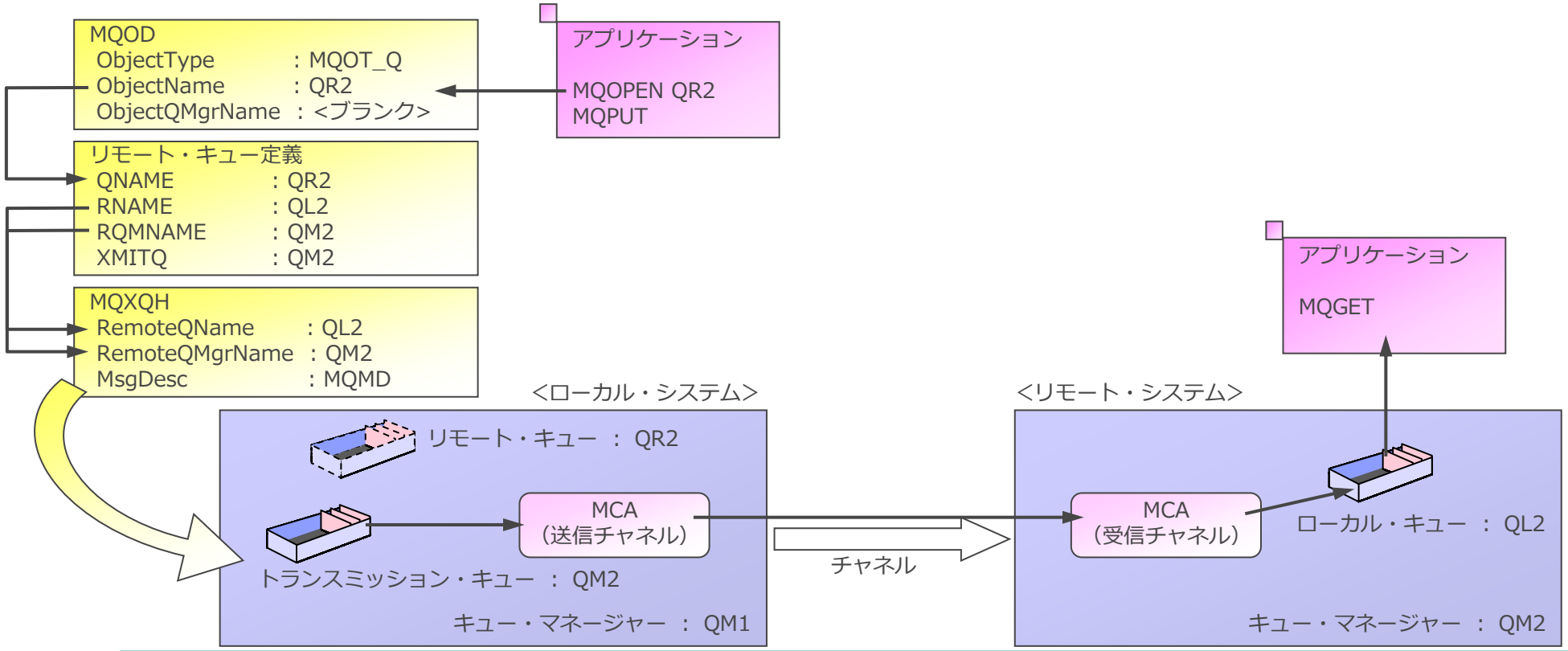
リモートのキューへの送信

- 宛先のキュー・マネージャー / キュー名を直接指定した送信
  - ◆ MQOPEN(MQPUT1)時に、MQODに宛先システムの情報を指定
    - ObjectQMgrNameフィールドに宛先システムのキュー・マネージャー名を指定
    - ObjectNameフィールドに宛先システムのキュー名を指定
  - ◆ ローカル・キュー・マネージャー上にリモート・キューの定義は不要
  - ◆ 宛先のキュー・マネージャー名と同一名のトランスミッション・キュー定義が必要
  - ◆ アプリケーションは宛先のキュー・マネージャー名とキュー名を意識



リモート・キューの定義

- リモート・キュー定義により、アプリケーションは宛先のキュー・マネージャーやキューを意識する必要はない
  - ◆ 宛先変更に対する柔軟性が高い
- リモート・キューに対する別名定義も可能
  - ◆ アプリケーション単位の別名定義により、アプリケーション単位でMQPUTの抑制ができる



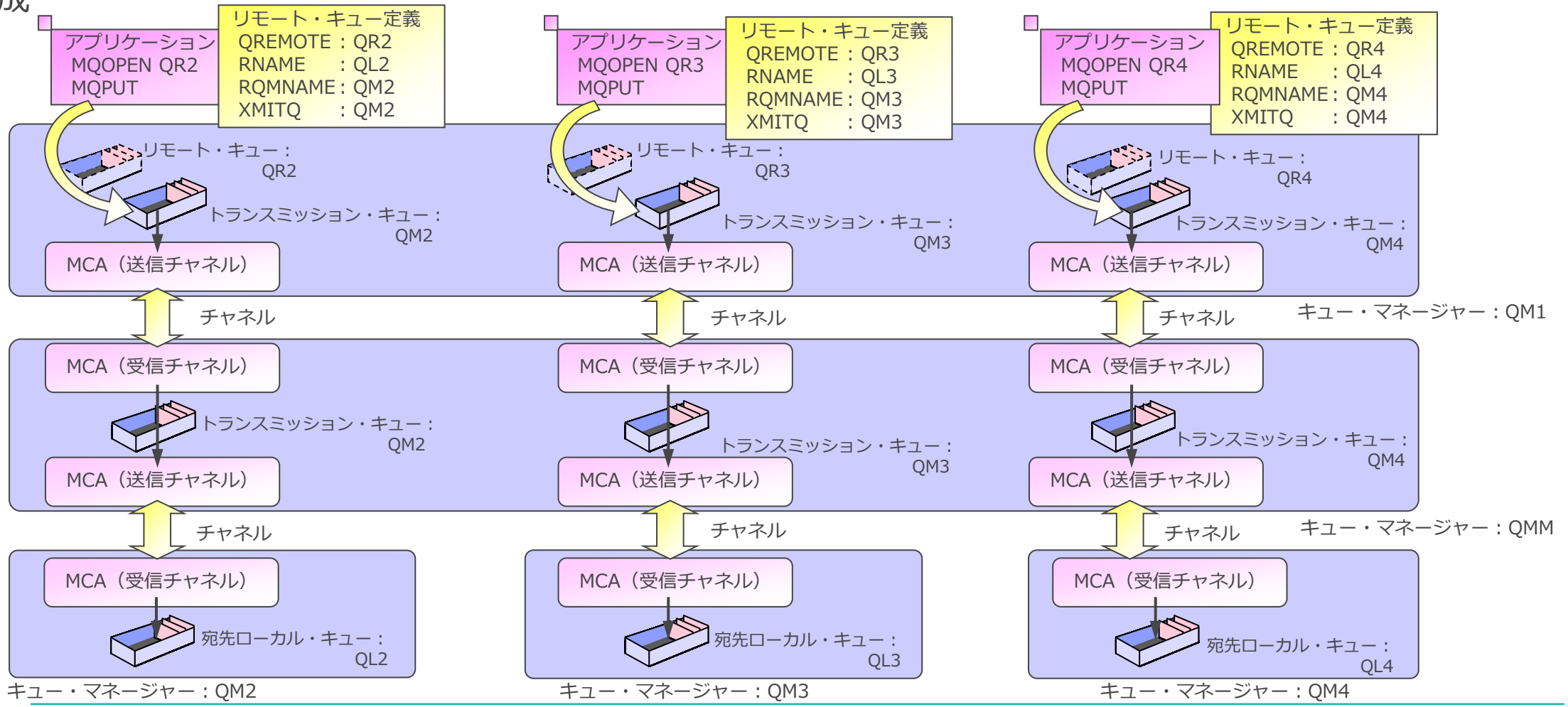
<参考> マルチ・ホップ構成

- 送信キュー・マネージャーと受信キュー・マネージャーの間にある、中間キュー・マネージャーを介して宛先へメッセージを伝送する構成

- マルチ・ホップ構成の際にはキュー・マネージャー別名定義を使用
  - ◆ キュー・マネージャー別名定義はリモート・キュー定義の特殊な指定方法
    - RNAME属性がブランクのリモート・キュー定義を、キュー・マネージャー別名定義と呼ぶ
  - ◆ キュー・マネージャー別名定義を使用することで以下が可能
    - メッセージ送信時のキュー・マネージャー名を再マップし、別のキュー・マネージャーに変更
    - メッセージ送信時に経路上の伝送キューを指定または変更
    - メッセージの受信時に、メッセージの宛先がローカル・キュー・マネージャーを指しているかどうかを判定し、適切なキューにルーティング

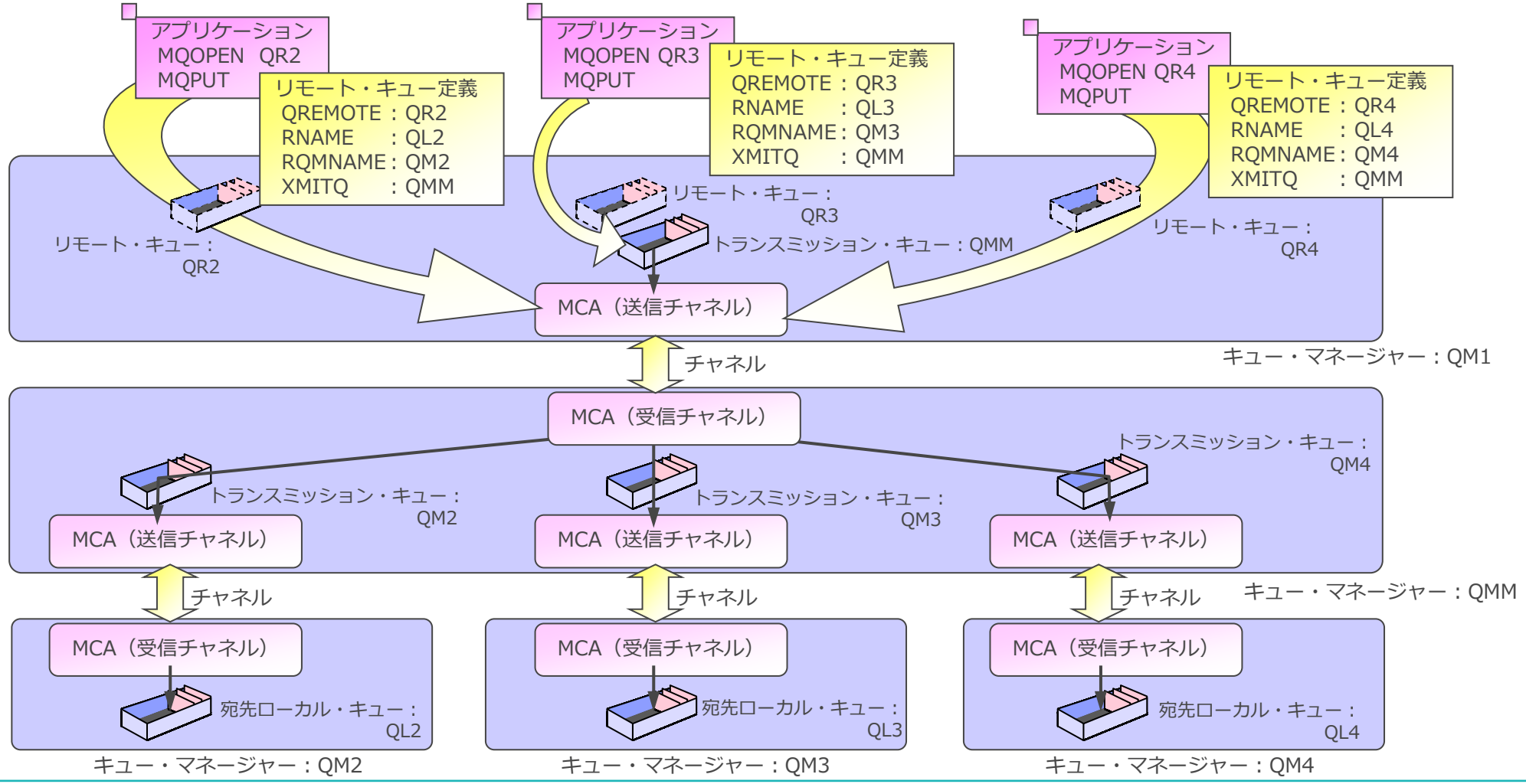
<参考> マルチ・ホップ（ステージング）　1対1方式

- キューの定義によって宛先のキュー・マネージャーへの転送も可能
  - 宛先のキュー・マネージャー名と同一のトランスミッション・キューを最終宛先のキュー・マネージャー以外に作成



