オブジェクト指向論 第5,6回
― モデリング 動的モデル―

6回目Keynote:シーケンス図、コミュニケーション図
7回目Keynote: 状態遷移図

立命館大学　情報理工学部
槇原　絵里奈
Mail: makihara@fc.ritsumei.ac.jp

講義内容

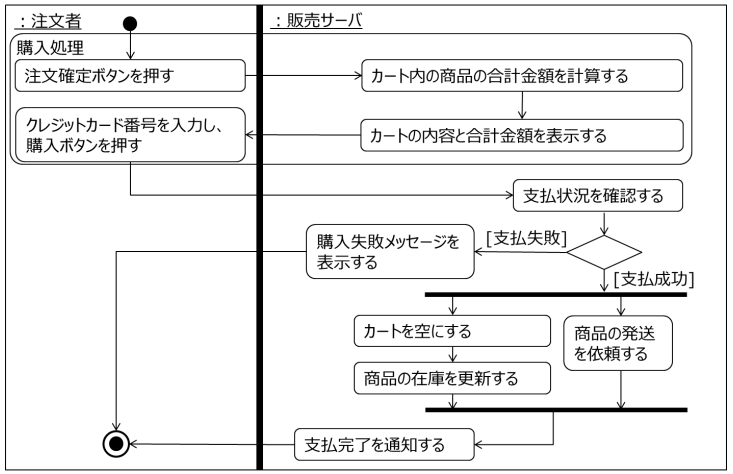
- 動的モデリング
  - アクティビティ図
  - シーケンス図
  - コミュニケーション図
  - 状態機械図

動的モデリング (dynamic modeling)

- ソースコード上に直接表れる構造ではなく、**プログラム稼働時の振る舞い**をモデル化
  - 静的
  - 動的
- 特定の機能に着目した処理の流れ
  - アクティビティ図(activity diagram)
- オブジェクト間の相互作用(interaction)
  - シーケンス図(sequence diagram)
  - コミュニケーション図(communication diagram)
- オブジェクトの状態遷移
  - 状態機械図(state machine diagram)

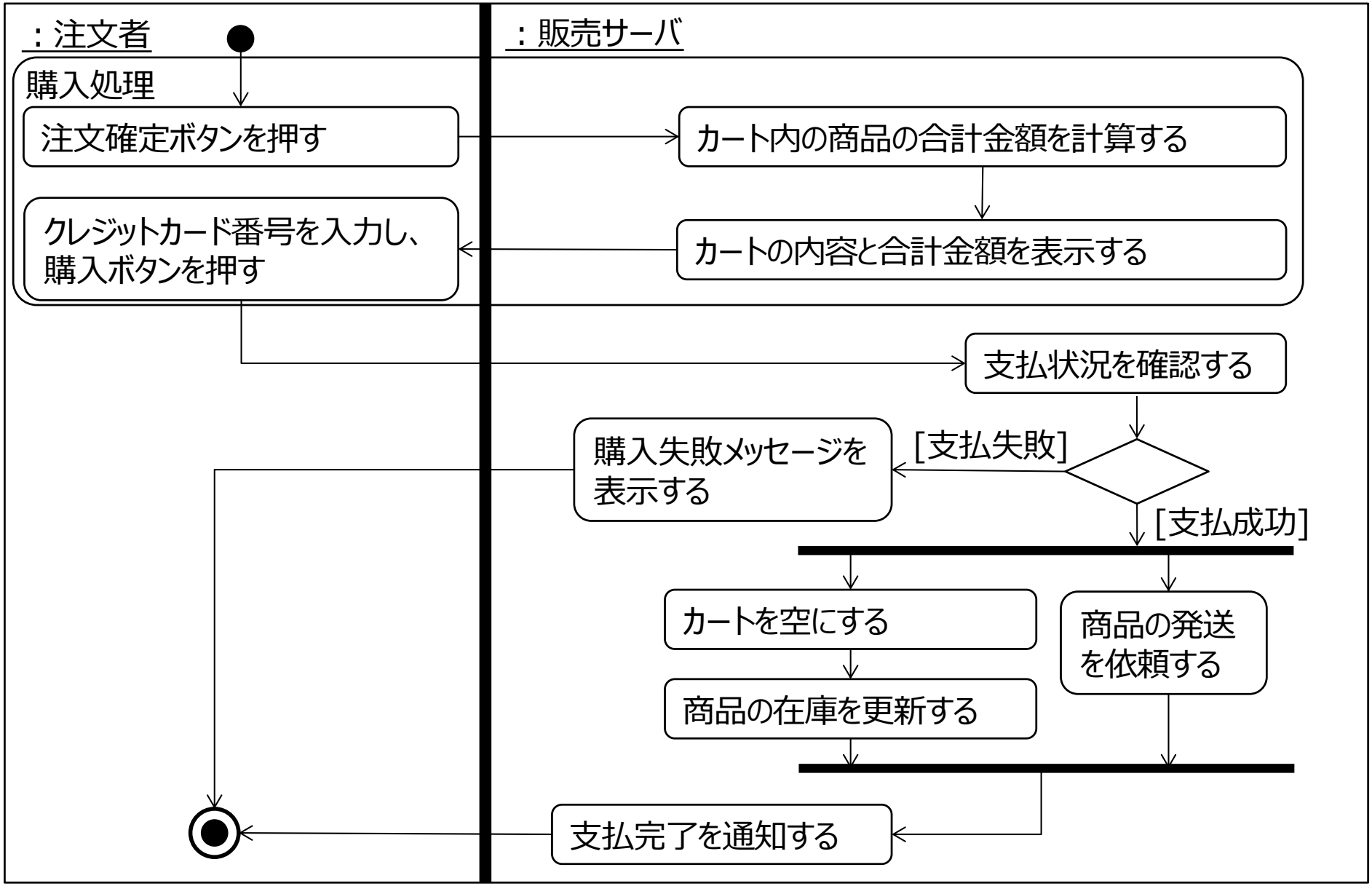
アクティビティ図

- 特定の機能に着目した場合の処理の流れ(実行手順)を表現
- **並列**した振る舞いも表現可能　同時に処理
  - 例：通信処理と画面描写(Now loading…)
- フローチャートに近いため、UMLに詳しくなくても使いやすい
- 上流工程(ビジネスプロセスの分析と記述)・下流工程(プログラムの詳細な制御フロー)ともに使用可能



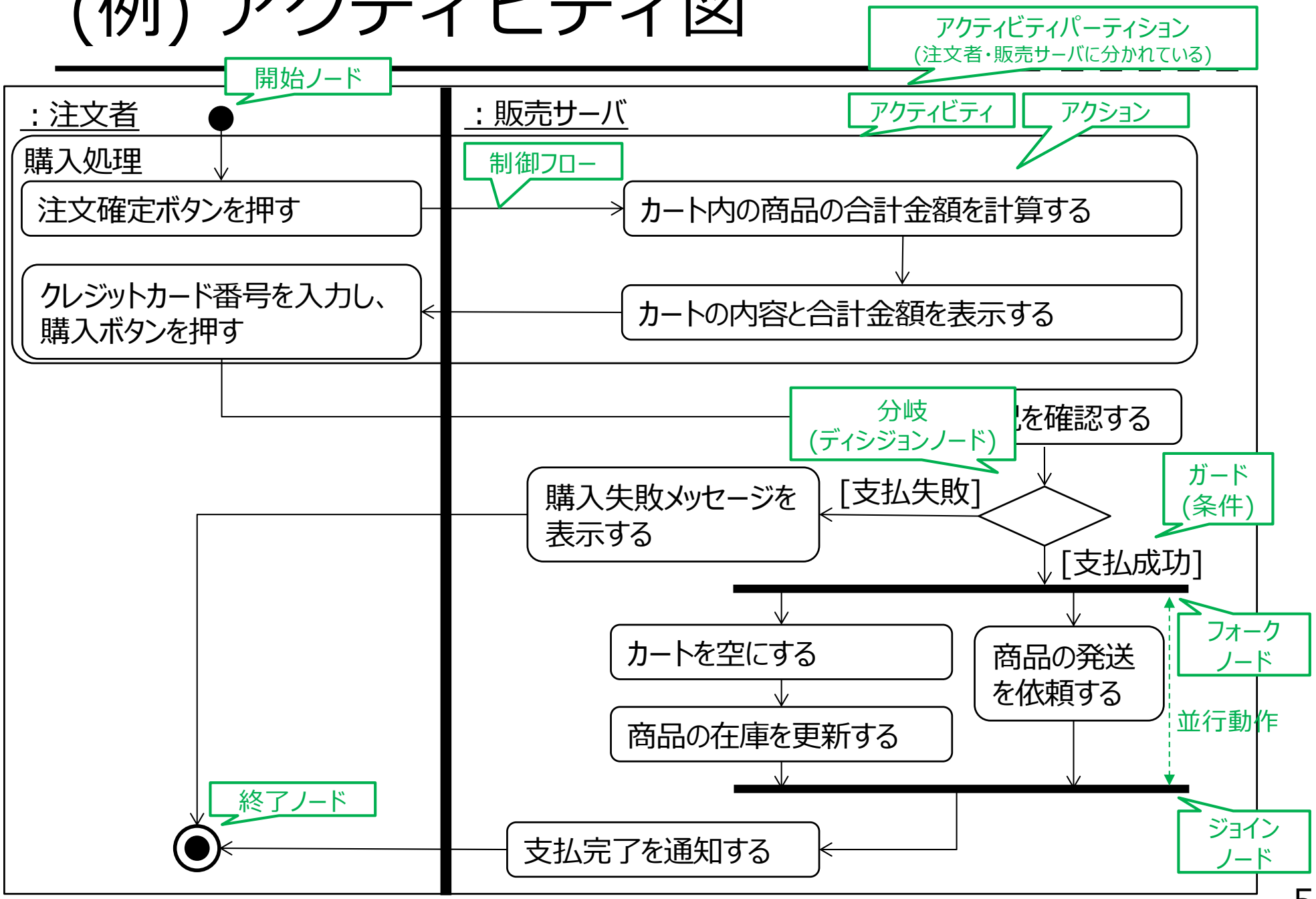
(例) アクティビティ図

ある機能を実現するために
必要な作業の順序を示す

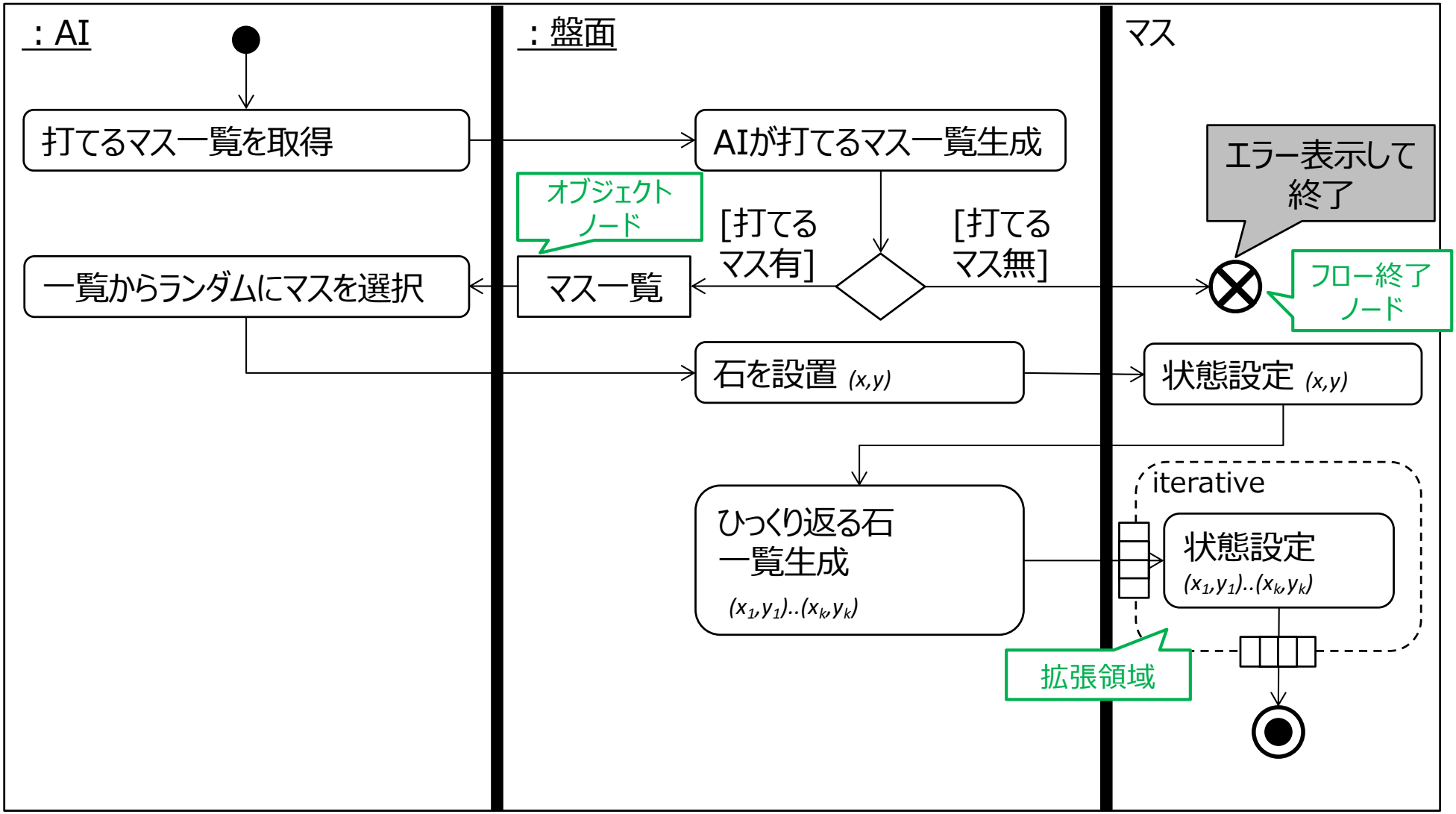


ある機能を実現するために
必要な作業の順序を示す

(例) アクティビティ図



(例) アクティビティ図

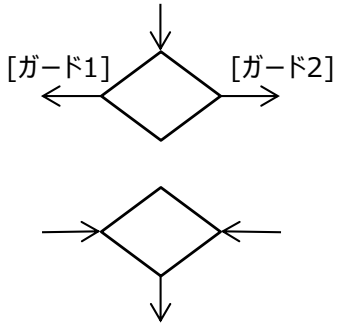

：AI
：盤面
マス
打てるマス一覧を取得
AIが打てるマス一覧生成
エラー表示して終了
オブジェクトノード
[打てるマス有]
[打てるマス無]
一覧からランダムにマスを選択
マス一覧
フロー終了ノード
石を設置 (x,y)
状態設定 (x,y)
iterative
ひっくり返る石一覧生成
(x1,y1)..(xk,yk)
状態設定
(x1,y1)..(xk,yk)
拡張領域