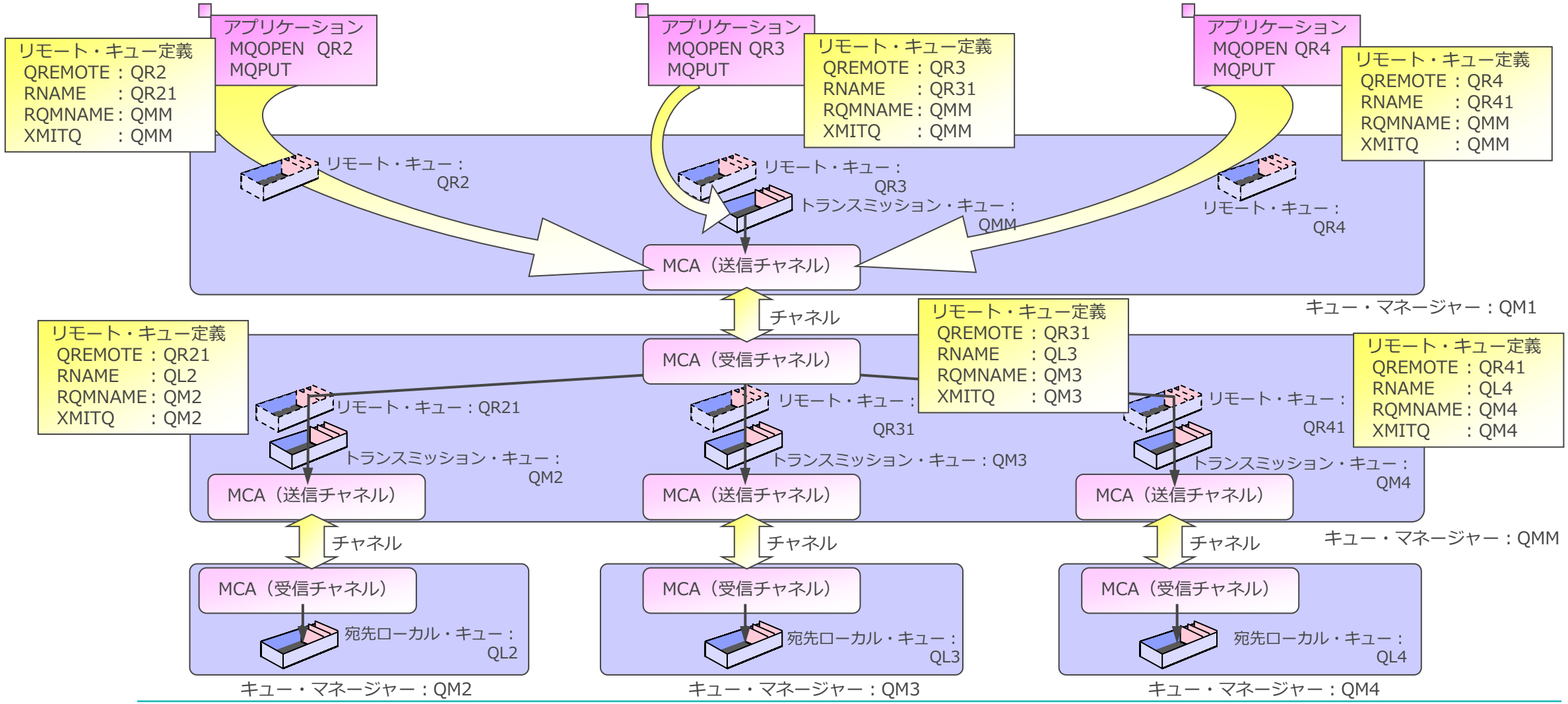
<参考> マルチ・ホップ（ステージング）　ハブ方式

- 中間のキュー・マネージャー（ハブ）を経由した宛先キュー・マネージャーへの転送も可能
  - ◆ チャネル・パスの最適化、資源管理の最適化



<参考> マルチ・ホップ（ステージング）　ハブ応用方式

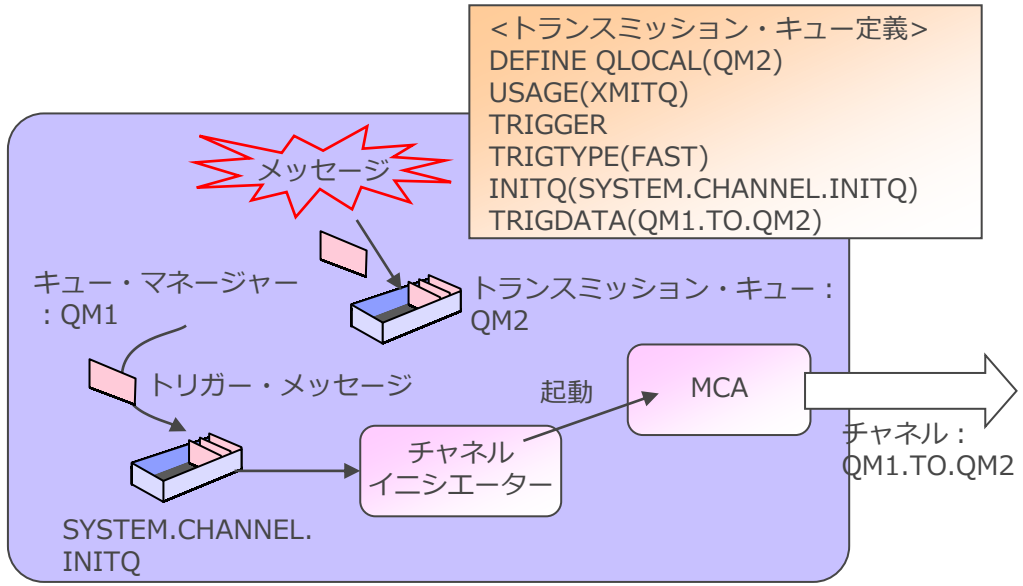
■ 中間のキュー・マネージャー（ハブ）での集中キュー管理



チャネルのトリガリング

- トリガリングの機能によりチャネルを自動的にスタートさせることが可能
  - トランスミッション・キューにトリガリングの設定をすることにより、転送用のメッセージが書き込まれると自動的に対応するチャネルが開始

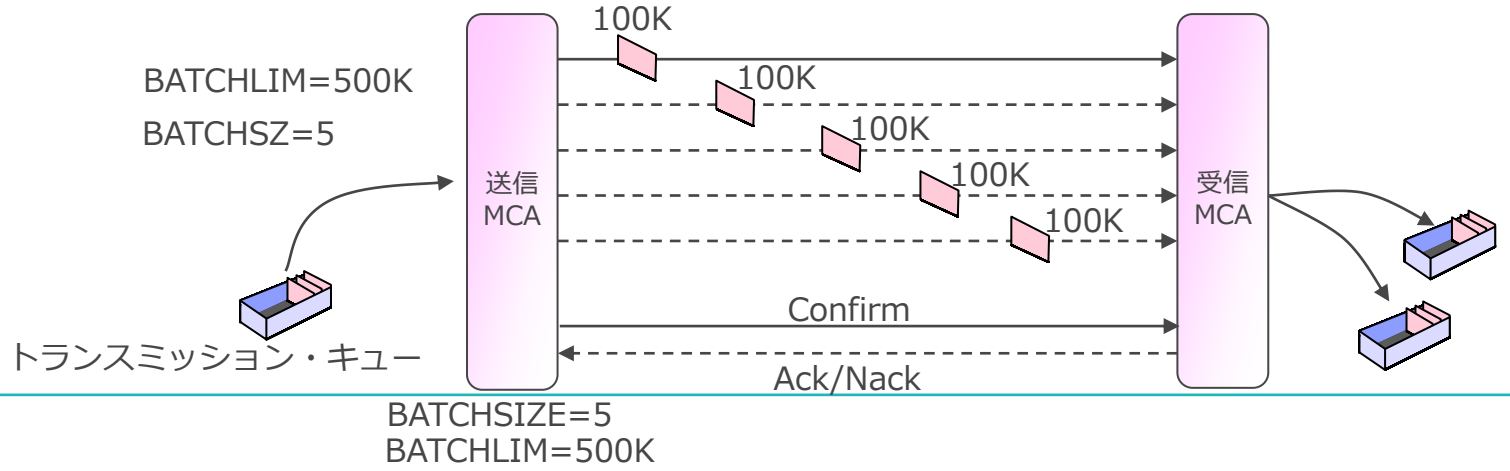
- チャネルのトリガリングに必要な設定
  - トランスミッション・キューに以下の属性を指定
    - TRIGGER
    - TRIGTYPE(FIRST)
    - INITQ(SYSTEM.CHANNEL.INITQ)
      - イニシエーション・キューは、SYSTEM.CHANNEL.INITQ で固定
    - TRIGDATA(チャネル名)



- チャネルの停止に注意が必要
  - STOPPEDのステータスのチャネルにはトリガーは発生しない
  - 切断間隔の満了やSTOP CHANNELのオプション指定でINACTIVEのステータスで停止させる

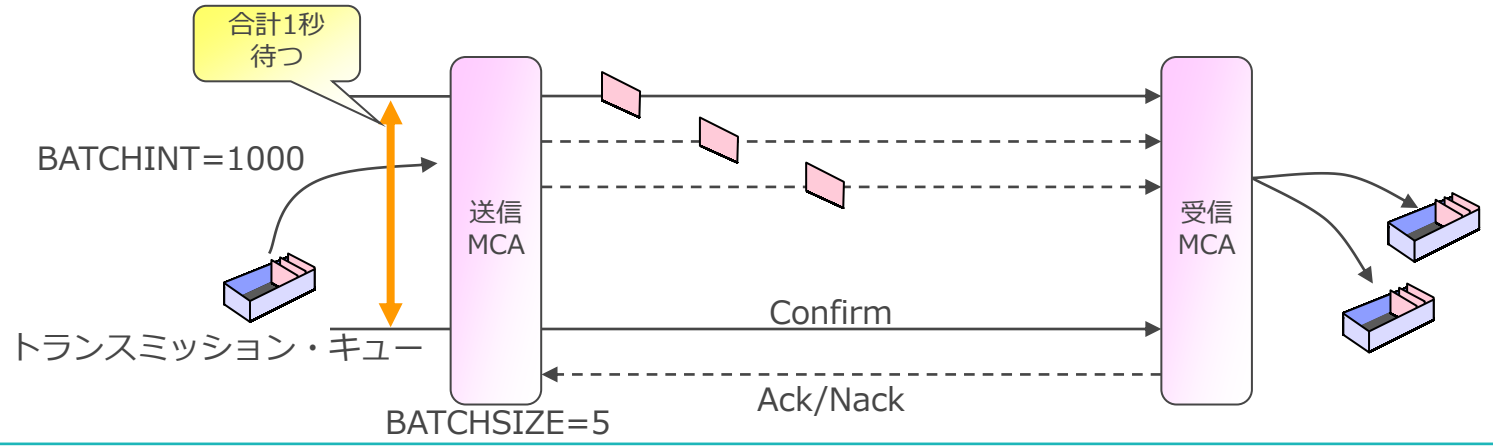
転送メッセージのバッチ化

- チャネルは、複数のメッセージを1つのUOWとして転送
  - 送受信MCA間で1UOW毎に同期がとられる
    - 同期がとられるまでは受信側ではメッセージは取り出せない
      （FASTチャネル使用時のノン・パーシステント・メッセージを除く）
    - バッチをコミットする前に受信側がActiveか確認することも可能（バッチ・ハートビート）
- チャネルのBATCHSZ属性やBATCHLIM属性で指定
  - BATCHSZ属性で幾つのメッセージを1UOWとして転送するかを指定
  - BATCHLIM属性で何キロバイトを1UOWとして転送するかを指定
  - バッチ・サイズやバッチ・リミットが大きいほど、メッセージ転送のスループットは高い
  - トランスミッション・キューにバッチ・サイズ分のメッセージがない、またはバッチ・リミット分のバイト数に達していない場合は、バッチ・インターバルに指定された時間分待機し、そこまでを1UOWとして転送
  - BATCHSZ属性とBATCHLIM属性を両方設定した場合、どちらか片方の指定数を満たした時点で1UOWとして転送される



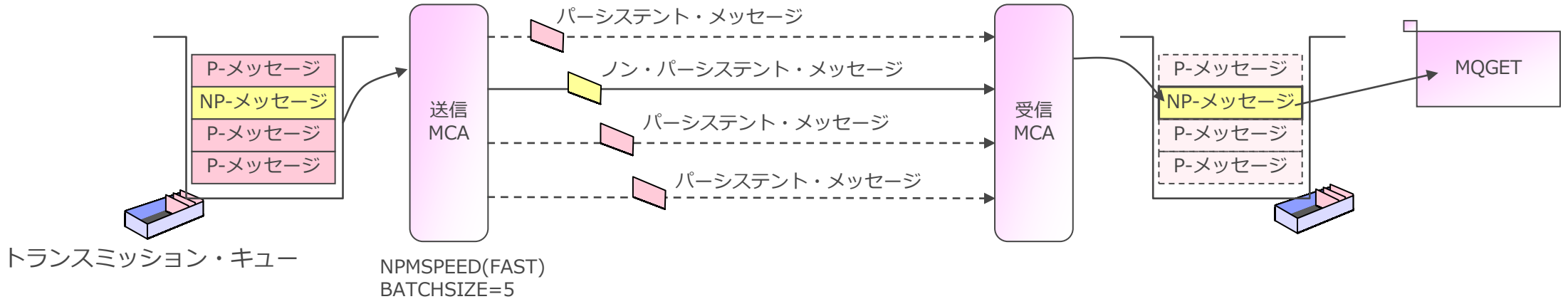
チャネルのバッチ・インターバル

- バッチ・サイズに到達しないメッセージの同期を、指定した間隔待つ機能
  - 小さなバッチ毎の同期フローを減らすことによるメッセージのスループット向上
  - メッセージが断続的にくる場合に、ネットワーク・コストを削減
  - 受信側がメッセージを受け取る時間は通常は遅くなる
  - FASTチャネル上のノン・パーシステント・メッセージには影響しない（バッチ・サイズには数えられる）
- チャネルのBATCHINT属性で指定
  - SDR, SVR, CLUSSDR, CLUSRCVRチャネルのみ指定可能
  - 最初のメッセージが来てからメッセージを待っている合計時間
    - メッセージをトランスミッション・キューから取り出して転送する時間は含まない
  - バッチ・インターバルが0であれば、最後のメッセージ送信後すぐに同期処理