アクティビティ図の構成要素

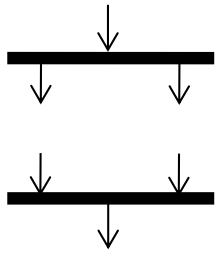
- 開始ノード(initial node)：アクティビティ開始
- 終了ノード(final node)
  - アクティビティ終了ノード：アクティビティの終了
  - フロー終了ノード：分岐したフローの終了(特に異常終了等)や並行フローの一つの終了
- アクティビティ(activity)：**一連の処理**。複数のアクションで構成
- アクション (action)：アクティビティを構成する**処理の単位**
  (他のアクションを内包できない=**これ以上分割できない**)
- 制御フロー (control flow)：**制御の流れ**。**アクションの順序**を示す
- アクティビティパーティション (activity partition)：
  アクションを実行する主体(垂直方向)や
  フェーズごと(水平方向)の切り分け。
  スイムレーン(swimlane)とも呼ばれる

アクティビティ図の構成要素

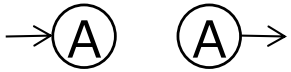
- ディシジョンノード(decision node)
  - 分岐条件はガード(guard)で表記
- マージノード(merge node)
  - 分岐したフローの合流



- フォークノード(fork node)：並行動作の開始
- ジョインノード(join node)：並行動作の終了
  フォークノード～ジョインノードの間
  の処理は並行的に行われる



- コネクタ(connecter)：図中の離れた地点を連結

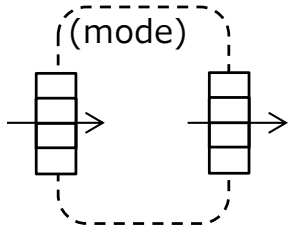


アクティビティ図の構成要素

- オブジェクトノード(object node)：
  アクション間でオブジェクトをやりとりする場合に使用

  - オブジェクトフロー：オブジェクトに出入りするフロー

- 拡張領域(expansion region)：複数要素の処理
  以下の3つのモードから選択
  - parallel：要素の処理を並行して行う
  - iterative：要素を順番に処理
  - stream：要素をストリームとして処理

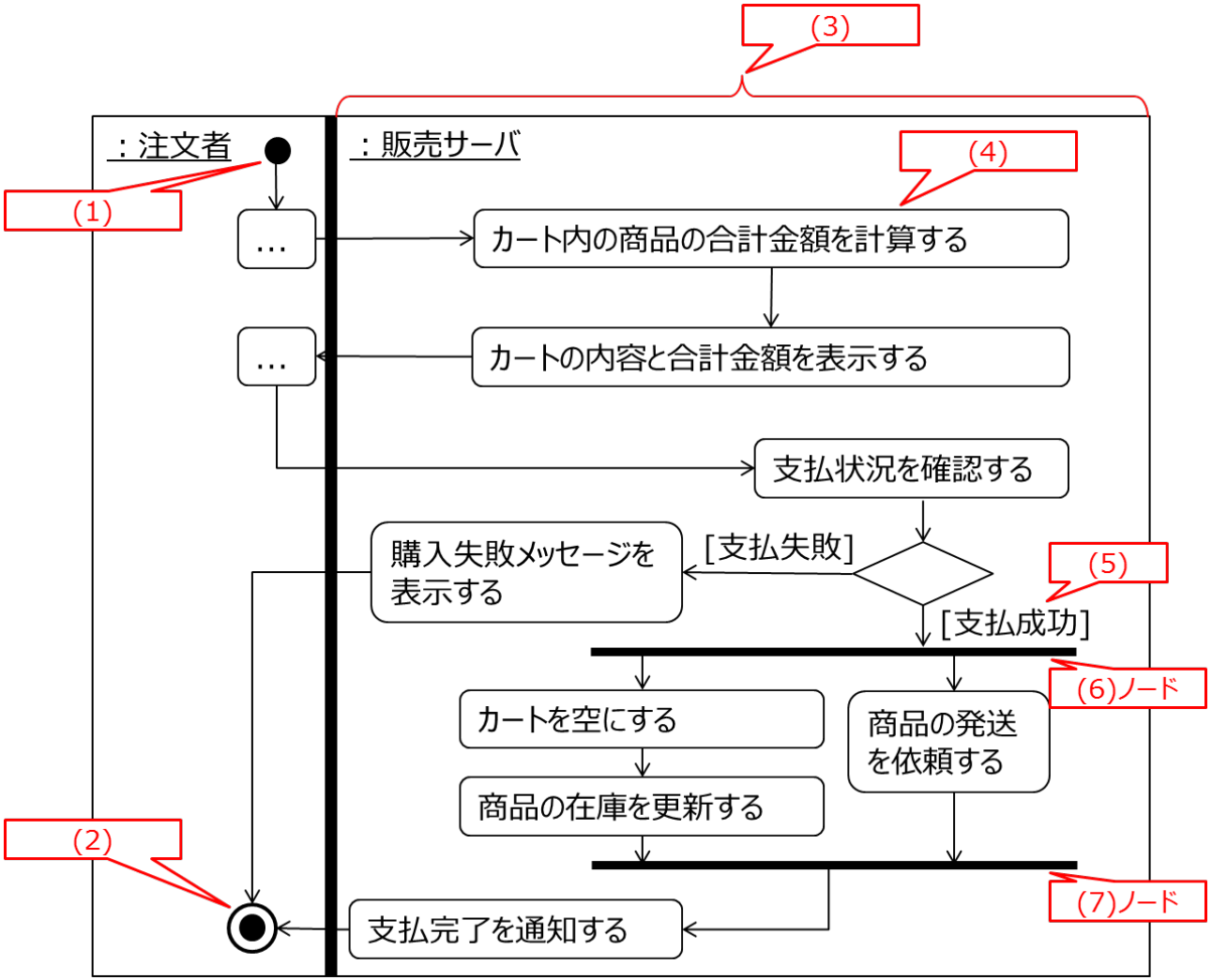


アクティビティ図の注意点

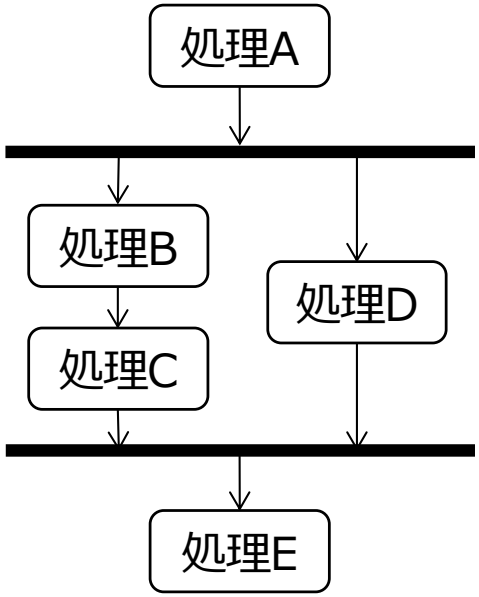
- 登場するクラス、オブジェクトの
  すべての振る舞いを記述するわけではない
  - 特定の処理に注目したときの振る舞いを記述
    (ユースケース図やシーケンス図でも同様)
- 極端に異なる粒度の処理を混在させない
  - 概念的なアクションと詳細なアクション
- 多くの要素を詰め込みすぎない
  - 見やすさ分かりやすさを意識

確認問題

■ アクティビティ図の各部の名称を答えよ。

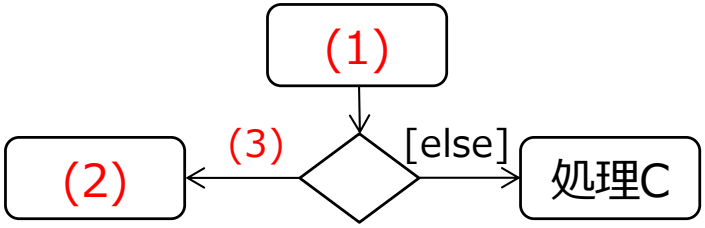


確認問題



左のアクティビティ図(一部)があるとき、
以下の各文は正しいか。○か×で答えよ。

- 処理Cと処理Dは同時に終了しなければならない。
- 処理Eの前に処理Cと処理Dは終了しなければならない。
- 処理Bの終了前に処理Dは開始して良い。
- 処理Bの終了前に処理Cは開始して良い。