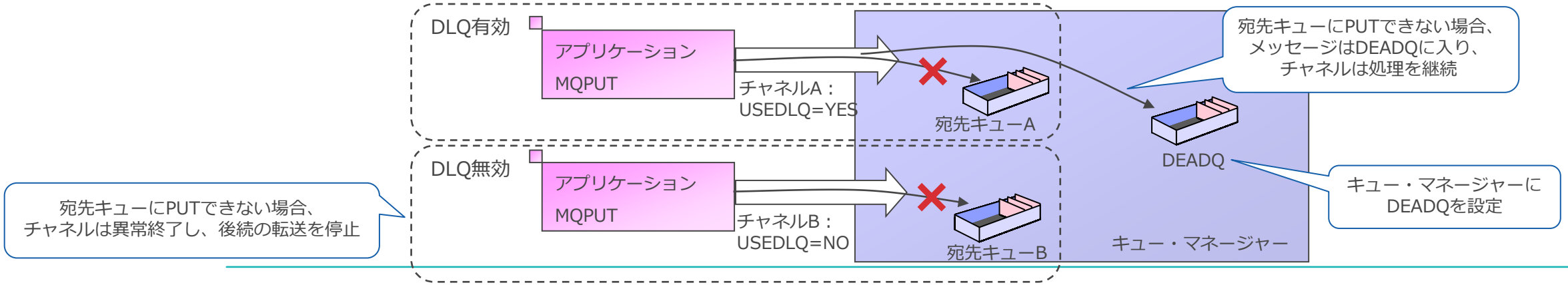
FASTチャネル

- ノン・パーシステント・メッセージをチャネル間の同期処理を行わずに転送する機能
  - ◆ ノン・パーシステント・メッセージは、より速く宛先のキューに配布される
  - ◆ ノン・パーシステント・メッセージもチャネルのバッチ数にはカウントされる
  - ◆ パーシステント・メッセージの転送は従来通りバッチ・サイズ毎に同期をとって配布される
- チャネルのNPMSPEED属性で設定
- FASTチャネル障害時、転送途中のノン・パーシステント・メッセージはロストする



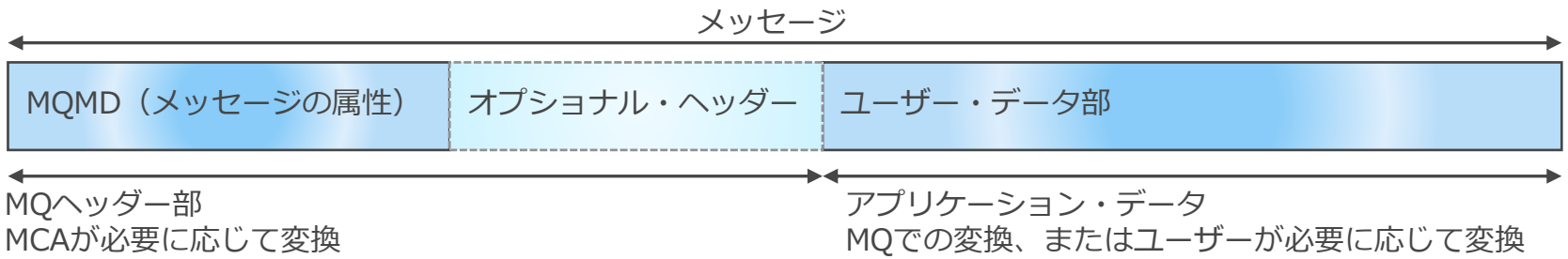
デッド・レター・キュー

- チャネル/トピックでデッド・レター・キュー (DLQ) 使用可否設定が可能
  - ◆ キュー・マネージャーのDEADQ属性にDLQを設定した後、各オブジェクトで使用可否を設定
    - USEDLQ属性にて使用可否を設定
      - YES (チャネルデフォルト)： DLQ設定が有効
      - NO：キュー・マネージャーのDEADQ属性にかかわらずDLQ設定が無効
      - ASPARENT (トピックデフォルト)(※) ： トピック・ツリー内の上位トピック・オブジェクトの設定を引き継ぐ
        ※ASPARENTはトピックのみ
    - チャネルではSVRCONN/CLNTCONNを除くすべてのチャネルタイプにUSEDLQ属性が追加
  - ◆ チャネルでDLQ使用可否設定を分ける例
    - 使用するチャネルごとにメッセージの順序保証要件が異なるケース
      - チャネルAでは順序保証が不要、チャネルBは順序保証が必要な場合（パーシステント・メッセージ使用）
      - チャネルAはDLQ有効、チャネルBはDLQ無効に設定
      - 宛先キューがPUT不可の場合、チャネルAはDLQにメッセージを破棄し転送を継続、チャネルBは異常終了



文字コード変換

- メッセージの文字コード変換を行う機能
  - ◆ CCSID（文字）とエンコーディング（数値）に基づいてコード変換を行う
    - CCSIDとエンコーディングはキュー・マネージャーや個々のメッセージで指定
  - ◆ MQヘッダー部（MQMDおよびオプショナル・ヘッダー）の変換
    - MQヘッダー部のコード変換はMCAが自動的に行う
      - チャネル間でお互いのCCSIDとエンコーディングをやりとりし、接続可否を決定
      - チャネル折衝時には両側のキュー・マネージャーのCCSIDとエンコーディング値が使用される
    - コード変換の対象
      - 文字型フィールドはCCSIDに基づいてコード変換
      - 数値フィールドはエンコーディングに基づいて変換
      - バイナリー・フィールド（メッセージIDや相関ID）は変換対象外
  - ◆ ユーザー・データ部の変換
    - 全て文字で構成　　：MQにより変換可能
    - ユーザー定義フォーマット：ユーザーEXITにより変換



文字コード変換

- CCSIDの設定と影響範囲
  - ◆ キュー・マネージャーのCCSID属性に設定
    - ● メッセージ・ヘッダーのコード変換に使用される
      - チャネルでのユーザー・メッセージの変換先CCSIDとしても使用
    - ● 転送メッセージの内容には関係しない
    - ● キュー・マネージャー間の接続（チャネル）およびメッセージ転送可否に影響を与える
  - ◆ メッセージ属性（MQMDのCodedCharsetId）に設定
    - ● ユーザー・データ部の文字型データのCCSIDを示す
      - アプリケーションで設定可能だが、特に指定しなければキュー・マネージャーのCCSIDがセットされる
    - ● ユーザー・データ部のコード変換に使用

- エンコーディングの設定と影響範囲
  - ◆ キュー・マネージャーが稼動するプラットフォームにより決定される
    - ● MQヘッダー内の数値データはMCAが自動的に変換
  - ◆ メッセージ属性（MQMDのEncoding）に設定
    - ● デフォルトではキュー・マネージャーの稼動する環境の値がセットされる
    - ● ユーザーが定義することも可能だが、デフォルトを使用することを推奨

文字コード変換

■ デフォルト・コードページ

◆ 変換をサポートしていないCCSIDのキューマネージャー間で接続を行う方法

- デフォルト指定されたCCSIDを使用してチャネルを接続
- デフォルト・コードページの利用により、DBCSとSBCSのキュー・マネージャー接続も可能

◆ MQクライアント・チャネルでも使用可能

■ デフォルト・コードページの指定

◆ ccsid.tblファイルにデフォルトのコードページを設定(※)
※Linux,Windowsでは、ccsid_part2.tblファイルに設定

- DEFAULT DATA CONVERSIONのパートにてデフォルトEBCDIC CCSIDとデフォルトASCII CCSIDを指定

```
                                        :
# DEFAULT DATA CONVERSION
                                        :
#
# Default conversions are enabled by creating two lines similar to the
# two following, but removing the # character which indicates a comment.
#default      0     500    1    1    0          ← コメントを外す
#default      0     850    1    2    0          ← コメントを外す
#
# The first line sets the default for EBCDIC CCSIDs to 500
# and the second sets the default for ASCII and similar CCSIDs to 850.    :
```

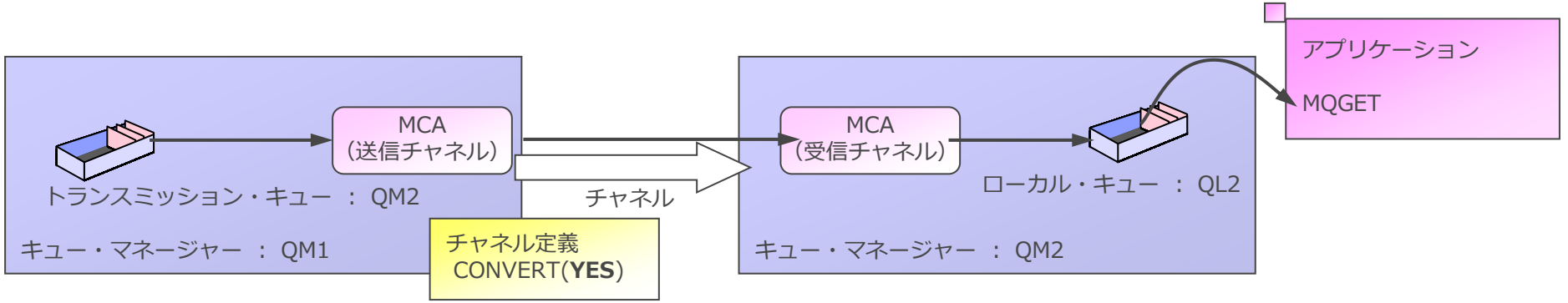
ccsid_part2.tblの配置先（デフォルト）
Linux　　: /var/mqm/conv/table
Windows : <data_dir>¥conv¥table　※<data_dir>のデフォルトは、 C:¥ProgramData¥IBM¥MQ

文字コード変換

■ ユーザー・データの変換場所

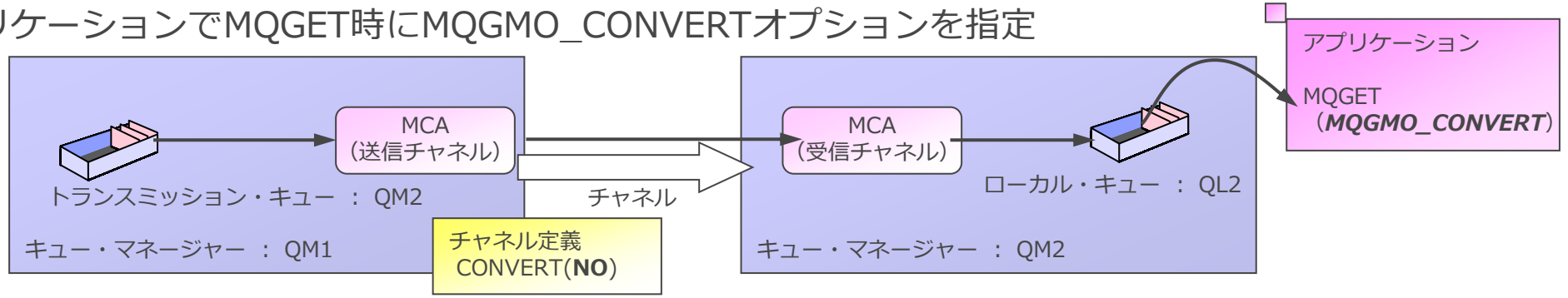
◆ 送信側でデータの変換を実施する場合

- 送信側でチャネルの属性をCONVERT(YES)に設定
- 基本的にはMQGET時に変換させるようにする。変換が送信側でしかサポートされないような場合はチャネルを使用する



◆ 受信側でデータの変換を実施する場合

- 受信側のアプリケーションでMQGET時にMQGMO_CONVERTオプションを指定



チャネルEXIT

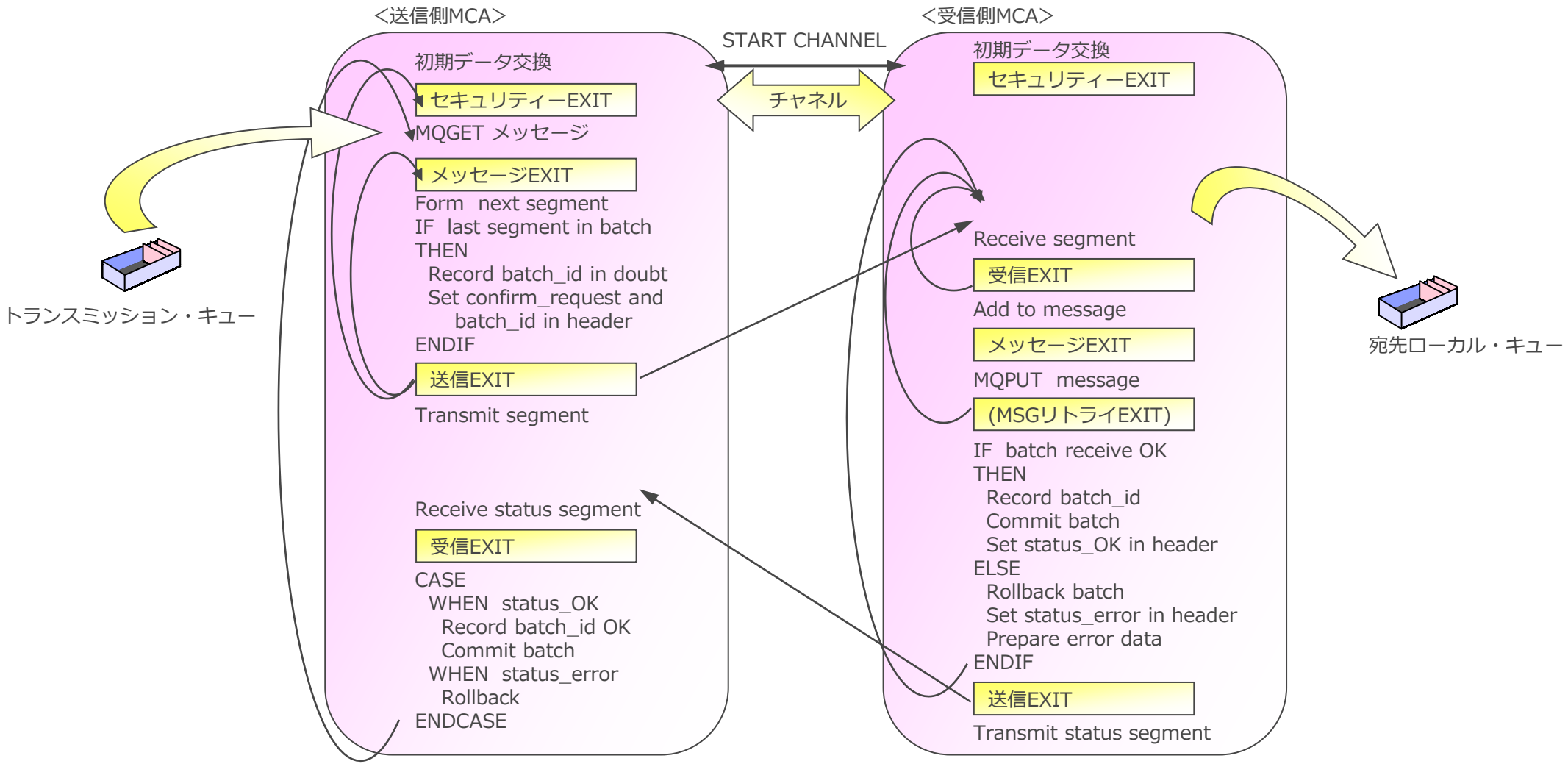
- チャネル接続におけるユーザーEXIT
  - ◆ MCAのチャネルEXIT
    - MCAの処理サイクル中にスケジュール
      - セキュリティーEXIT
      - 送信EXIT
      - 受信EXIT
      - メッセージEXIT
      - メッセージ・リトライEXIT
      - 自動定義EXIT
- EXITの設定
  - ◆ チャネルの定義で指定
    - セキュリティーEXIT　　：SCYEXIT
    - 送信EXIT　　：SENDEXIT
    - 受信EXIT　　：RCVEXIT
    - メッセージEXIT　　：MSGEXIT
    - メッセージ・リトライEXIT：MREXIT
  - ◆ キュー・マネージャーの定義で指定
    - 自動定義EXIT　　：CHADEXIT

チャネルEXIT

EXITタイプ	スケジュール・ポイント	主な用途
セキュリティー	チャネル開始時の初期データ交換後	EXIT内で定義した形式でのセキュリティー・メッセージの交換の起動か可能 （両側のEXITが許可した場合にのみ後続の転送処理が可能）
メッセージ	［送信側］ メッセージをXmitキューから取り出した直後で必要に応じてセグメント化する直前 ［受信側］ メッセージを宛先のキューに書き出す直前	MQXQH（MQMD、ユーザー・メッセージを含む）内のフィールドの変更が可能 －暗号化/複合化 －入力ユーザーIDの検証 －ユーザー・メッセージ部のコード変換 －メッセージ部の宛先の変換 －MQMDのReply-to情報の変更 －MQMDのコンテキスト情報の変更 －ユーザー・ジャーナリング
送信	1つのセグメントを送信する直前（両側） （ユーザー・メッセージ以外（ステータス・データなど）のセグメント・データに対してもスケジュールされるが初期データの交換、およびセキュリティー・チェック・フェーズではスケジュールされない）	各転送データの先頭8バイトを除いては全ての転送情報の変更が可能 －暗号化/複合化 －データの圧縮/拡張 －ユーザー・ジャーナリング
受信	1つのセグメントを受信した直後（両側） （ユーザー・メッセージ以外（ステータス・データなど）のセグメント・データに対してもスケジュールされるが初期データの交換、およびセキュリティー・チェック・フェーズではスケジュールされない）	
メッセージ・リトライ	宛先キューへの書き込みが失敗した直後	MQXQH（MQMD、ユーザー・メッセージを含む）内のフィールドの変更が可能 －自動的なリトライ機能の提供

<参考> チャネルEXIT

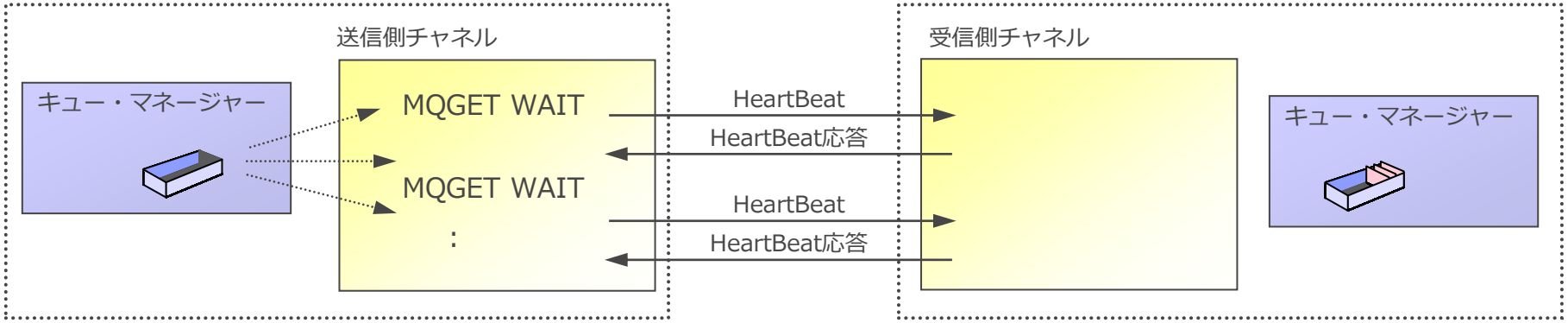
■ チャネルEXITのスケジュールとタイミング



チャネルのハートビート

- チャネル間でメッセージが転送されていない時にハートビートを行う機能
  - ◆ 送信側チャネルはハートビートを送信、受信側チャネルはハートビート応答を折り返す
  - ◆ 送信側チャネルがネットワークや受信側チャネルの障害をより早く検知することができる
  - ◆ 受信側チャネルがハートビートにより、MQリソース（オープンしているキューなど）を開放できる

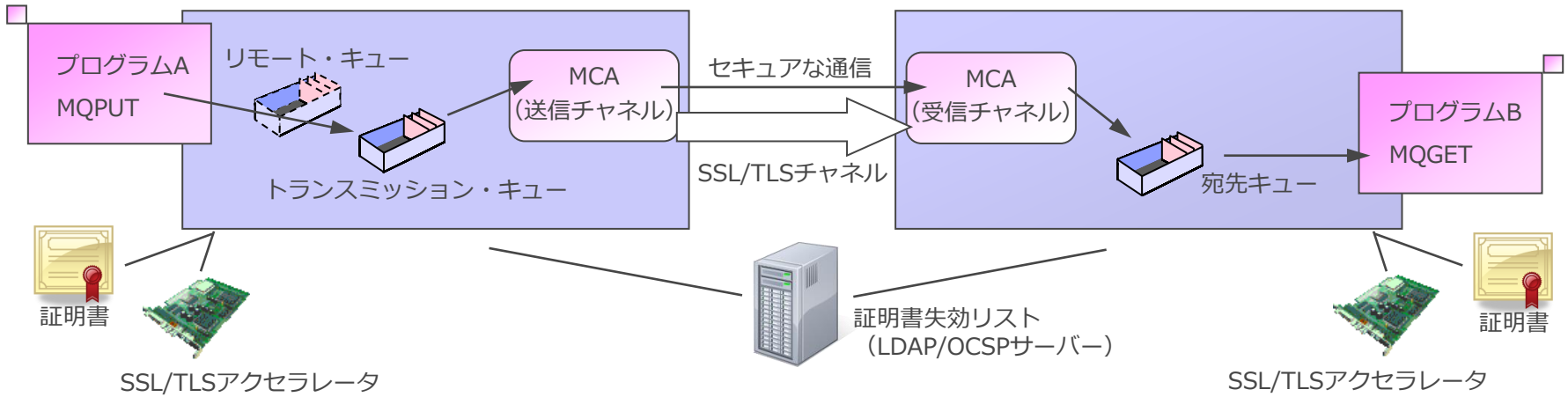
- チャネルのHBINT属性で設定
  - ◆ 全てのチャネル・タイプで設定可能
  - ◆ 送信側と受信側の設定が異なる場合は、チャネル間の折衝で長い方の間隔がとられる



メッセージの暗号化

- チャネル接続でSSL/TLSプロトコルをサポート
  - ◆ クライアント認証も選択可能
  - ◆ 全てのチャネル・タイプで使用することができる
    - メッセージ・チャネル、MQIチャネル、クラスター・チャネル
  - ◆ CipherSpecの選択が可能
  - ◆ 証明書の識別名（DN）によるフィルタリングが可能
    - 証明書のDNをもとに接続の許可 / 拒否を制御することができる
  - ◆ 証明書失効リスト（CRL）への照会
    - LDAPサーバーやOCSPサーバーへのアクセス設定
  - ◆ SSL/TLSアクセラレータのサポート

TLS 1.3はV9.2からサポート
SSL 3.0は非推奨



チャネルのメッセージ圧縮

- チャネル間でメッセージを圧縮して転送
  - ネットワーク資源を効率的に利用
  - 簡易なメッセージ暗号化の手段としても利用可能
  - MQクライアント接続でも利用可能
- チャネル毎にヘッダー部とメッセージ部に分けて、圧縮有無 / 圧縮方法を指定可能
  - チャネルのCOMPHDR、COMPMSG属性で設定

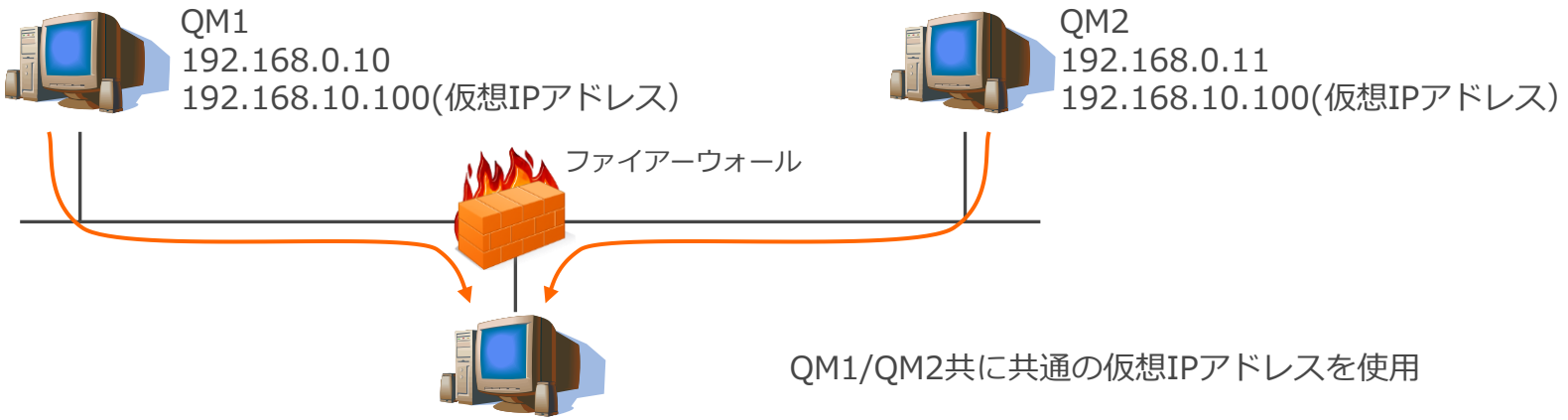
ヘッダー部で指定可能なアルゴリズム
NONE　：なし
SYSTEM：XOR+RLE

メッセージ部で指定可能なアルゴリズム
NONE　：なし
RLE　：同じ文字が連続する場合に有効
ZLIBFAST/ZLIBHIGH：ZLIB圧縮関数を利用
ANY　：送信側に従う

- 圧縮方法の決定
  - チャネル開始時に折衝を行い、同じ圧縮方法が設定されている時のみ圧縮を行う
  - 必ず、送信側チャネルの指定に従う場合は、受信側チャネルをANYに設定する