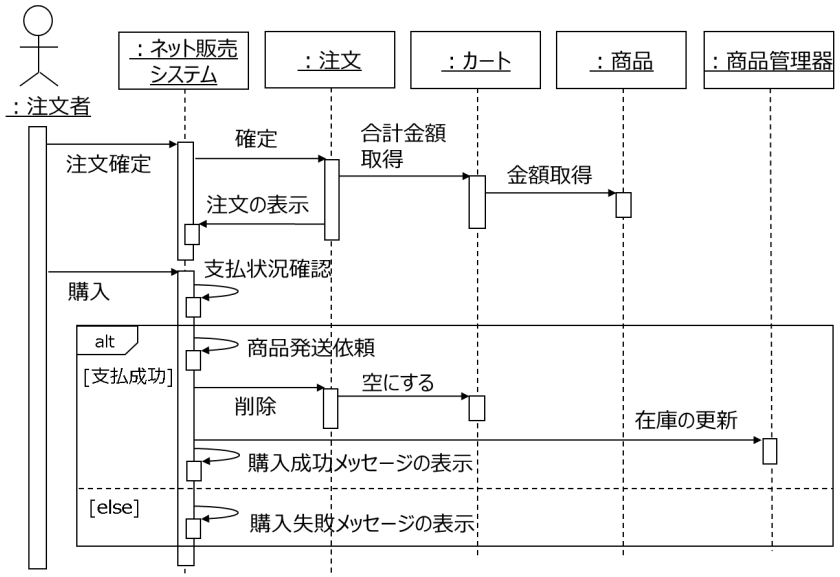
左のアクティビティ図(一部)が以下の
各文をすべて満たすように空欄を埋めよ。

- 処理Aの後、条件判定を行う。
- a>0のとき、処理Bを実行。
- そうでないとき、処理Cを実行。

シーケンス図

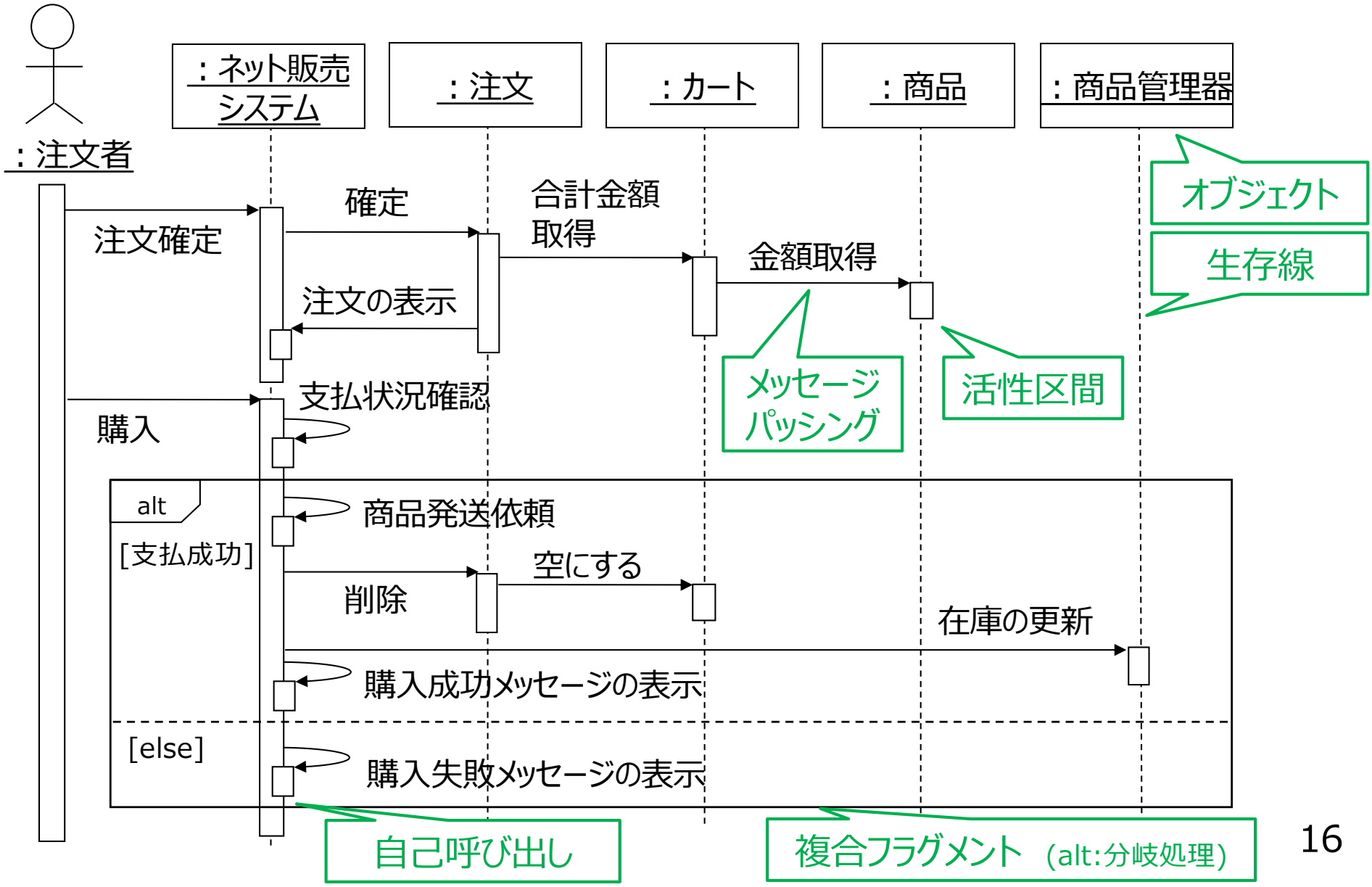
■ オブジェクト間の相互作用
(メッセージのやり取り)を時系列に沿って表現

- ●特定の機能に関するメッセージの流れを表現
- ●メッセージの流れが直感的
  - ■メッセージの順番や送受信の対象が明確
  - ■メッセージは直接メソッド呼び出しに対応可能

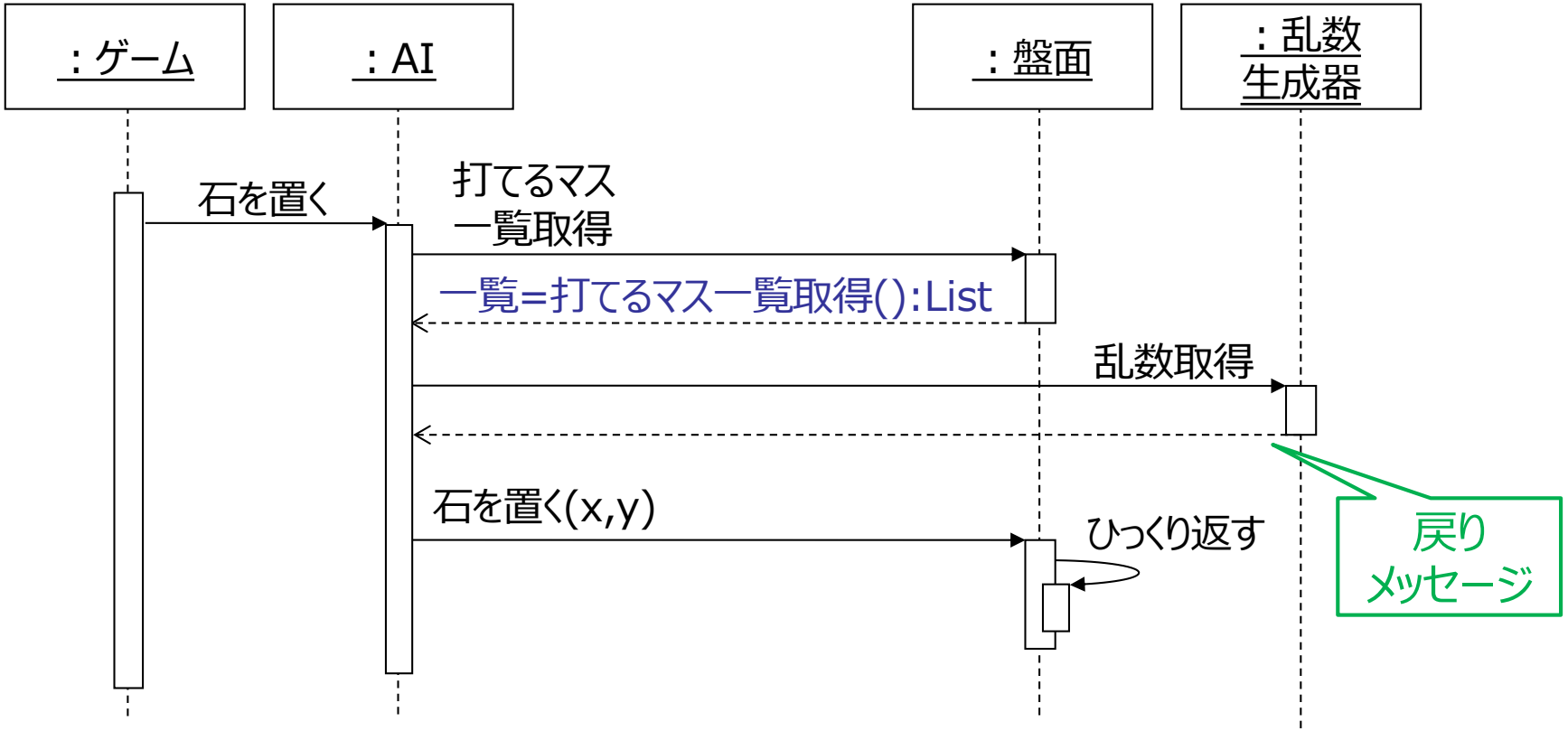


(例) シーケンス図

オブジェクト間のメッセージ
送受信の時系列



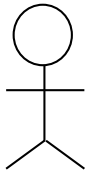
(例) シーケンス図



・引数や戻り値は省略可能
(乱数取得→戻り値は「乱数」であることが自明なので省略)

シーケンス図の構成要素

- アクタ・オブジェクト：
  メッセージやり取りの主体

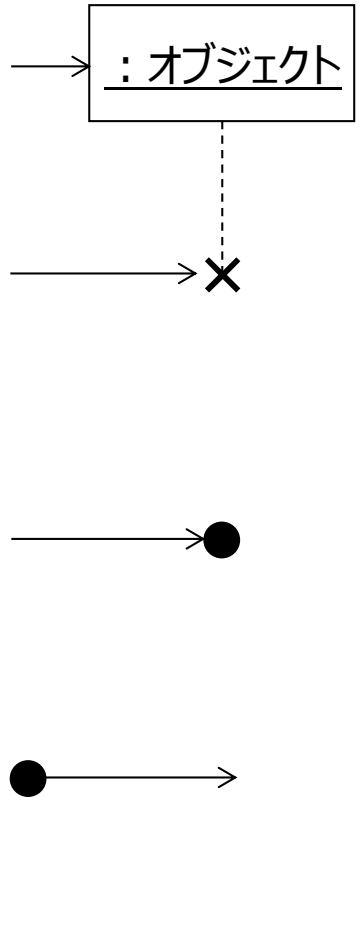


※UML2.xではシーケンス図の
オブジェクトに下線は付けない

- 生存線(lifeline)：
  各アクタ・オブジェクトから伸ばした線。
  活性区間(処理を行っている時間)は太く(矩形で)示す
- メッセージパッシング：
  オブジェクト間でやり取りされるメッセージ
  (同期 ⟶　非同期 ⟶　)
  メッセージの名前を必ず指定する
- 戻りメッセージ：　⇠------
  メッセージによって起動された操作の終了。
  割当先 = 操作名(引数):戻り値 という記述で内容説明
  (単に戻り値を書く略記法を採用することが多い)

シーケンス図の構成要素

- 生成メッセージ(create message)：
  オブジェクトの生成を示す
- 破棄メッセージ(destroy message)、
  破棄イベント：
  オブジェクトの破棄を示す
- 消失メッセージ(lost message)：
  メッセージの送り先が
  図の範囲外等のため存在しない
- 出現メッセージ(found message)：
  メッセージの送り元が
  図の範囲外等のため存在しない

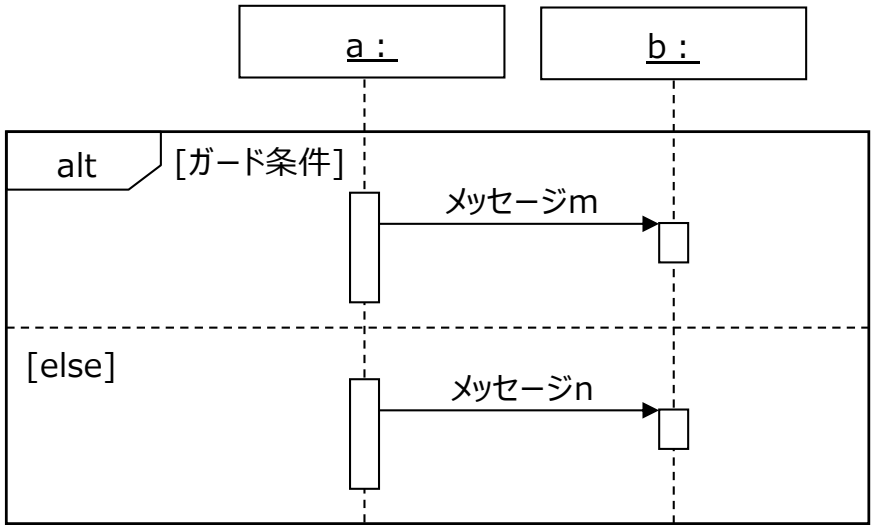


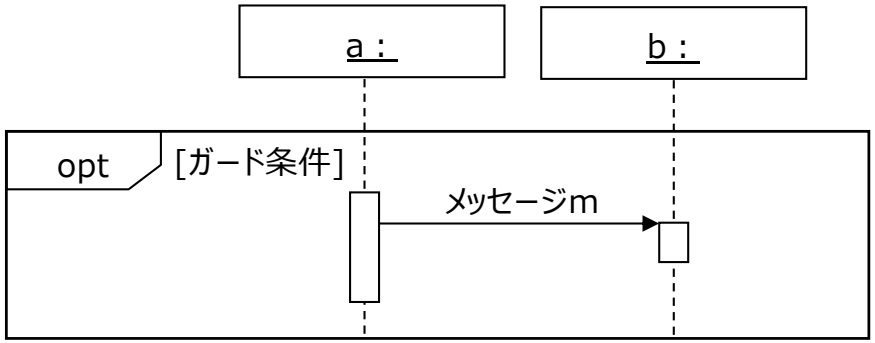
シーケンス図の構成要素

- 複合フラグメント(combined fragment)：相互作用の一部に特別な意味を付加
  - alt ガード条件により分岐
  - opt ガード条件を満たす場合のみ実行
  - loop 繰り返し
    - loop(1,10) 繰り返し回数の指定(別途ガード条件記述可)
  - break 実行後、親フラグメントの実行を中断
  - par 並列動作(内部を水平線で複数の領域に分割)
  - critical 重要な処理であり、他からの割り込みを受けない
  - seq(弱シーケンス) 異なるライフラインの処理の順番を入替可
  - strict(強シーケンス) 処理の順番を厳密に守る必要がある
  - ignore(無効) 領域内処理中、発生しても無視可能なメッセージを記述
  - consider(有効) 領域内処理中、発生を考慮する必要があるメッセージを記述(それ以外のメッセージにignoreを指定したのと同意)
  - neg(否定) 正常に動作しない可能性がある処理(例外処理が必要等)
  - assert 唯一の妥当な実行パスであることを示す

シーケンス図の構成要素 - 複合フラグメント

- alt:
  ガード条件により分岐
- opt:
  ガード条件を満たす
  場合のみ実行

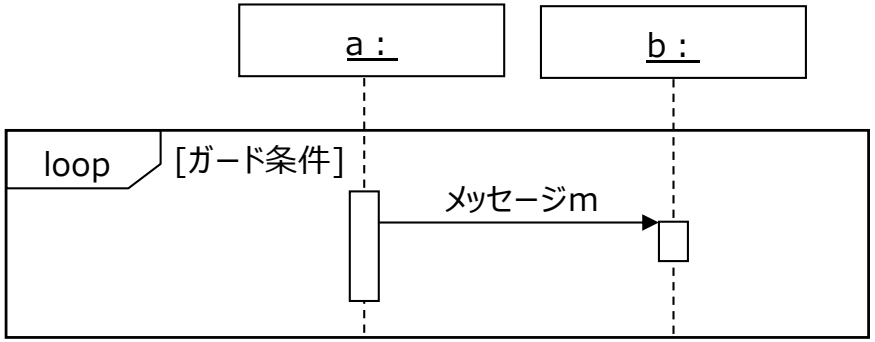




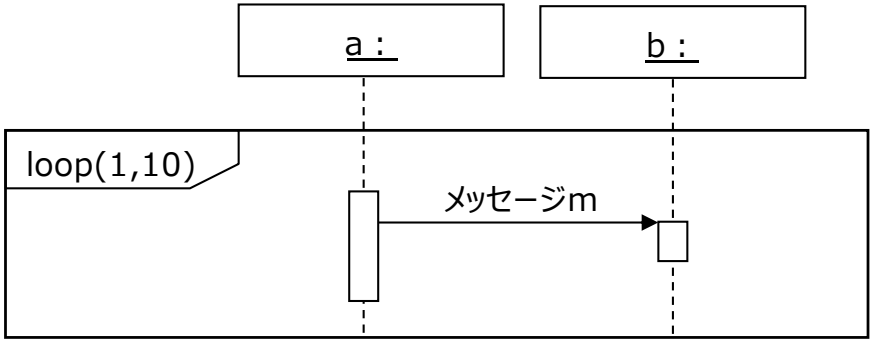
シーケンス図の構成要素 - 複合フラグメント

- loop：繰り返し

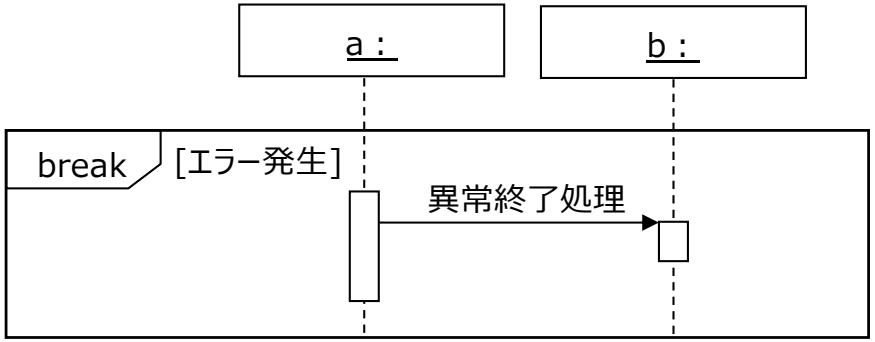
条件が満たれている間繰り返し



最低1回、最高10回繰り返し



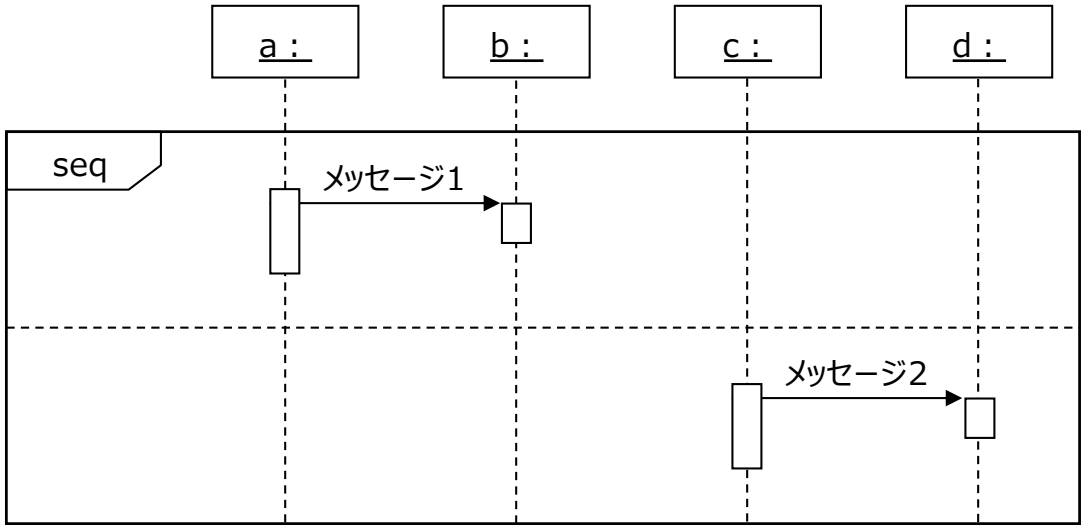
- break：実行後、親フラグメントの実行を中断



他の複合フラグメントの中にない
場合は、その図の実行を中断

シーケンス図の構成要素 - 複合フラグメント

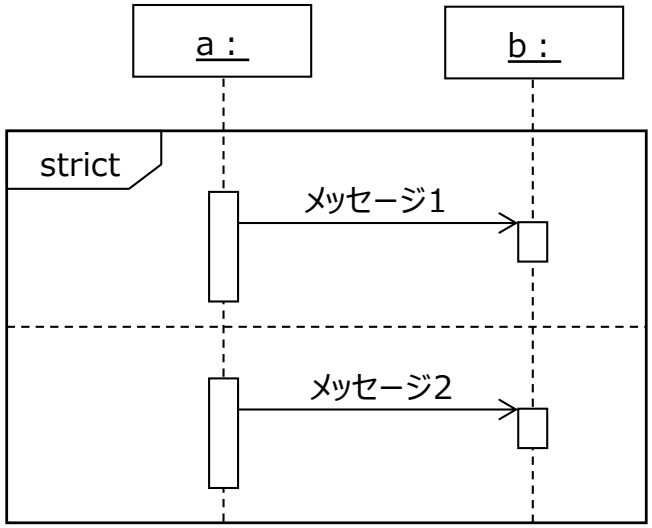
■ seq：弱シーケンス



・各領域内のイベントの順序は維持
・異なる領域内のイベントが異なるライフライン上で発生する場合は任意の順序でOK
・異なる領域内のイベントが同じライフライン上で発生する場合は順序を維持

e.g., <1,2>、<2,1>ともに可能

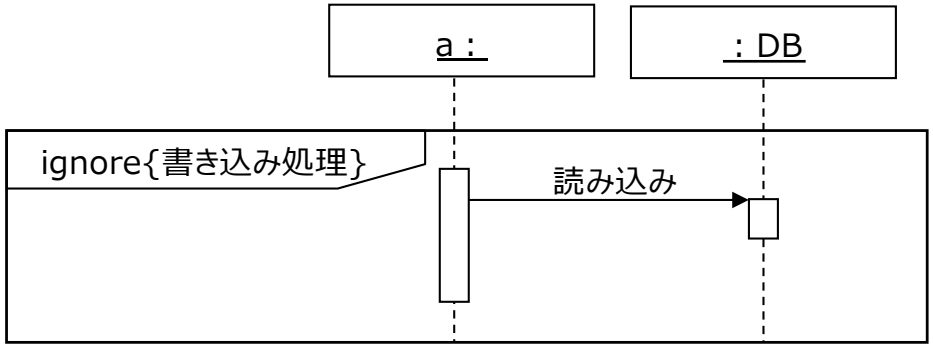
■ strict：強シーケンス



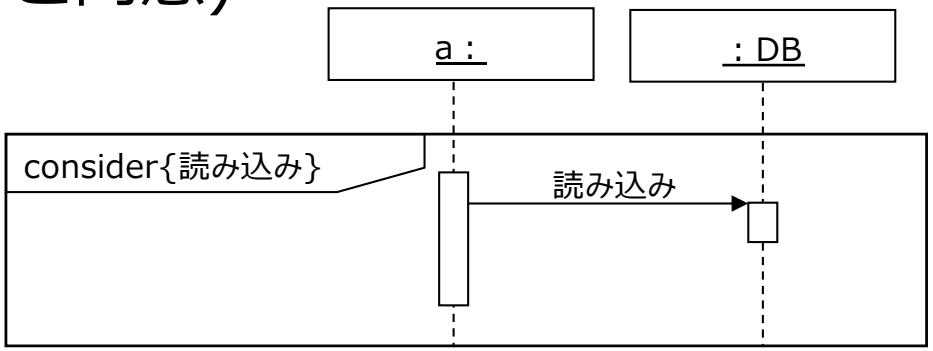
・フラグメント内のすべてのイベントの順序は維持 (非同期メッセージも含めて順序を明示可能)
e.g., <2,1>は不可能
・フラグメント内を領域に分けても分けなくても順序制約は変わらない

シーケンス図の構成要素 - 複合フラグメント

- ignore：無効(無効) 領域内処理中、発生しても無視可能なメッセージを記述

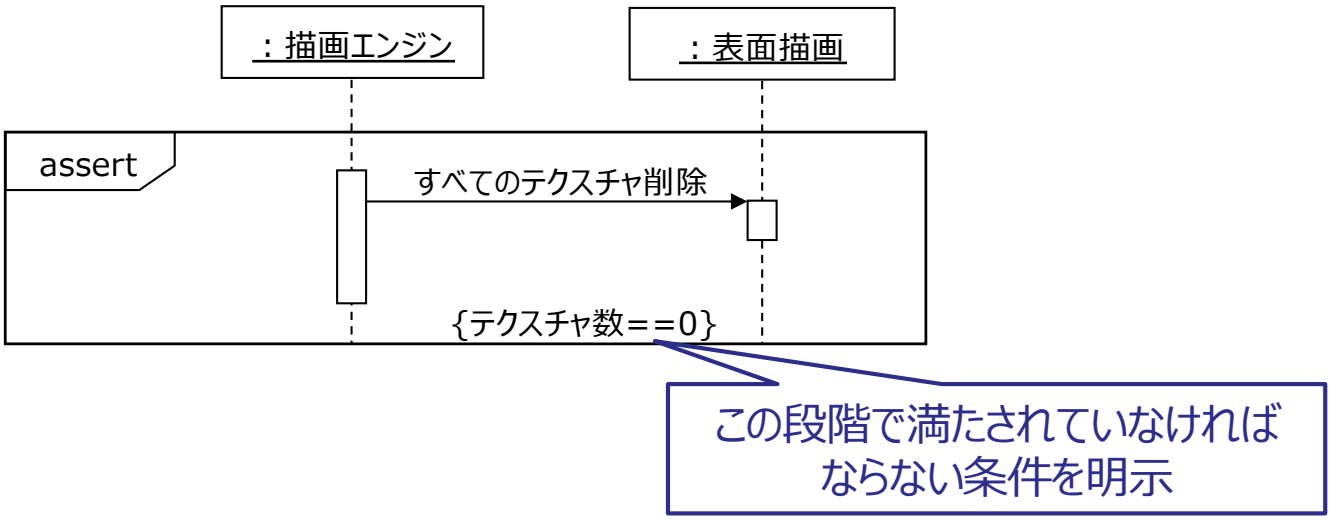


- consider(有効) 領域内処理中、発生を考慮する必要があるメッセージを記述(それ以外のメッセージにignoreを指定したのと同意)



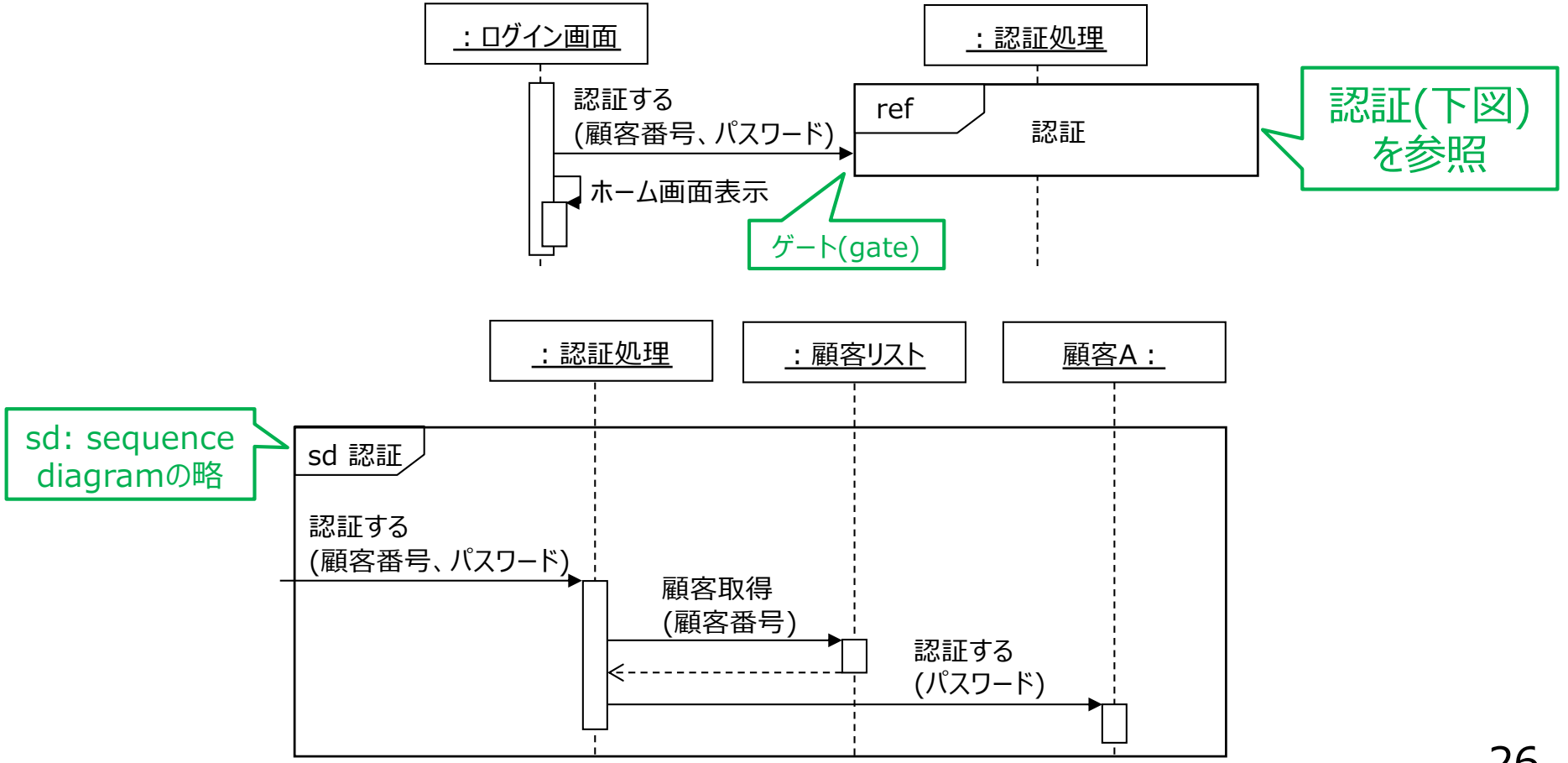
シーケンス図の構成要素 - 複合フラグメント

■ assert：唯一の妥当な実行パスであることを示す
(実行時に満たされなければならない条件も
記述できる)