ローカル・アドレスの指定

- アウトバウンドの通信に使用するIPアドレス、ポートを固定することが可能
  - トランスポート・タイプがTCPの以下のチャネルでのみ指定可能
    - SDR、SVR、 RQSTR、CLNTCONN、CLUSSDR、CLUSRCVR (※)
      ※クラスター受信チャネルの指定は自動定義される全てのクラスター送信チャネルに適用されるので、指定する際は注意が必要
  - チャネルのLOCLADDR属性で指定
    - IPアドレスおよびポート（ポート範囲）で指定
  - ファイアウォールを介して通信を行っている場合などで有効
    - ローカル・アドレスを設定できるため、仮想IPアドレスを指定してファイアー・ウォールを通過させることが可能
    - ファイアー・ウォール上での管理が減少
      - 複数キュー・マネージャー上で共通の仮想IPアドレスを使用することが可能

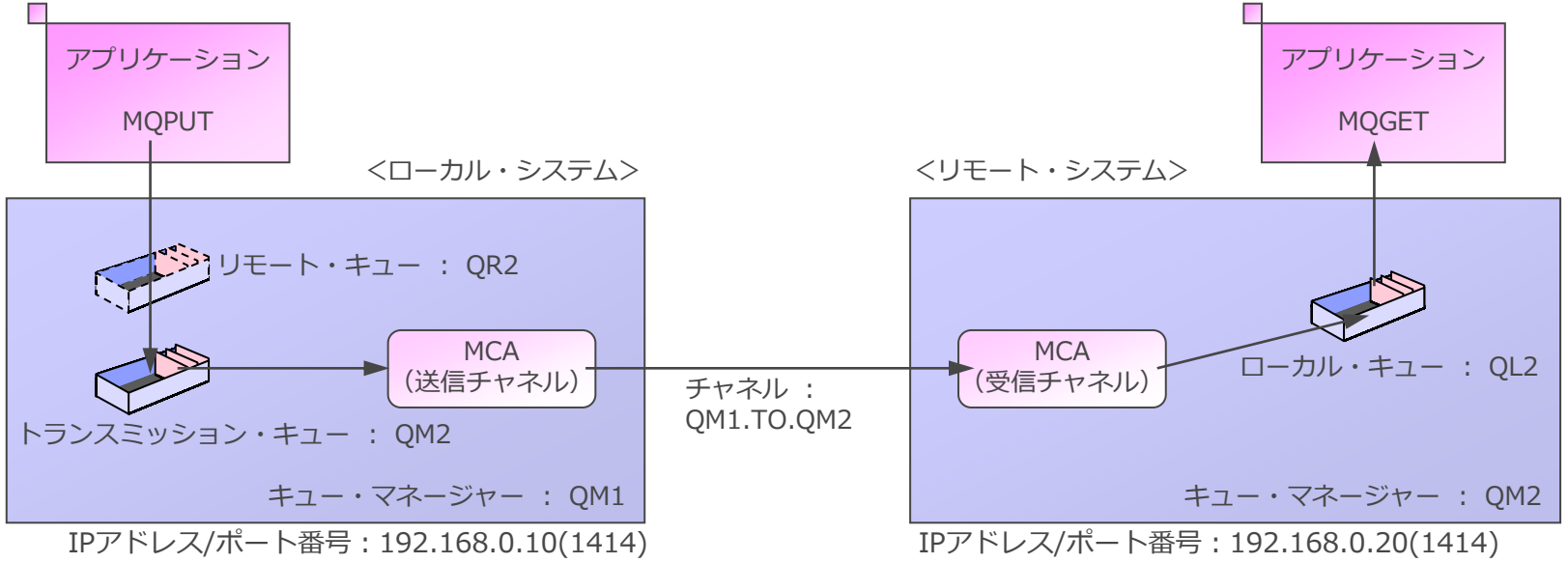


<参考> キュー・マネージャー間接続環境の構築

■ キュー・マネージャー間接続環境の構築

◆ 作成する環境

- キュー・マネージャー：QM1とQM2で構成されるキュー・マネージャー間接続環境を構築
- チャネル・タイプはSENDER－RECEIVERで作成
- QM1のアプリケーションから、QM2のローカル・キュー：QL2にメッセージを送信する



<参考> キュー・マネージャー間接続環境の構築

- QM1での作業（前提：キュー･マネージャーは作成済み）
  - ◆ リモート・キューの作成

```
DEFINE QREMOTE(QR2) RNAME(QL2) RQMNAME(QM2)
```

  - ◆ トランスミッション・キューの作成

```
DEFINE QLOCAL(QM2) USAGE(XMITQ)
```

  - ◆ 送信チャネルの作成

```
DEFINE CHL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP)
  CONNAME(192.168.0.20) XMITQ(QM2)
```

<参考> キュー・マネージャー間接続環境の構築

- QM2での作業（前提：キュー･マネージャーは作成済み）
  - ◆ ローカル・キューの作成

```
DEFINE QLOCAL(QL2)
```

  - ◆ 受信チャネルの作成

```
DEFINE CHL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(TCP)
```

  - ◆ リスナーの作成

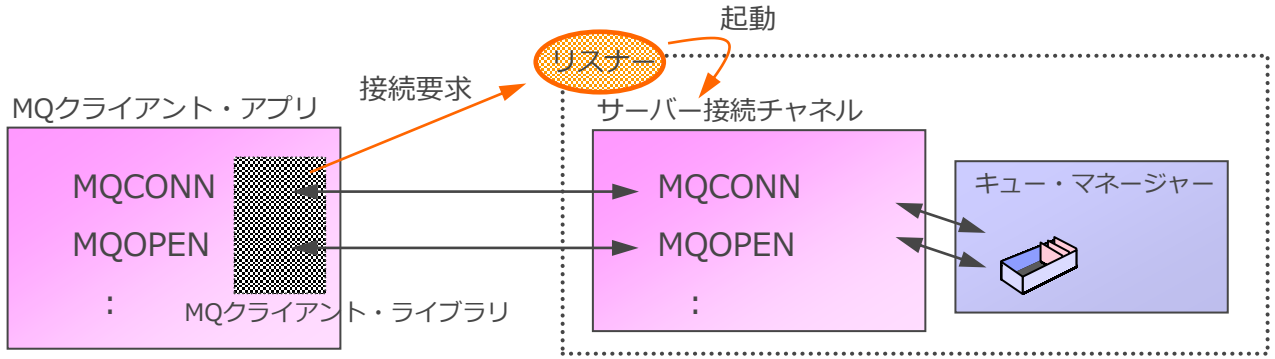
```
DEFINE LISTENER(LISTENER1414) TRPTYPE(TCP)
```

MQクライアント

MQクライアント接続

MQクライアント接続の動作

- MQCONNの発行により、キュー・マネージャー側でサーバー接続チャネルが起動
  - マルチスレッド・アプリケーションでは、スレッド間でTCP/IPソケットの共有も可能
- アプリケーションの通信相手はサーバー接続チャネル
  - サーバー接続チャネルがキュー・マネージャーと接続し、MQIコールを代替実行
- アプリケーションとサーバー接続チャネルは同期処理(リクエスト / リプライ通信)を行う
  - アプリケーションは、MQI実行要求を送信し、MQIの実行結果を応答として受信



- チャネル接続している間（MQCONN ～ MQDISC）はハートビートを送信
  - 全てのタイミングでハートビートによる障害検知が可能
  - キュー・マネージャー⇔MQクライアントの双方向で送信し双方で早期に障害を検知

MQクライアント接続

MQクライアント環境でのアプリケーション

◆ ローカル接続のアプリケーションとコーディングは同等

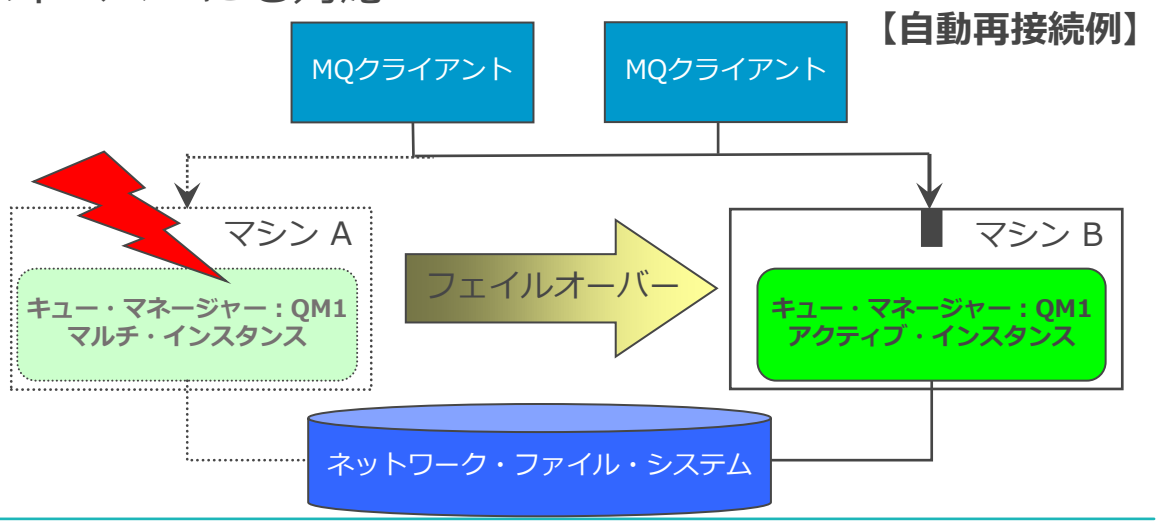
- C、C++、・・・　MQクライアント・ライブラリをリンクするとMQクライアント・アプリケーション
- Java　MQEnvironmentクラス(MQ Base Java)、接続ファクトリ(JMS/Jakarta Messaging)にローカル接続/クライアント接続を記述

◆ 考慮点、その他

- キュー・マネージャーをコーディネータとした2PhaseCommitは不可 (MQBEGINを発行できない)
- 明示的に同期点オプションを指定しないと、接続先プラットフォームのデフォルト値に従う
  - MQ for z/OSに接続　－－＞ 同期点処理に参加
  - MQ for Multiplatformに接続　－－＞ 同期点処理に参加しない
- キュー・マネージャーへ接続するために、接続先情報が必要　（後述）

クライアント自動再接続

- クライアント自動再接続機能とは
  - ◆ クライアント・ライブラリーが障害を検知して、再接続を行う機能
  - ◆ 同一、または異なるキュー・マネージャーへの再接続が可能
  - ◆ ライブラリー・レベルでの再接続。アプリケーションはキュー・マネージャーの再接続を検知しない
    - 自動でコネクション・ハンドルが入れ替わるため、アプリケーションでの考慮が必要
  - ◆ 再接続間隔は徐々に長くなる
  - ◆ 再接続タイムアウトを設定することが可能
  - ◆ Uniform Clusterを利用する場合は必須
    - Uniform Clusterは、クライアント自動接続を利用したクラスター機能
  - ◆ マルチインスタンス・キュー・マネージャーのフェイルオーバーにも対応
- クライアント自動再接続を使用するための設定項目
  - ◆ 自動再接続のON/OFF
    - MQCONNXのMQCNOオプション、mqclient.ini
  - ◆ 接続先キュー・マネージャーの登録
    - MQSERVER環境変数、チャネル定義テーブル
  - ◆ タイムアウト時間、自動再接続までの待機時間
    - mqclient.ini、コールバック関数コンテキスト(MQCBC)



接続先情報の与え方

■ 接続先情報の与え方

◆ MQクライアントがキュー・マネージャーに接続するために下記の情報が必要

「クライアント接続チャネル名」、「ネットワーク・プロトコル」、「接続先アドレス情報」

◆ 接続先情報の与え方は3種類

接続先情報の与え方	Java以外(C、C++、COBOL、・・・)	Java(MQ Base Java、JMS)
MQSERVER環境変数	○	×
チャネル定義テーブル	○	○
プログラム内に記述	○(MQCONNX)	MQ Base Java(MQEnvironmentクラス)
		(※)JMS(接続ファクトリ)

※以降、JMSと表記されている場合にはJakarta Messagingも含む

◆ 自動再接続を使用する場合は、「接続先アドレス情報」に複数アドレスを定義する

- ただし、MQ Base Javaでは自動再接続はサポートされない

MQSERVER環境変数

MQSERVER環境変数

◆ アプリケーションの起動環境に設定

- 「チャネル名」、「プロトコル」、「アドレス情報」を‘/’(スラッシュ)区切りで指定
- ‘,’(カンマ)区切りで「アドレス情報」を複数記載可能

```
(AIX,Linux)
    export  MQSERVER=クライアント接続チャネル名/プロトコル/アドレス情報
(Windows)
    set       MQSERVER=クライアント接続チャネル名/プロトコル/アドレス情報
(IBM i)
    set       ADDENVVAR ENVVAR(MQSERVER) VALUE('クライアント接続チャネル名/プロトコル/アドレス情報')
```

・例1.　MQSERVER=TO.QMGR01/TCP/111.111.111.111(1111)
・例2.　MQSERVER=TO.QMGR01/TCP/111.111.111.111(1111),111.111.111.222(2222)
・クライアント接続チャネル名はキュー・マネージャー側のサーバー接続チャネルに合わせる