シーケンス図の構成要素

■ フレーム(frame)：
図の一部を外部に切り出す

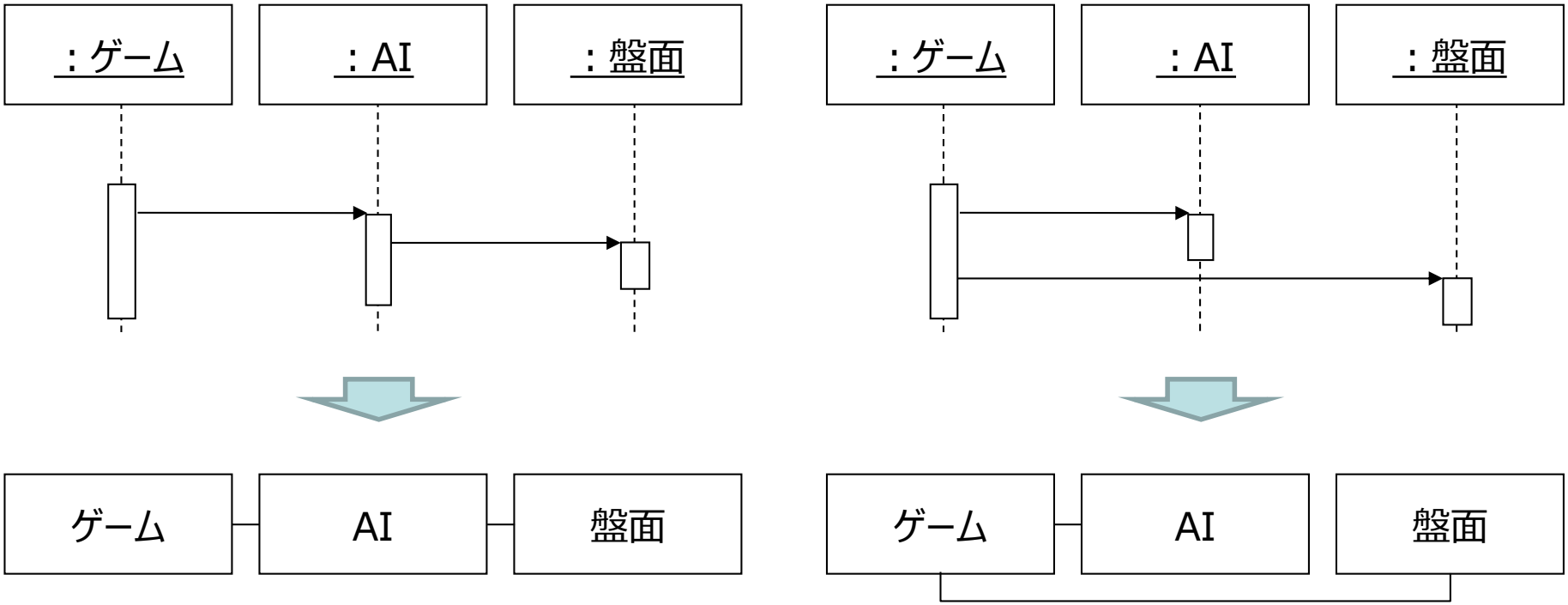


シーケンス図の注意点

- 振る舞いの詳細までは示しにくい
  - e.g., メソッドの中でどのような処理をするか
- 機能ごとに図を作らなければならない
- 他の図との整合性
  - e.g., クラス名は合っているか？
    関連は？(→次P参照)
- シーケンス図の粒度(細かさ)も様々

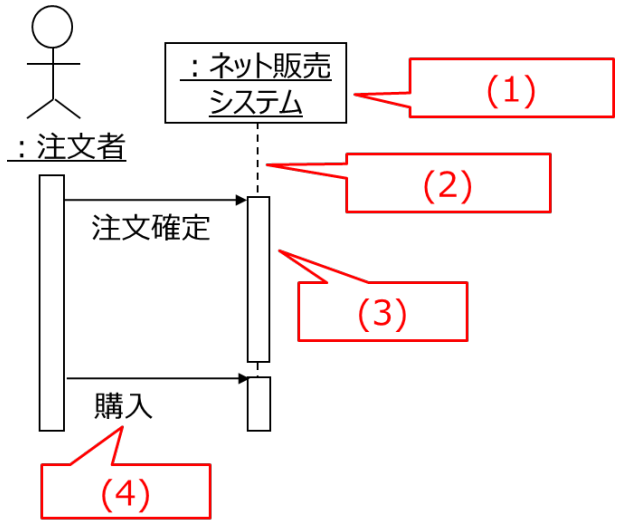
シーケンス図とクラス図の関係

■ メッセージパッシングは関連として表れる



確認問題

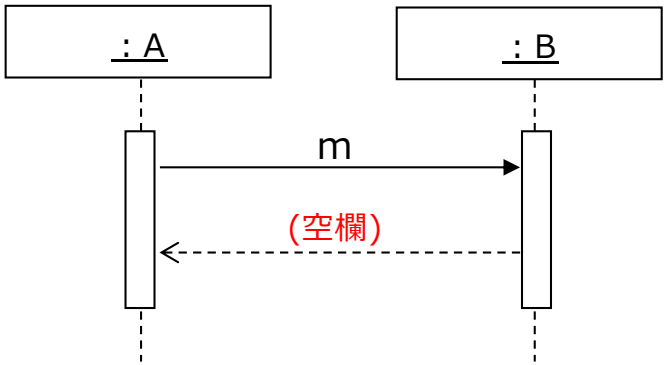
■ シーケンス図の各部の
名称を答えよ。



- 以下の各文は正しいか。○か×で答えよ。
  - シーケンス図中では、実際に発生するメッセージのやり取りは省略してはならない。
  - シーケンス図では、インスタンス生成が行われることを生成メッセージとして表現できる。
  - シーケンス図中のメッセージパッシングは必ず右向きの矢印として示す。

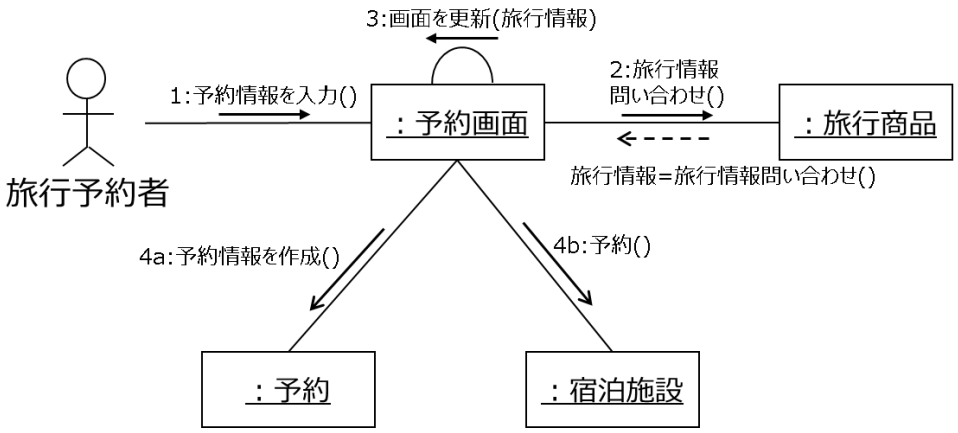
確認問題

- 以下の説明に最も合うシーケンス図の複合フラグメントの種類を答えよ。
  - ガード条件により処理を分岐させる。
  - ガード条件を満たす場合のみ実行する。
  - 繰り返しを行う。
  - 並列動作を行う。
- A型のオブジェクトからB型のオブジェクトに対してmというメッセージパッシング(引数なし)が発生し、そのint型の戻り値がxという変数に割り当てられるものとする。このことを示すように図中の空欄を埋めよ。

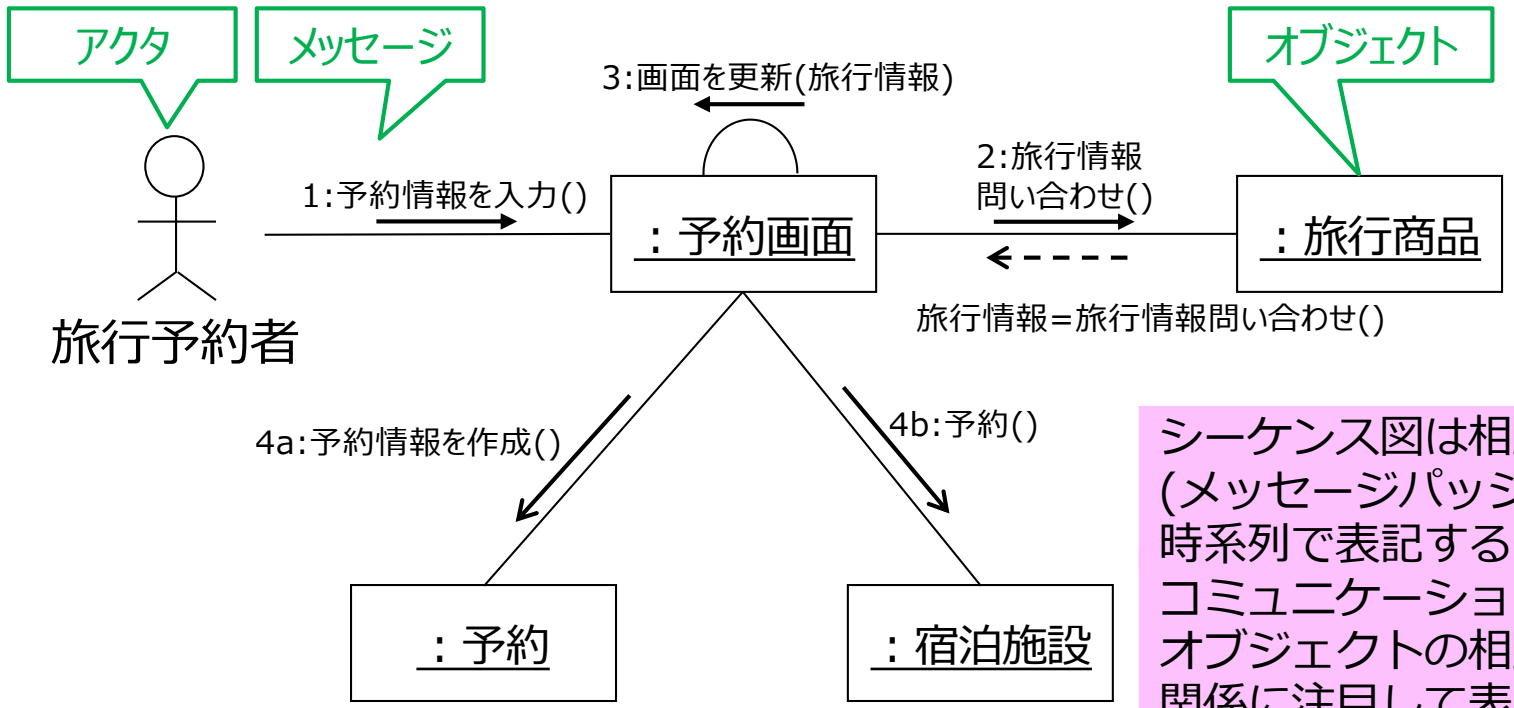


コミュニケーション図

- オブジェクト間の相互作用(メッセージのやり取り)をオブジェクト間の結び付きに注目して表現
  - メッセージの集中を確認するのに適している
- 構成要素
  - アクタ、オブジェクトはシーケンス図と同様
  - メッセージはオブジェクト間のリンクに説明を付加する形式で表現
    - ガード条件付加可能
  - フレームも利用可能



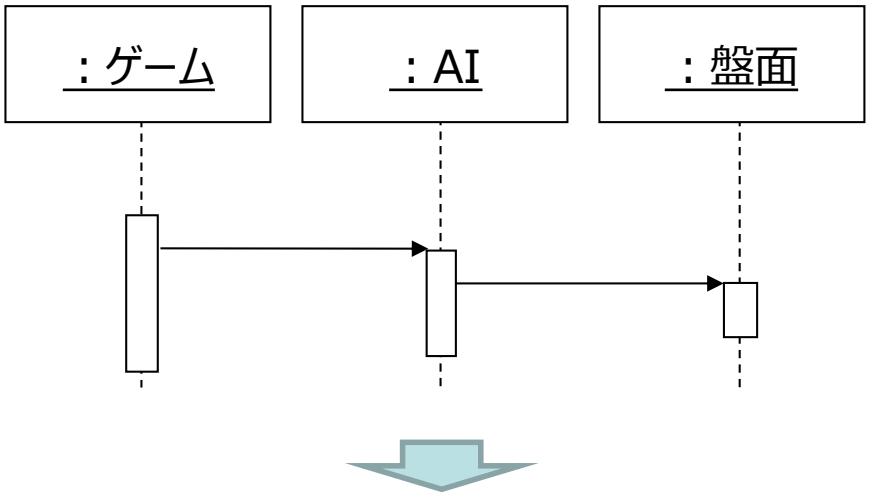
(例) コミュニケーション図

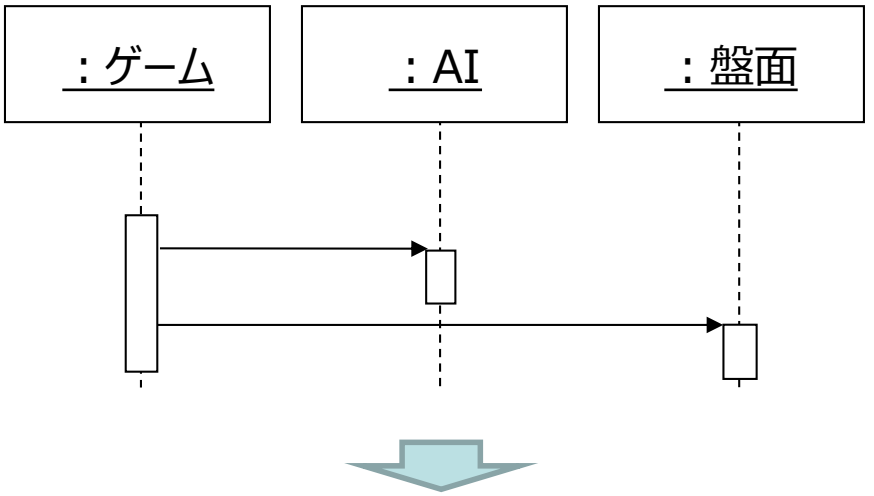


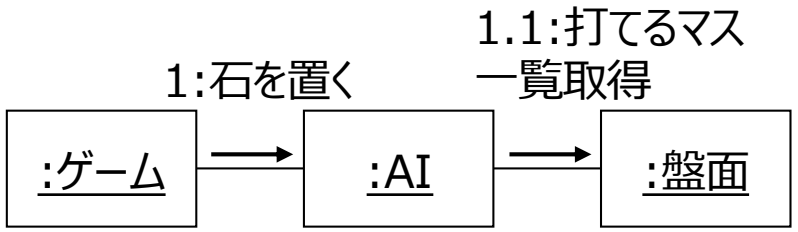
シーケンス図は相互作用(メッセージパッシング)を時系列で表記する
コミュニケーション図はオブジェクトの相互作用や関係に注目して表記する

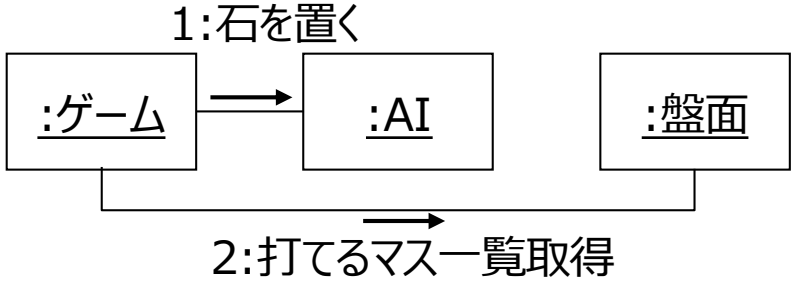
- 引数や戻り値は省略可能
- メッセージには説明と向きを示す矢印を付ける　⟶ 同期　⟶非同期 <⋯⋯ 戻りメッセージ
  メッセージの順番を示す番号を付加してもよい(非必須)
- 番号で処理の構造を表現
  e.g., 1の処理から呼ばれる処理を1.1, 1.2, ...とする (1.1.1, 1.1.1.1, ...も可能)

コミュニケーション図とシーケンス図









確認問題

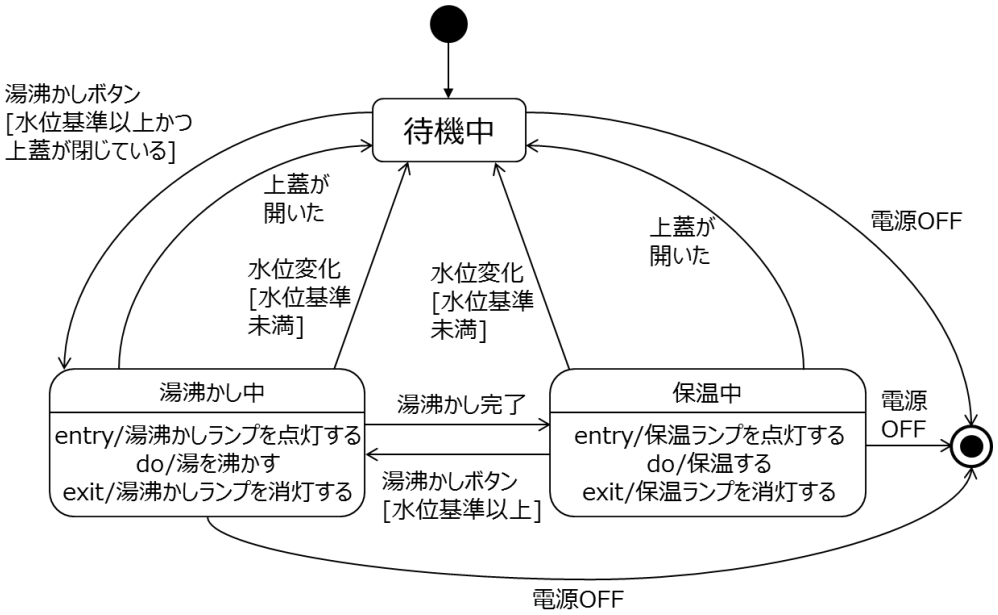
■以下の各文は正しいか。○か×で答えよ。

- コミュニケーション図はオブジェクト間の相互作用を表現する図である。
- コミュニケーション図よりもシーケンス図の方がメッセージパッシングの順番を把握するのに適している。
- コミュニケーション図ではメッセージパッシングの順番は表現できない。

状態機械図(state machine diagram)

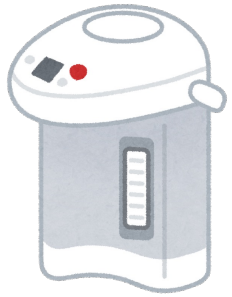
■特定のオブジェクトやシステムの
状態遷移を視覚的に示す

- ●状態、遷移、イベント(トリガ)により表現
- ●1つの状態が内部状態を含むこともできる



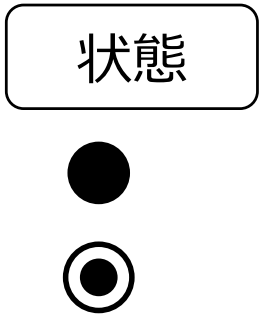
(例) 状態機械図

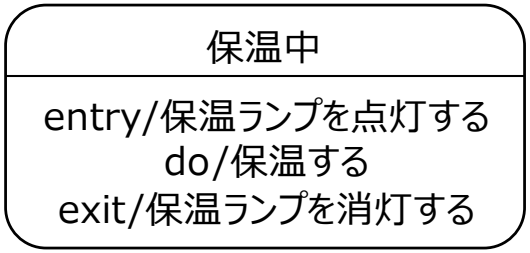
オブジェクトの状態と
イベントによる状態遷移



状態機械図の構成要素

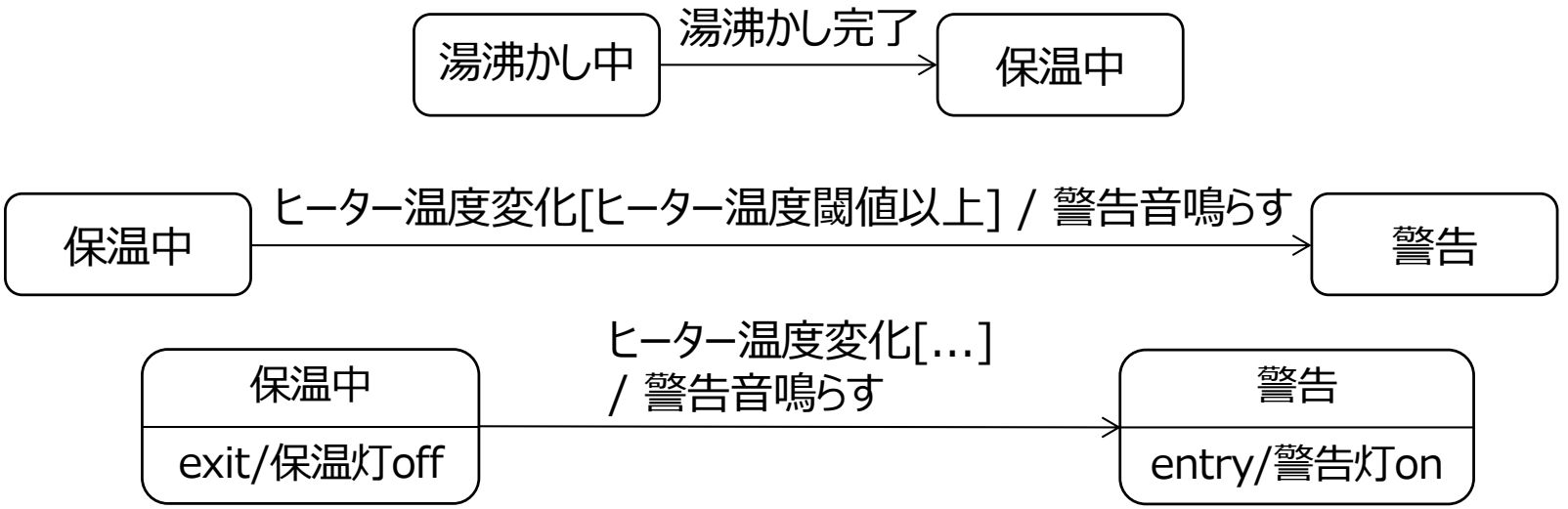
- 状態(単純状態)：オブジェクトの状態を表現
- 開始擬似状態：
  オブジェクト生成直後(遷移前)の状態
- 終了状態：
  遷移終了後(オブジェクト破棄時)の状態
- entryアクション：状態に遷移したときに
  一度だけ実行される処理
- exitアクション：状態から離れるときに
  一度だけ実行される処理
- doアクティビティ：その状態の間、
  継続して実行される処理
- 内部遷移：1つの状態の中で変化が
  あったときの処理
  (トリガ、ガード、エフェクト指定)