◆ Java(MQ Base Java、JMS)からは使用できない

◆ クライアント接続チャネルの属性がデフォルト値になるため、チャネルの詳細オプションを利用できない

- SSL/TLSチャネル、チャネルのメッセージ圧縮機能を利用できない
- アプリケーション側でチャネルEXITを利用できない　　など

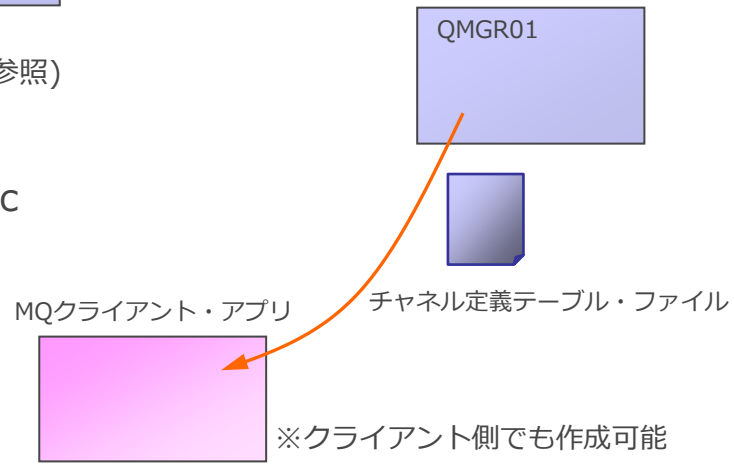
チャネル定義テーブル

- チャネル定義テーブル(CCDT)とは、クライアント接続のチャネル定義と認証情報を持つファイル
  - ◆ バイナリ形式もしくはJSON形式で作成可能
  - ◆ キュー・マネージャー側でクライアント接続チャネルを作成
    - V8.0からクライアント側でも作成可能
      - runmqsc –nを使用して作成
    - 対になるサーバー接続チャネルと同じ名前で作成
    - チャネル定義テーブル・ファイルにチャネル定義が保存される
  - ◆ バイナリ形式の場合

```
DEFINE CHANNEL(チャネル名) CHLTYPE(CLNTCONN)
    TRPTYPE(xxxx) CONNAME(xxxxx) QMNAME(xxxxxx)  ・・・
```

・同名のチャネル定義を記述するのは不可
・QMNAMEは、アプリケーションが接続先を検索するためのキー(詳細はP50「チャネル定義の検索」を参照)

- チャネル定義テーブル・ファイルの位置
  - IBM i (※1)　・・・　/QIBM/UserData/mqm/qmgrs/QUEUEMANAGERNAME/&ipcc
    ※1：統合ファイル・システムで上記を実行
  - AIX,Linux　・・・　/var/mqm/qmgrs/<QMgrName>/@ipcc/AMQCLCHL.TAB
  - Windows (※2)　・・・　<data_dir>¥qmgrs¥<QMgrName>¥@ipcc¥AMQCLCHL.TAB
    ※2：<data_dir>のデフォルトは、 C:¥ProgramData¥IBM¥MQ



チャネル定義テーブル

◆ JSON形式の場合

- V9.1.2からJSON形式のCCDTが作成可能

```
{
 "channel":
 [
  {
   "general":
   {
    "description": "a channel"
   },
   “name”: “channel”,                    //クライアント接続チャネル名
   "clientConnection":
   {
    "connection":
    [
     {
      “host”: “xxx.xxx.xxx.xxx”,        //接続先IPアドレス
      “port”: xxxx                      //接続先ポート番号
     }
    ],
    "queueManager": "QMNAME"
   },
   "type": "clientConnection"
  }
 ]
}
```

・同名のチャネル定義も記述可
(詳細は「サンプルのJSON形式CCDT」を参照)

- JSON形式のCCDTスキーマの位置
  - AIX,Linux ・・・ /opt/mqm/lib/ccdt_schema.json
  - Windows (※) ・・・ <inst_dir>¥bin¥ccdt_schema.json ※<inst_dir>のデフォルトは、 C:¥Program Files¥IBM¥MQ
- JSON形式のCCDTで使用可能な属性は下記を参照
  - https://www.ibm.com/docs/en/ibm-mq/9.3?topic=ccdt-channel-attributes-supported-by-json
- サンプルのJSON形式CCDT
  - https://www.ibm.com/docs/en/ibm-mq/9.3?topic=ccdt-json-examples

チャネル定義テーブル

■ アプリケーション側のチャネル定義テーブル・ファイルの指定

◆ C、C++、COBOL・・・(Java以外)

- チャネル定義テーブル・ファイルをアプリケーション側に配布する場合
  - クライアント端末へのファイル転送を行う場合は、バイナリ・モードで転送
  - デフォルトの配置先

    AIX,Linux ・・・ /var/mqm/AMQCLCHL.TAB

    Windows (※) ・・・ <inst_dir>¥AMQCLCHL.TAB

    ※<inst_dir>のデフォルトは、 C:¥Program Files¥IBM¥MQ

    IBM i ・・・ /QIBM/UserData/mqm/AMQCLCHL.TAB

  - 環境変数の設定で任意の位置、名前に配置することも可能

    AMQCHLLIB ・・・ 配置ディレクトリ

    AMQCHLTAB ・・・ チャネル定義テーブルのファイル名

- リモート・サーバー上のチャネル定義テーブル・ファイルを参照する場合
  - URLでチャネル定義テーブル（CCDT）の保管場所を指定
    - “ftp://”、“file://” 、“http://” プロトコルの指定をサポート

  ① URL指定方法: クライアント・プログラムで指定
    - CCDTUrlLength：URL Stringの長さ
    - CCDTUrlOffset：CCDTを指定するURLの開始位置オフセット（CNOの開始からの）
    - CCDTUrlPtr：CCDTを指定するURLへのポインター

チャネル定義テーブル

① URL指定方法: クライアント・プログラムで指定(続き)
- コーディング・イメージ

```
cno.Version = MQCNO_VERSION_6;
cno.CCDTUrlPtr = “file://var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB”;
cno.CCDTUrlLength = strlen(cno.CCDTUrlPtr);
MQCONNX(qmname, &cno, &hconn, &CompCode, &Reason);
```

② URL指定方法:環境変数
- V9.0から追加された環境変数：MQCCDTURLを使用
  - これまでのMQCHLLIBとMQCHLTABを組み合わせた環境変数
    - URL形式のみサポート

```
MQCCDTURL=file://var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB
```

  - ベーシック認証を使用可能

```
MQCCDTURL=ftp://user:password@example.com//files/MYCHLTAB.TAB
```

③ URL指定方法:MQクライアント構成ファイル（mqclient.ini）
- チャネルスタンザのChannelDefinitionDirectory / ChannelDefinitionFile に指定

```
CHANNELS:
ChannelDefinitionDirectory = file:///C:¥client
ChannelDefinitionFile = MYAMQCLCHL.TAB
```

チャネル定義テーブル

アプリケーション側のチャネル定義テーブル・ファイルの指定

◆ Java

- MQ Base Java
  - MQQueueManagerを作成するとき、チャネル定義テーブル・ファイルの位置をURL指定で渡す

```
java.net.URL table = new URL("file:///home/admdata/ccdt1.tab");
MQQueueManager qmgr = new MQQueueManager("*QMGR_A", table);
                    :
```

・file以外のftpプロトコルなども利用可能

- JMS
  - コネクション・ファクトリでチャネル定義テーブル・ファイルの位置をURL指定

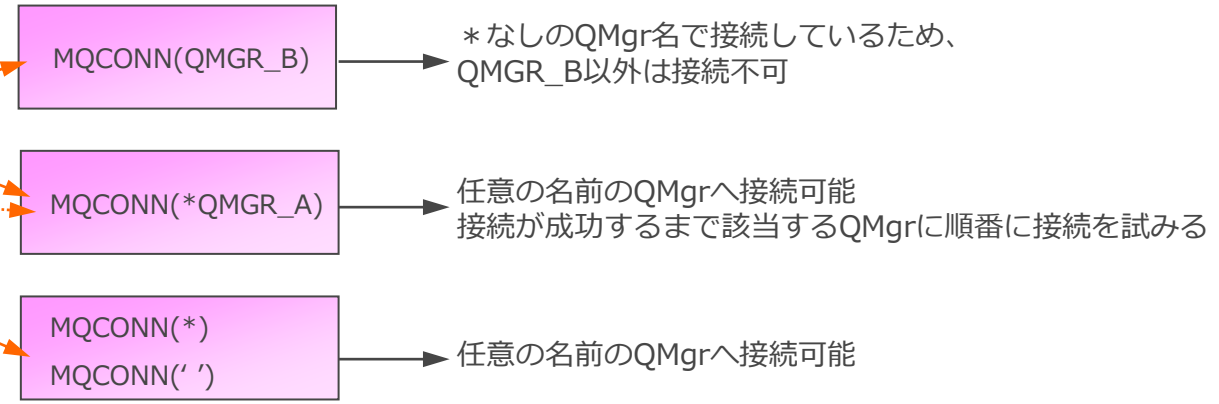
チャネル定義テーブル

- チャネル定義の検索
  - チャネル定義テーブルには、複数のクライアント接続チャネルの定義を含めることが可能
  - MQCONNに指定したキュー・マネージャー名、クライアント接続チャネルのQMNAME属性から接続に使用するチャネル定義を決定

MQCONNの指定	クライアント接続チャネルのQMNAMEパラメータ	実際に接続するQMgr
QMgr名	QMNAMEがQMgr名のチャネル定義を検索 ・複数のチャネル定義がマッチした場合、チャネル名で優先順位付け 数字 –> 英大文字 –> 英小文字	指定したQMgrと同一の必要あり
＊QMgr名		チェックされない
＊、無指定(ブランク)	QMNAMEがブランクのチャネル定義を検索	チェックされない

チャネル定義テーブル

クライアント接続チャネル名	CONNAME	QMNAME
TO.QMGR_A.01		QMGR_A
TO.QMGR_B.01		QMGR_B
TO.QMGR_A.02		QMGR_A
TO.QMGR_A.03		

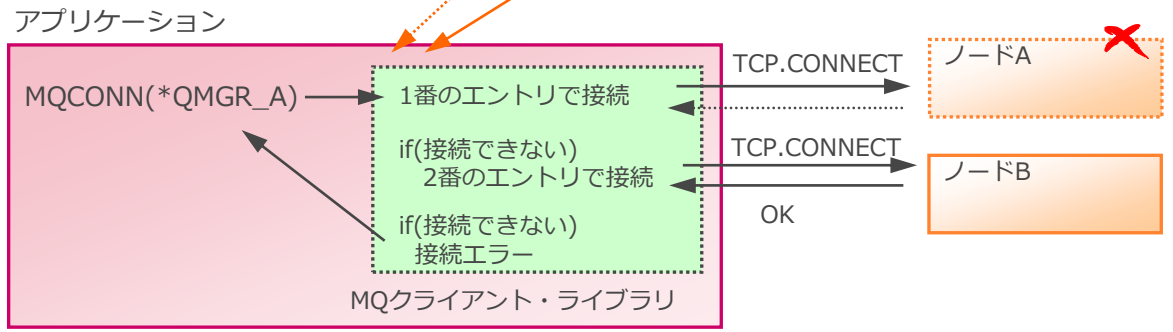


チャネル定義テーブル

- グループ接続
  - ◆ チャネル接続が失敗した際に、次の条件にマッチするチャネル定義を使って自動で接続を試行させることが可能
    - チャネル定義テーブルに複数のエントリを事前に定義しておく
  - ◆ 接続先の切り替えは、アプリケーションに透過的に行われる
    - MQクライアント・ライブラリは、内部的にチャネル定義テーブル内のエントリに対して接続を順番に試行する
    - 条件にマッチする全エントリへの接続が失敗すると、MQCONNがエラー(MQRC_Q_MGR_NOT_AVAILABLE)