状態機械図の構成要素

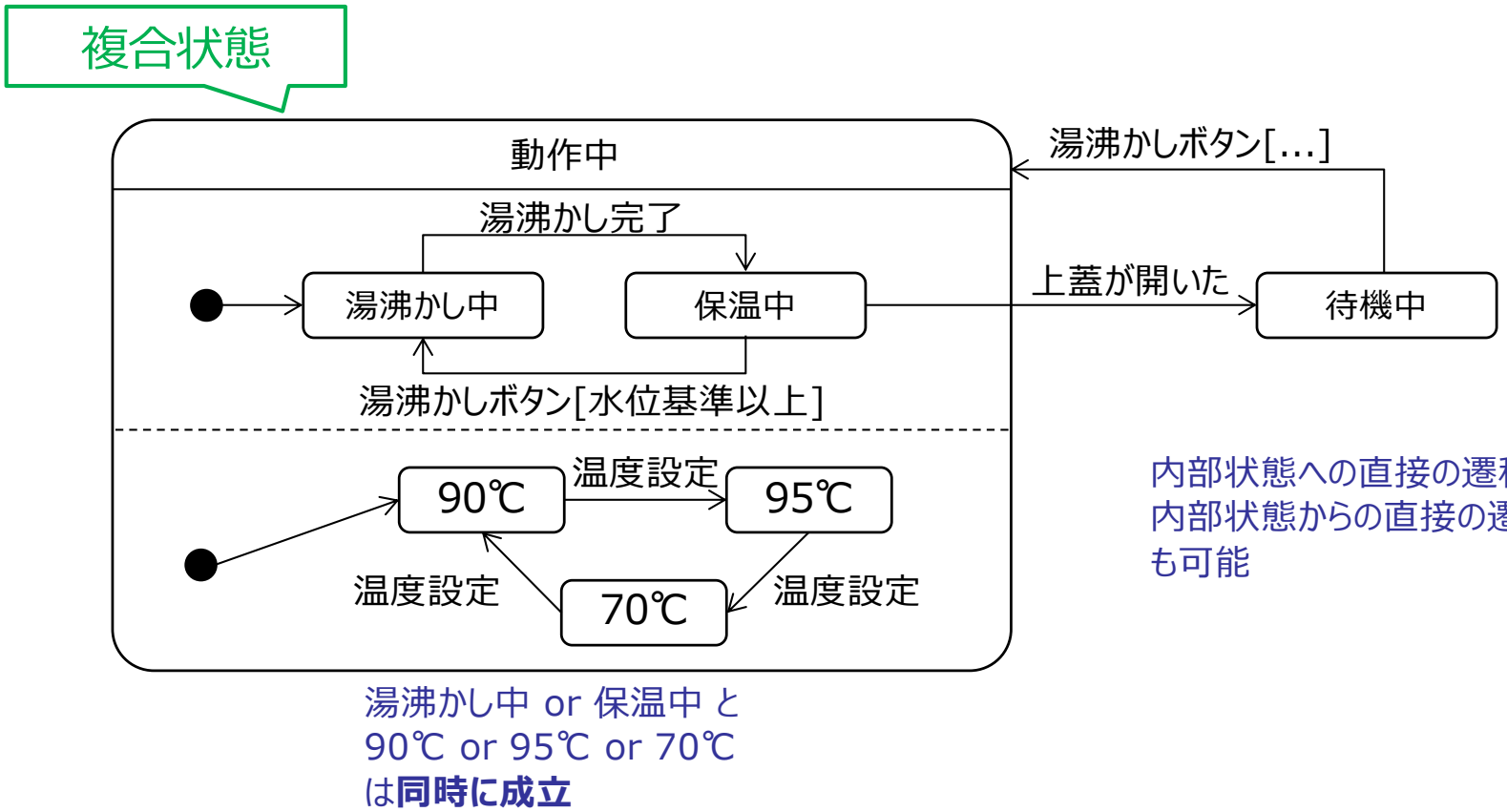
- 遷移
  - 書式： トリガ[ガード]/エフェクト
    - トリガ：遷移のきっかけとなるイベント(事象)
    - ガード：遷移の条件
    - エフェクト：遷移に伴って実行される処理



状態機械図の構成要素

■直交状態

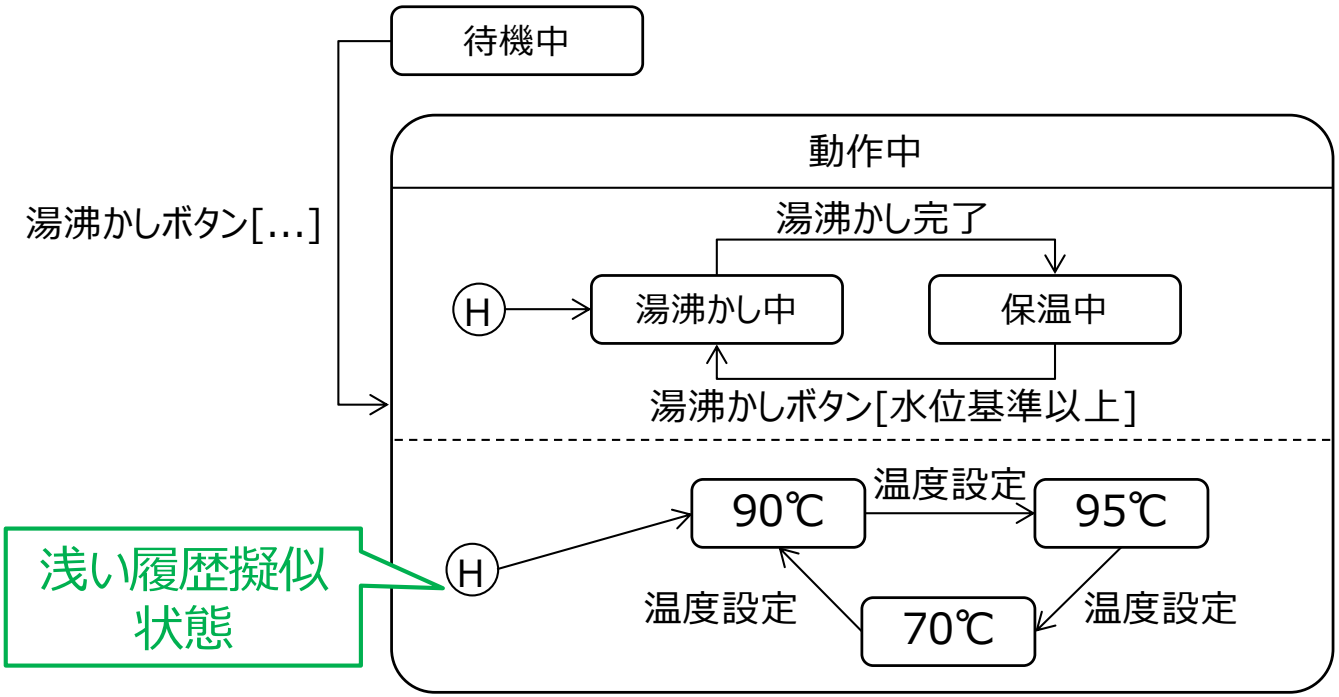
- ●同時に複数の状態を取る場合に使用
- ●状態の内部を複数の領域に分割して表現



状態機械図の構成要素

■擬似状態

- ●状態遷移を制御するための擬似的な状態
- ●浅い履歴擬似状態 (shallow history pseudostate)



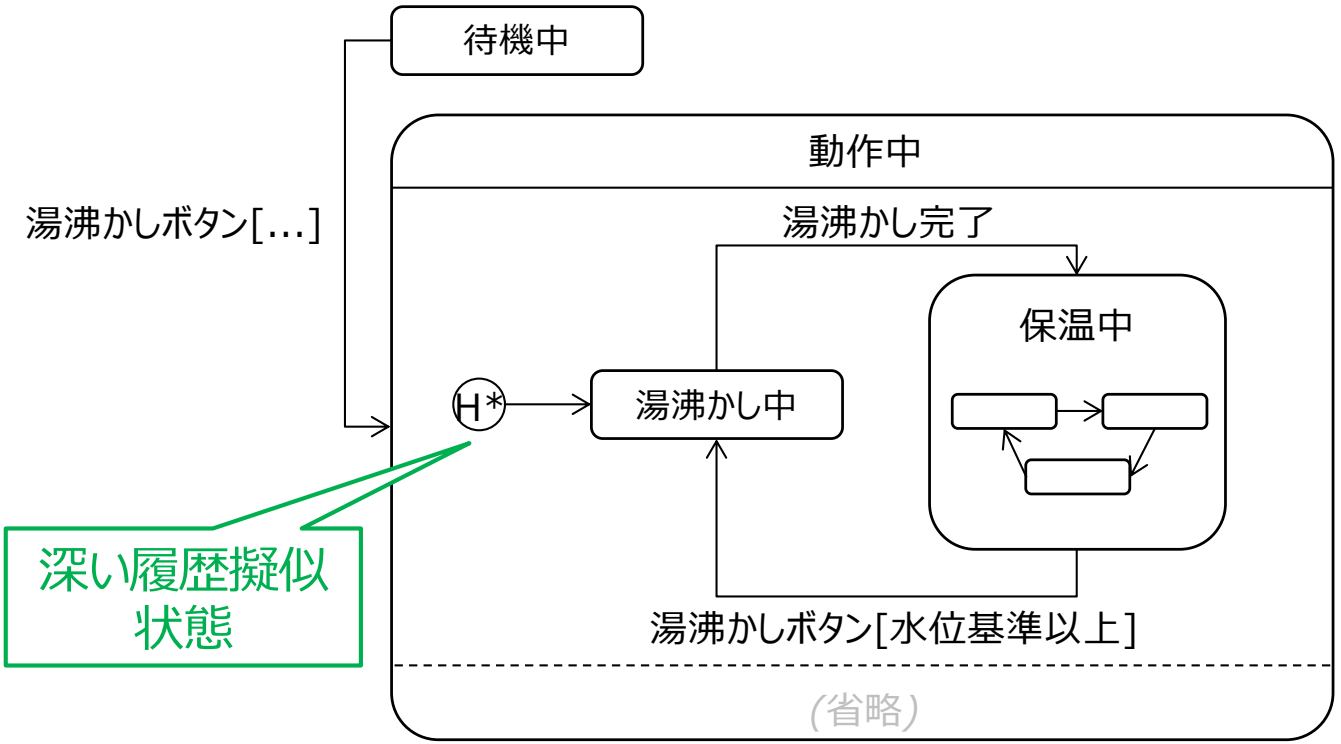
動作中に遷移すると、
内部状態は以前の状態に戻る
(初めての場合や
終了状態の場合は
デフォルトの遷移をする)

履歴擬似状態は複合状態内の
1つの領域に1つのみ

状態機械図の構成要素

■擬似状態

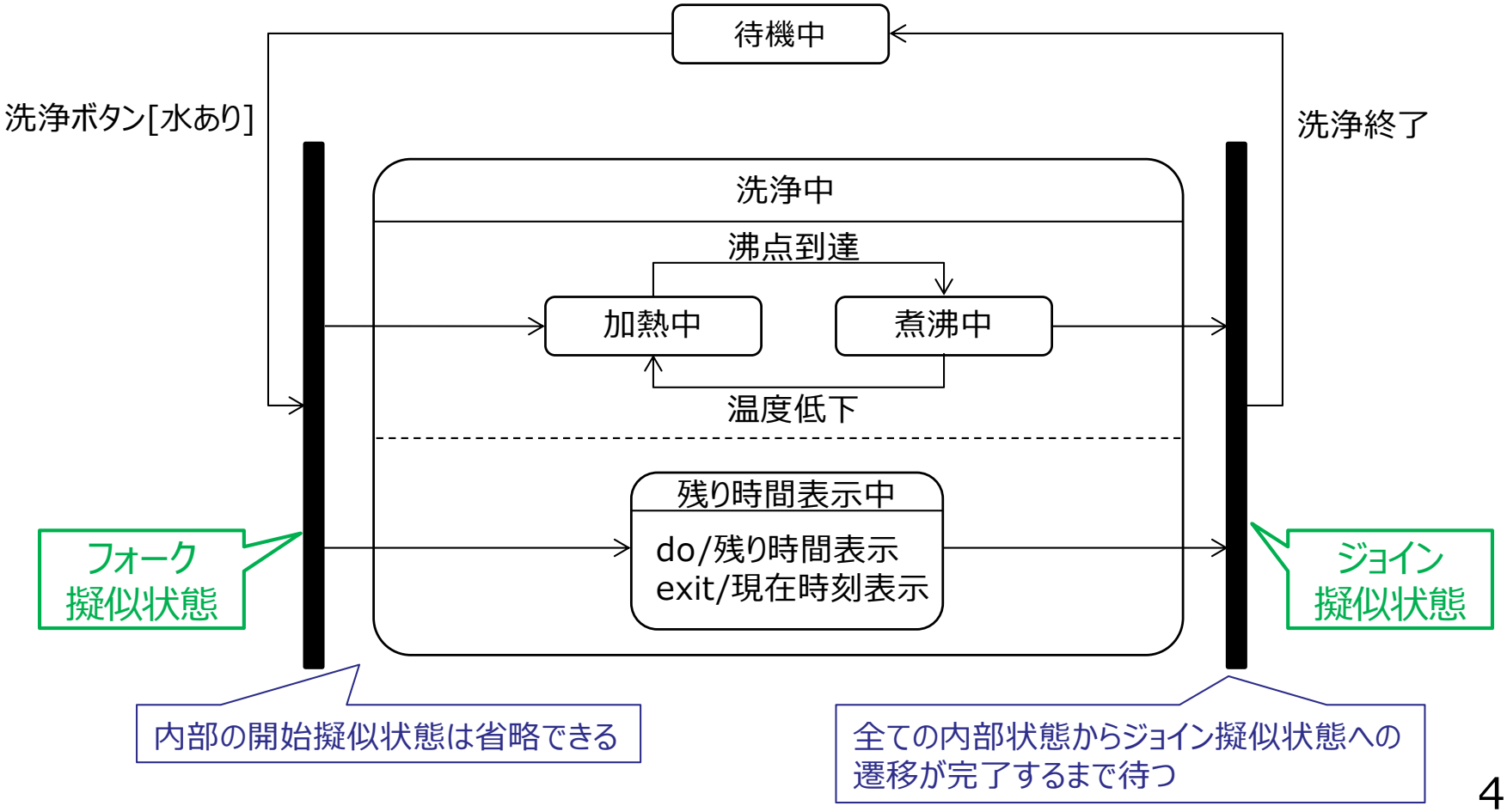
●深い履歴擬似状態 (deep history pseudostate)



深い履歴擬似状態を
使えば、入れ子になっている
状態遷移も記憶可能

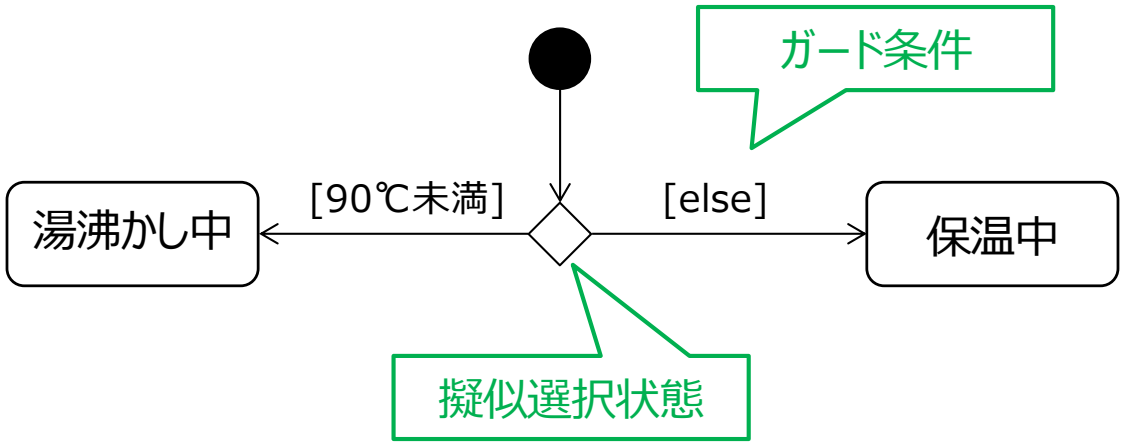
状態機械図の構成要素

- フォーク擬似状態：直交状態に並行的に遷移
- ジョイン擬似状態：フォークした遷移の統合

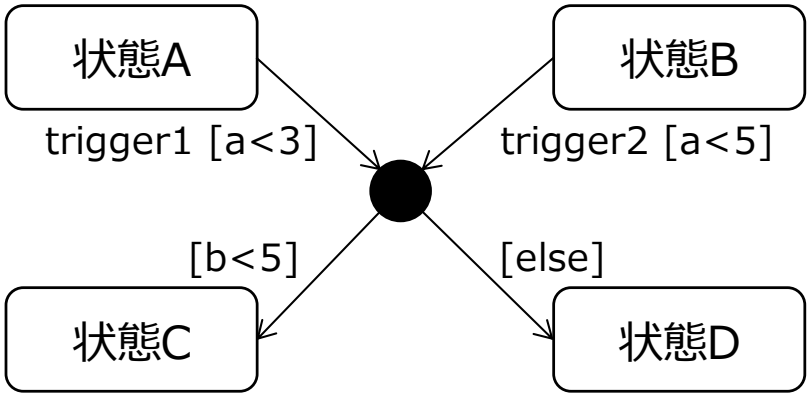


状態機械図の構成要素

- 擬似選択状態：その時の状況に応じて分岐を行う(動的条件分岐)



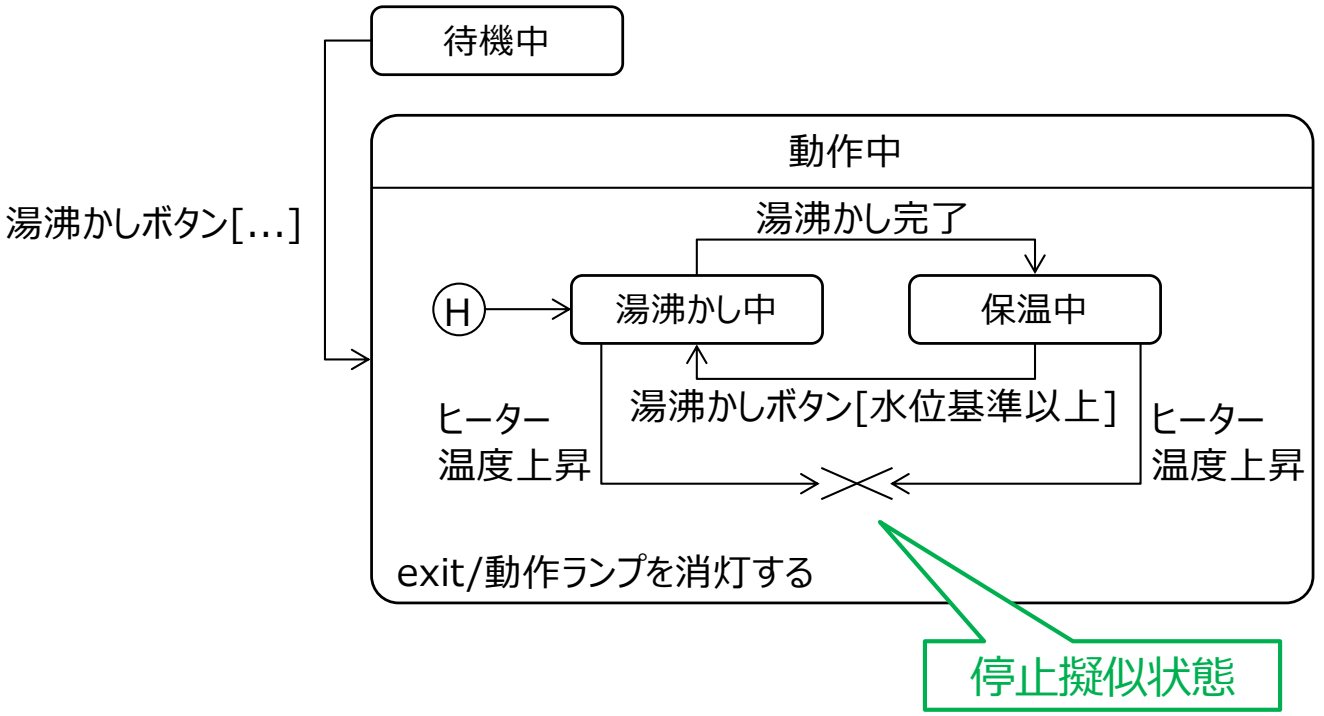
- ジャンクション擬似状態：事前に評価した結果によって遷移先を分岐させる(静的条件分岐)



bに関する評価はtrigger1,2発生時に終わっている

状態機械図の構成要素

- 停止擬似状態：すべての処理を停止してオブジェクトを破棄する



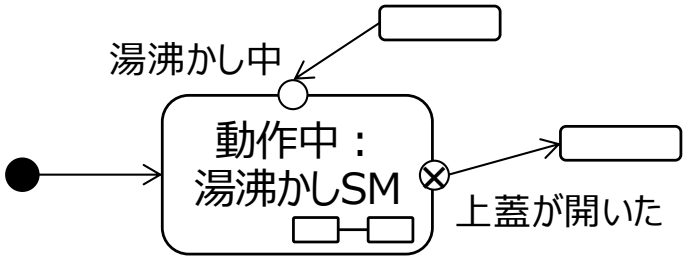
停止擬似状態で終了後、親状態のexitアクションは実行しない

実行途中のその他の処理も中断する

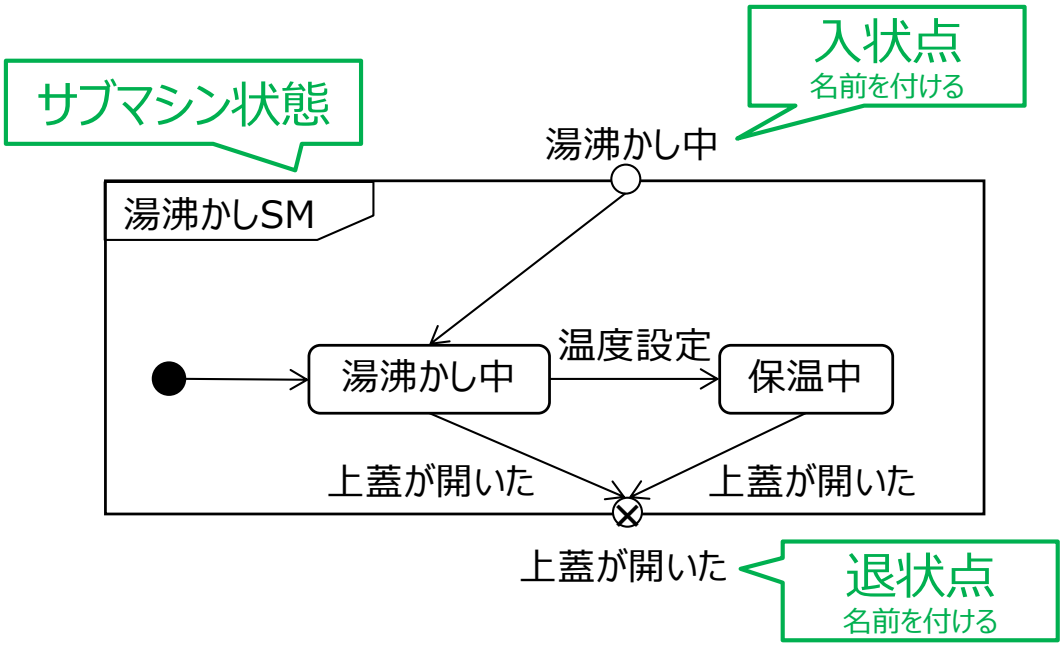
状態機械図の構成要素

■ サブマシン状態

- 状態機械図の部品を定義



←サブマシン状態、入状点、退状点の参照



状態機械図の注意点

■一度に目にする状態数を減らす

- ●複雑な状態図は理解しにくい
- ●状態遷移の漏れも発生しやすい

UMLの利点が失われる

■**意味のない状態**はないか？

- ●すべての状態において振る舞いがあるか確かめる(擬似状態、終了状態除く)

■**処理手順を書かない**

→アクティビティ図やシーケンス図を使用

■必要に応じて状態遷移表を併用

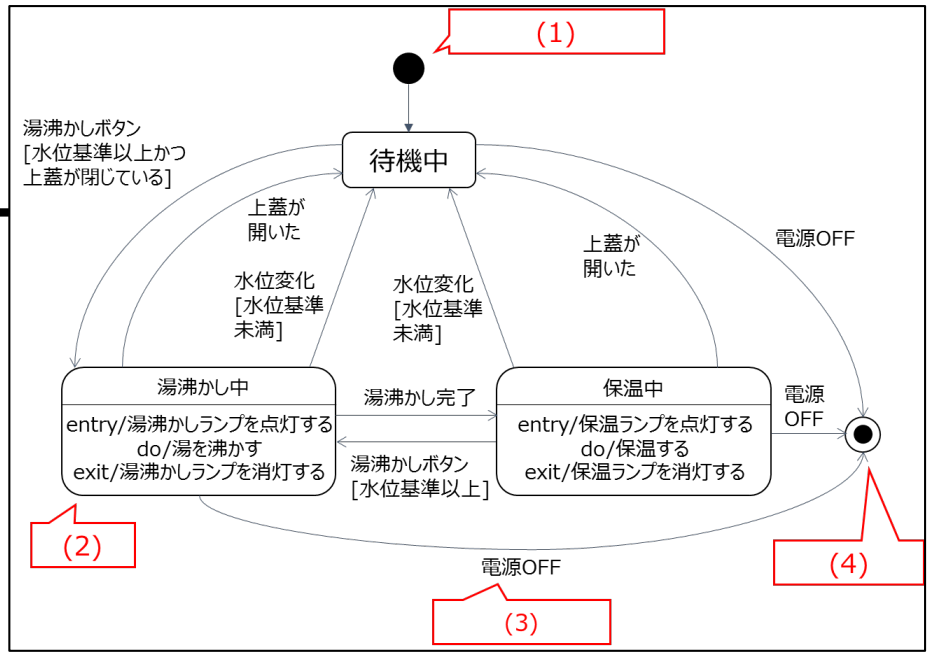
- ●状態遷移の検討漏れを防止

モデリングの注意点

■各図の整合性

- 例えば、クラス図のクラスや関連とシーケンス図のオブジェクト、メッセージパッシングが整合しているか
- 各図の抽象度によっては、一部を省略することがある(実装に近い図では詳細を描き込むが、ビジネスフローでは詳細は略す)

確認問題



- 状態機械図の各部の名称を答えよ。

- 以下の各文は正しいか。○か×で答えよ。
  - doアクティビティは、その状態の間に一度だけ実行される処理である。
  - 状態遷移のトリガには必ずガード条件を記述する。
  - 状態が内部状態の遷移図を含むとき、外部から内部状態へ直接遷移することが可能である。
  - 浅い履歴擬似状態を使用しても、同一領域内に記憶できない状態が存在する可能性がある。
  - 停止擬似状態に遷移すると、その階層での状態遷移を停止し、即座に親状態の遷移に移行する。
  - オブジェクト指向設計で複数種類のUMLを使用するとき、各図の整合性が重要である。
  - オブジェクト指向設計で複数種類のUMLを使用するとき、各図の粒度を合わせることが必要である。