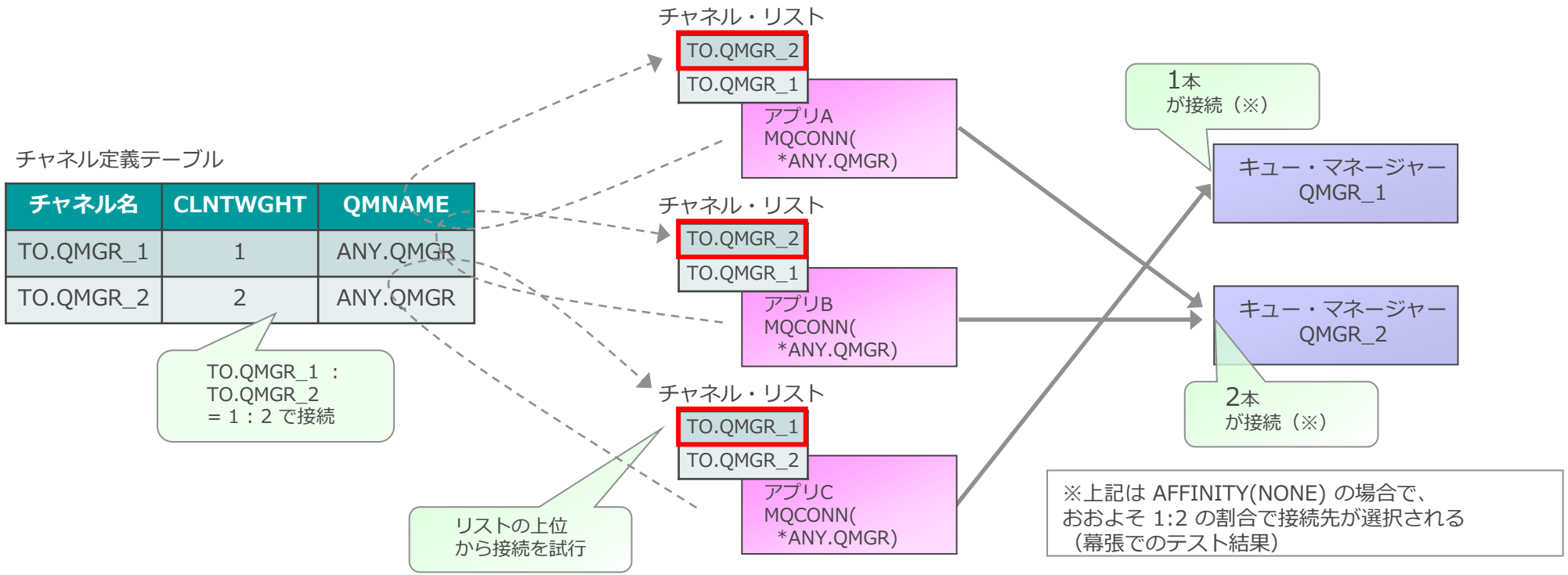
チャネル定義テーブル

クライアント接続チャネル名	CONNAME	QMNAME
TO.QMGR_A.01		GROUP_A
TO.QMGR_B.01		GROUP_B
TO.QMGR_A.02		GROUP_A
TO.QMGR_A.03		



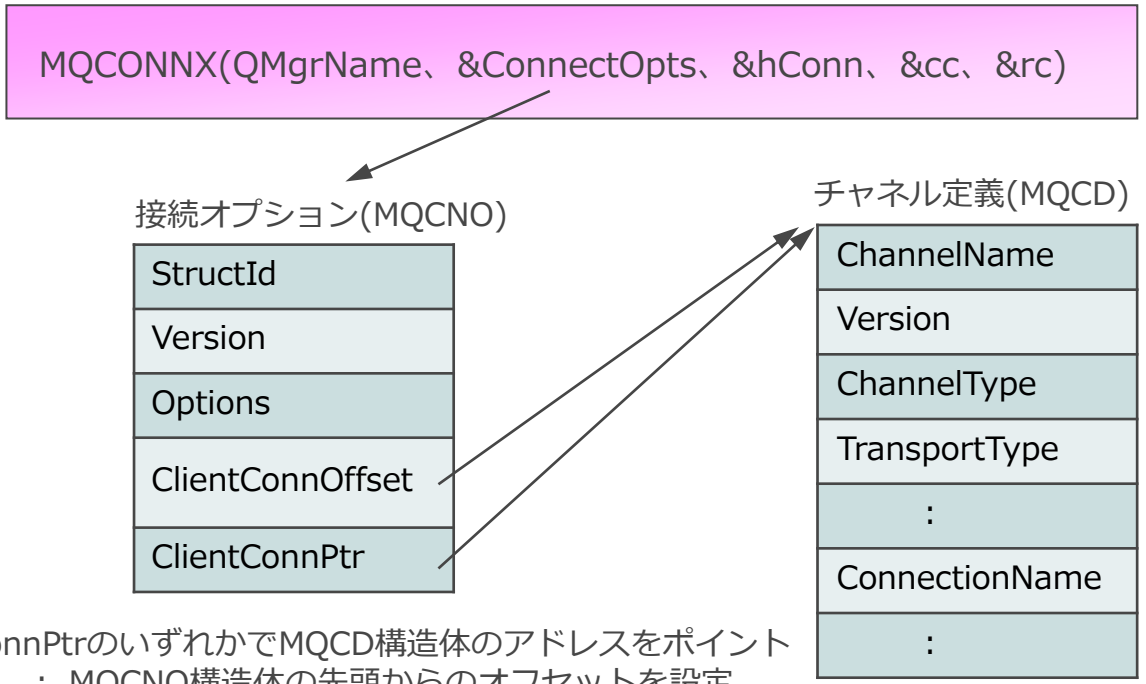
チャネル定義テーブル

- クライアント・チャネルの重み付け
  - ◆ クライアント・チャネルに重み付けをし、接続先を分散させることが可能
    - チャネル定義テーブルでグループ化されているクライアント接続チャネルが対象
    - CLNTCONNのAFFINITY属性がNONEの場合、CLNTWGHTに指定された値の割合で接続先を分散
  - ◆ 内部的に作成するチャネル・リストを基にチャネルを選択
    - プロセス（アプリケーション）内の初回MQCONN時に、チャネル定義テーブルを参照し、チャネル・リストを作成



プログラム内に記述

- C、C++、・・・(Java以外)
  - ◆ MQCD構造体でクライアント接続チャネルをプログラム内に定義
    MQCONNXでキュー・マネージャーに接続



・ClientConnOffset、または、ClientConnPtrのいずれかでMQCD構造体のアドレスをポイント
　ClientConnOffset　　：MQCNO構造体の先頭からのオフセットを設定
　ClientConnPtr　　：MQCD構造体のアドレスを設定

プログラム内に記述

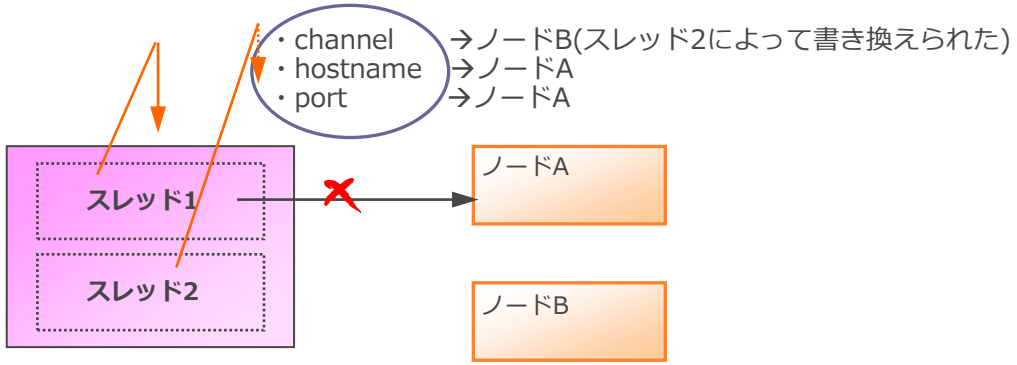
- Java
  - ◆ MQ Base Java
    - MQEnvironmentクラスに接続先情報を設定

```
MQEnvironment.channel      = “xxxxxxxxx”;
MQEnvironment.hostname     = “111.111.111.111”;
MQEnvironment.port          =  1111;

MQQueueManager qmgr = new MQQueueManager(“＊”);
                    :
```

  - ◆ JMS
    - コネクション・ファクトリのTRANSPORT属性をCLIENTに設定
      CHANNEL、HOSTNAME、PORT属性に接続先情報を設定
      複数アドレスを記載する場合は、CONNECTIONNAMELIST属性に‘,’(カンマ)区切りで接続先情報を設定
      - 他属性は、以下の「JMSオブジェクトのプロパティー」を参照
        - https://www.ibm.com/docs/en/ibm-mq/9.3?topic=reference-properties-mq-classes-jms-objects

<参考> プログラム内に記述

- MQ Base Java利用時の考慮点
  - MQEnvironmentはstaticクラス
  - 複数スレッドが異なるキュー・マネージャーへの接続/切断を繰り返すと、
    接続先情報が中途半端に書き換えられた状態で接続にいく可能性がある

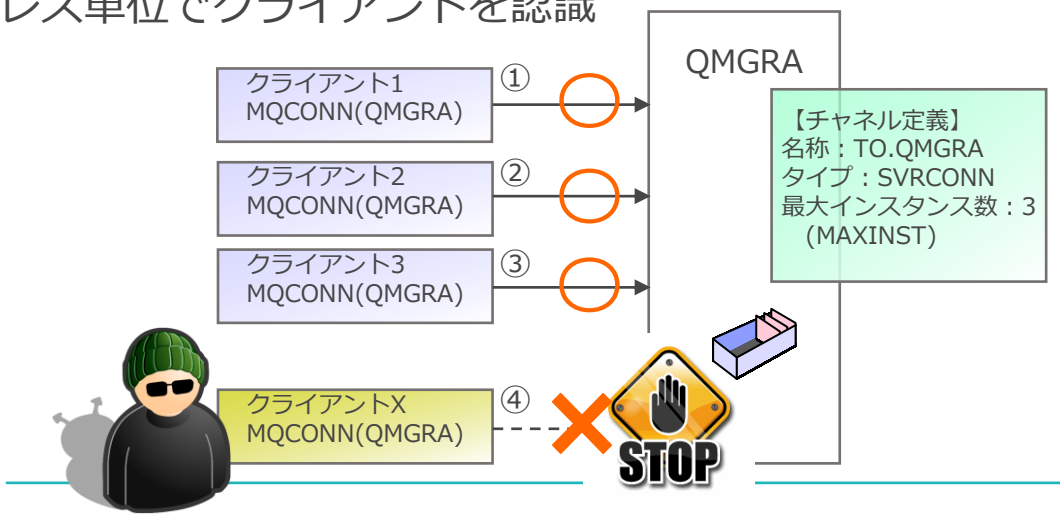


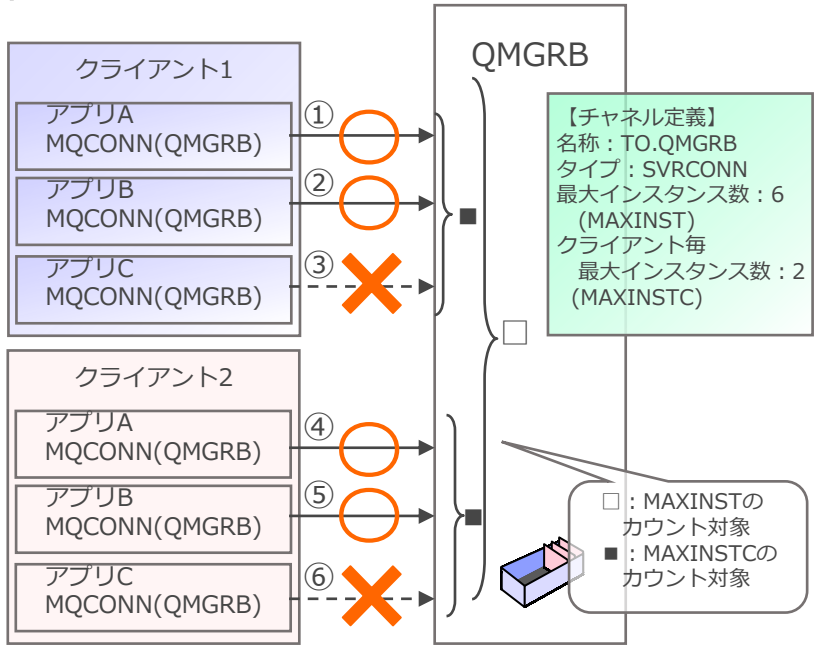
  - 下記の方法で接続

```
java.util.Hashtable properties = new Hashtable();
properties.put(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_CLIENT);
properties.put(MQC.CHANNEL_PROPERTY, “xxxxxxxxx”);
properties.put(MQC.HOST_NAME_PROPERTY, "111.111.111.111");
properties.put(MQC.PORT_PROPERTY, new Integer(1111));

MQQueueManager qmgr = new MQueueManager(“QMGR01”, properties);

                    :
```

チャネル数の制御

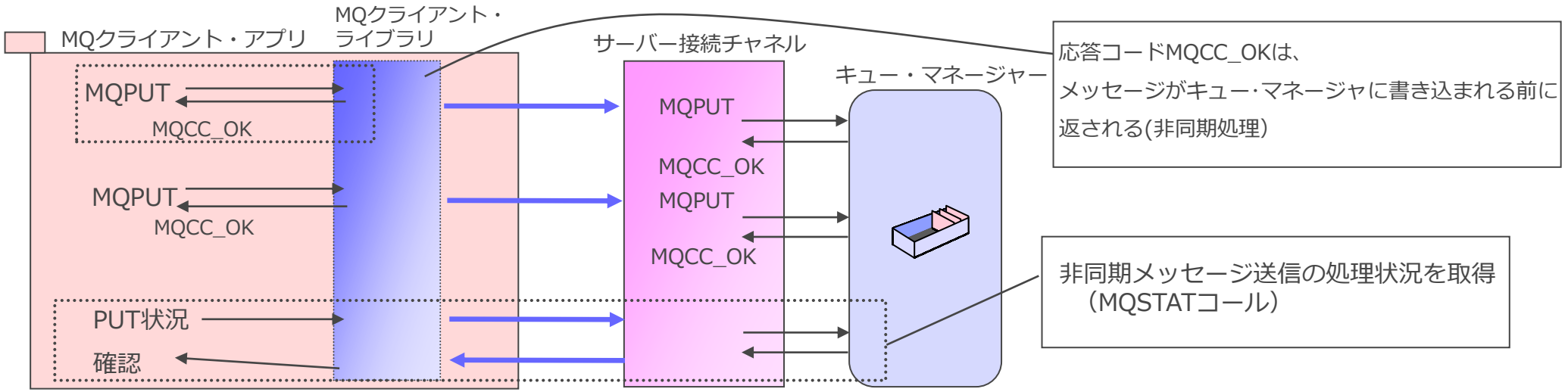
- SVRCONNチャネル毎にクライアントからのインスタンス数を制限可能
  - ◆ キュー・マネージャーが存在するマシンのリソースを制御
  - ◆ 単一クライアントが接続を独占することを防止
    - プログラムの不具合
    - DoS（Denail of Service）攻撃
    - 計画外MQクライアント・アプリケーションの接続
- SVRCONNチャネルのMAXINST、MAXINSTC属性により動作を制御
  - ◆ MAXINST・・・最大チャネル・インスタンス数
  - ◆ MAXINSTC・・・クライアントあたりの最大チャネル・インスタンス数
    - IPアドレス単位でクライアントを認識





非同期メッセージ送信

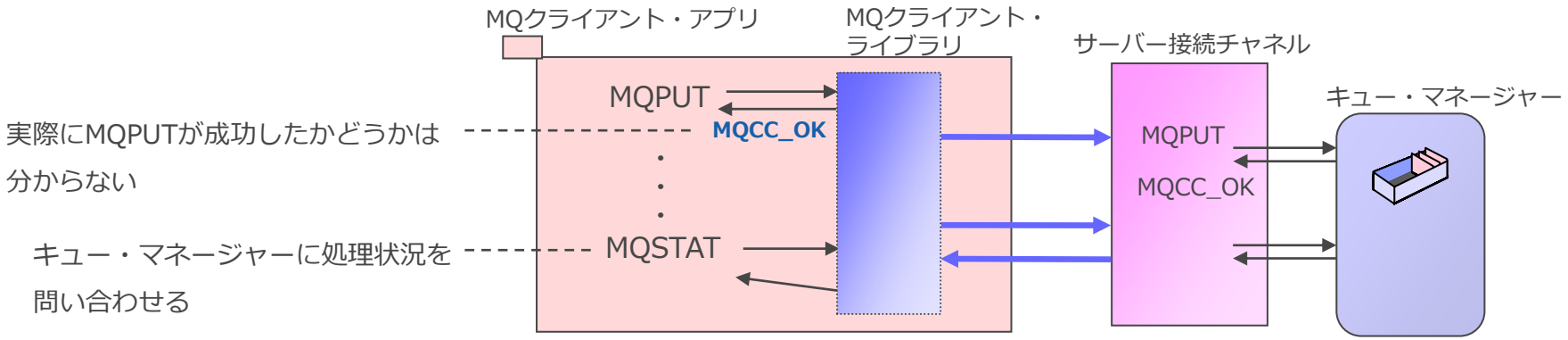
- キュー・マネージャーからの応答を待つことなく、非同期にメッセージをキューに書き込む機能
  - MQPUTの応答時間が短縮
  - キューにメッセージが書き込まれた事をそのつど確認する必要のないアプリケーションのパフォーマンスが向上



- 非同期メッセージ送信を有効にするにはキューやアプリケーションでの設定が必要
  - キューのDEFPRESP属性で設定
  - アプリケーションで指定
    - MQI、MQ Base Javaアプリケーションは MQPMOで設定
      - 優先順位：MQPMOの設定　＞　キューのDEFPRESP属性の設定
    - JMSでの設定
      - 宛先オブジェクトにPUTASYCALLOWED（YES）を設定、かつ、キュー属性DEFPRESP（ASYNC）を設定

非同期メッセージ送信

- 非同期のMQPUTでは、即座に応答コード（正常終了）が返る
  - メッセージがキューに正常に書き込まれたかどうかは判別できない
  - キューにメッセージが書き込まれたかを確認するには、MQSTATコールを用いてキュー・マネージャーから応答コードを取得することが必要



非同期メッセージ送信の制限事項

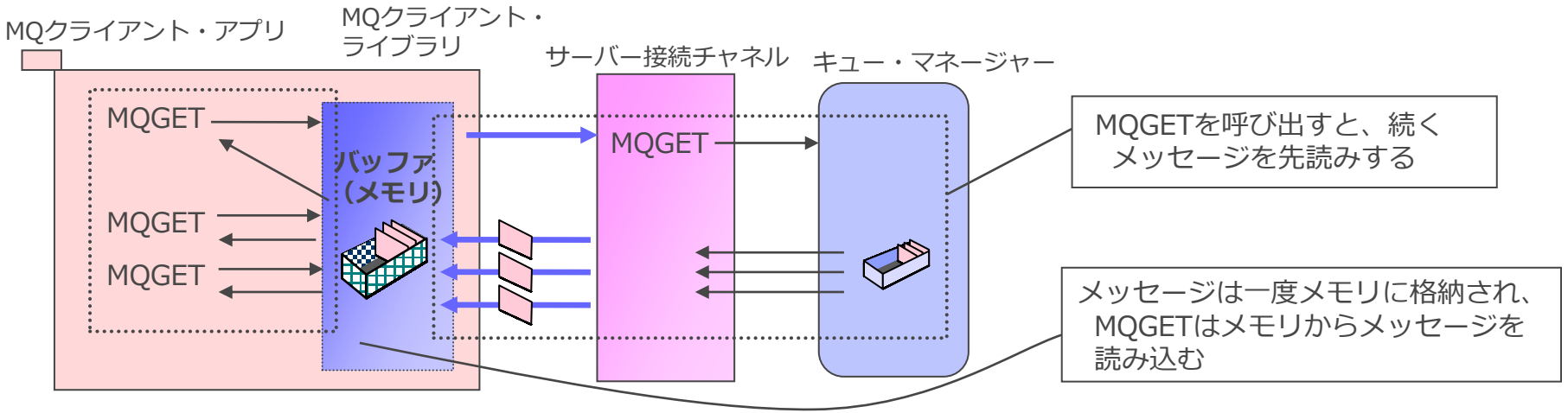
- ノン･パーシステント・メッセージと同期点付きのパーシステント・メッセージのみ、非同期的にメッセージが送信される
  - ◆ 同期点なしのパーシステント・メッセージを送信する場合、メッセージは同期的に送信される

	MQPUT(SYNCPOINT)	MQPUT(NO_SYNCPOINT)
パーシステント	非同期PUT可	同期（非同期PUT不可）
ノン･パーシステント	非同期PUT可	非同期PUT可

  - ◆ MQPUT(SYNCPOINT)、MQPUT(NO_SYNCPOINT)を混ぜて発行することが可能
- LOGICAL_ORDERを用いてのグループ・メッセージの非同期メッセージ送信は不可
  - ◆ 自動的に同期通信に切り替わる
  - ◆ MQクライアント側でGroupID、MsgSeqNumberを設定すれば非同期送信が可能

メッセージ先読み

- 一度のMQGETで複数のメッセージをキュー・マネージャーから先読みする機能
  - ノン･パーシステント・メッセージがメッセージ先読みの対象
  - 大量のノン･パーシステント・メッセージを読み込む際のパフォーマンスが向上



- メッセージ先読み有効にするにはキューやアプリケーションでの設定が必要
  - キューのDEFREADA属性で設定
  - アプリケーションの指定
    - MQI、MQ Base Javaアプリケーションは MQOO （MQ Open Option）で設定
      - 優先順位：MQOOの設定　＞　キューのDEFREADA属性の設定
    - JMSでの設定
      - 宛先オブジェクトにREADAHEADALLOWED(YES)を設定

メッセージ先読み

- 先読みするメッセージの量や頻度の調整が可能
  - mqclient.iniファイルのMessageBufferスタンザで値を設定
    - Maximumsize (Kbytes)
      - メッセージを読み込むバッファサイズを指定。メッセージはMaximumサイズ分先読みされる
    - UpdatePercentage (%)
      - バッファに入っているメッセージが残り何%になったら再び先読みを行うかを指定
    - PurgeTime(seconds)
      - バッファに入っているメッセージを何秒で削除するか

メッセージ先読みの制限事項

- ノン・パーシステント・メッセージのみメッセージ先読みが可能
  - パーシステント・メッセージの場合、メッセージ先読み機能は自動的にOFFになる
- 同期点つきのMQGETは不可
  - メッセージ先読み開始前、MQGMO_SYNCPOINTを指定した場合
    - メッセージ先読み機能は自動的にOFFになる
  - メッセージ先読み開始後、MQGMO_SYNCPOINTを指定した場合
    - MQRC_OPTIONS_ERRORが返る

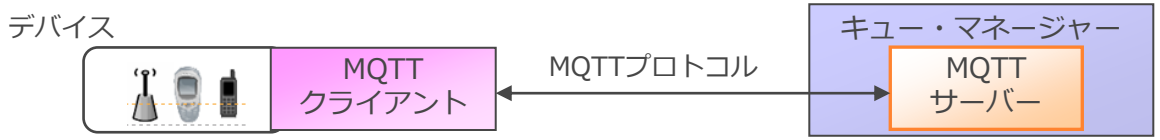
	MQGET(SYNCPOINT)	MQGET(NO_SYNCPOINT)
パーシステント	不可	不可
ノン･パーシステント	不可	可

- MQクライアントで接続のみメッセージ先読みが可能
  - ローカル接続の場合、メッセージ先読み機能は自動的にOFFになる

その他チャネル

MQ Telemetryサービス、MQTTチャネル

- MQ Telemetryは、軽量なパブリッシュ/サブスクライブ処理を実現するIBM MQのフィーチャー
  - Cloud Pak for IntegrationもしくはMQ Advancedのライセンスが必要
  - Windows、Linux、AIXで使用可能
  - 様々な機器上のデバイスとMQメッセージ基盤を連携可能
  - オープン・プロトコルMQTT（MQ Telemetry Transport）を通信プロトコルに使用



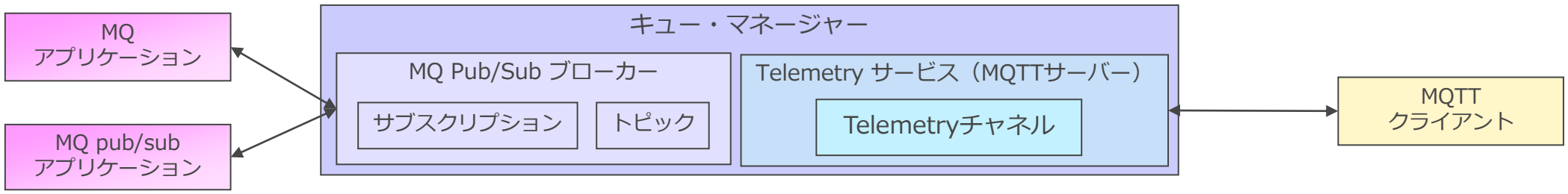
- 構成コンポーネント
  - Telemetryサービス
    - MQTTのサーバー機能を提供するコンポーネント
    - MQのサービスとして稼働（Javaプロセス）
    - キュー・マネージャーに１つのみ構成可能
  - Telemetryチャネル
    - MQTTクライアントからの接続を受け付け、通信を行うコンポーネント
      - Telemetryチャネルのプロパティで使用するポートを指定
    - 複数のチャネルを構成可能
      - クライアントをグループ化し、異なった設定のチャネルを使い分けることが可能
    - チャネル毎にJAASやSSL/TLSのセキュリティ設定が可能

MQ Telemetryサービス、MQTTチャネル

■ 構成コンポーネント（続き）

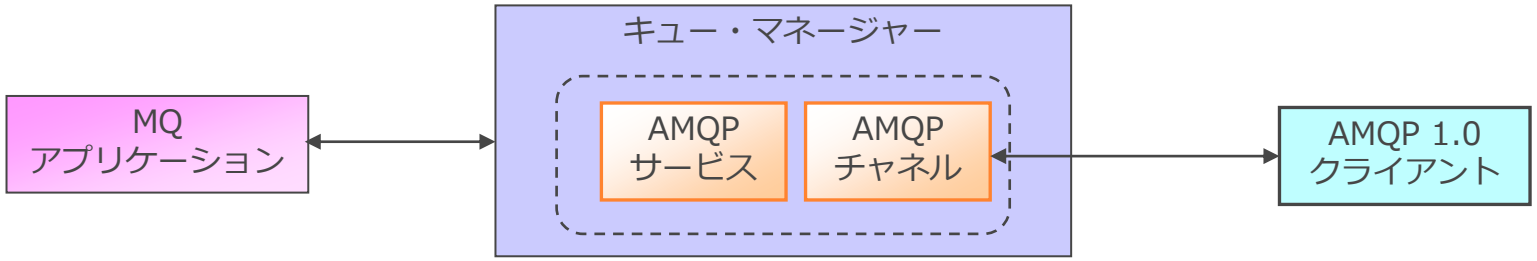
◆ MQTTクライアント

- MQ TelemetryのAPIを使用して、Telemetryチャネルに接続し、Pub/Sub通信を実施
- Java、C、JavaScriptで実装可能
- MQTTクライアントのライブラリは、以下のサイトにて入手可能
  - Eclipse Paho project : https://eclipse.dev/paho/
  - MQTT.org : https://mqtt.org/



AMQP(Advanced Message Queuing Protocol)チャネル

- MQアプリケーションとAMQP 1.0クライアントの相互連携が可能
  - ◆ Windows、Linux、AIXで使用可能
    - MQキュー・マネージャーが、AMQP 1.0クライアントのブローカーとして機能



- AMQPチャネルの特徴
  - ◆ AMQP 1.0プロトコルのサブセットをサポート
  - ◆ AMQPチャネルを利用するためにAMQPサービスの稼働が必須
  - ◆ AMQP 1.0互換クライアントが接続可能
  - ◆ AMQP 1.0仕様のうちの主な非サポート機能
    - トランザクション
    - 複数セッション
    - その他は、KnowledgeCenter 「AMQP 1.0サポート」を参照
      - https://www.ibm.com/docs/ja/ibm-mq/9.3?topic=applications-amqp-10-support
    - MQの他のチャネルと異なり、リスナーを内包
      - デフォルトのポートは、5672
    - SYSTEM.DEF.AMQP がポート5672で定義済み（要起動）