「標準/汎用SoCプラットフォーム」を目指すダイナミックリコンフィギュラブルプロセッサの最新動向

村上, 和彰
九州大学大学院システム情報科学研究院 | 九州大学情報基盤センター

出版情報：SLRC プレゼンテーション, 2005-06-29
バージョン：
権利関係：

「標準／汎用SoCプラットフォーム」を目指すダイナミックリコンフィギュラブルプロセッサの最新動向

村上和彰
九州大学
murakami@i.kyushu-u.ac.jp

概要と目次

- 概要：SoC（システムLSI）の設計生産性向上の切り札として期待されているダイナミック・リコンフィギュラブル・プロセッサについて、その最新動向、さらにそれをベースにした標準／汎用SoCプラットフォーム化の動きについて紹介する
- 目次：
  - SoC設計への諸アプローチ
    - カスタム・ロジックの実現方法
  - ダイナミック・リコンフィギュラブル・プロセッサの具体例
    - CLUSS Redefis

目次

- SoC設計への諸アプローチ
  - カスタム・ロジックの実現方法
    - プロセッサ＋ソフトウェア
    - コンフィギュラブル・プロセッサ＋ソフトウェア
    - リコンフィギュラブル・プロセッサ＋ソフトウェア
    - リコンフィギュラブル・ハードウェア
- ダイナミック・リコンフィギュラブル・プロセッサの具体例
  - CLUSS Redefis

Cost^3/Performanceに優れたSoC実現のための技術的課題

Performance
（性能）

- 計算処理の高速化
  - クロック周波数の高速化
  - 計算処理のハードウェア化，並列化
- メモリ・アクセスの高速化
  - メモリ自身の低レイテンシ化，高バンド幅化
  - メモリの分散配置の最適化
- オンチップ＆オフチップ通信の高速化
- 競合他社との差別化

Design Cost
（設計コスト）

- 短TAT（Turn-Around Time）化
  - 要求分析，仕様設計，システム設計，設計最適化，設計検証の各期間の短縮
  - 設計支援技術／ツールの向上
- 設計投資の高効率化
  - アーキテクチャの共通プラットフォーム化
  - 設計資産の標準化（IPコア活用）

Manufacturing&Testing Cost
（製造／テスト・コスト）

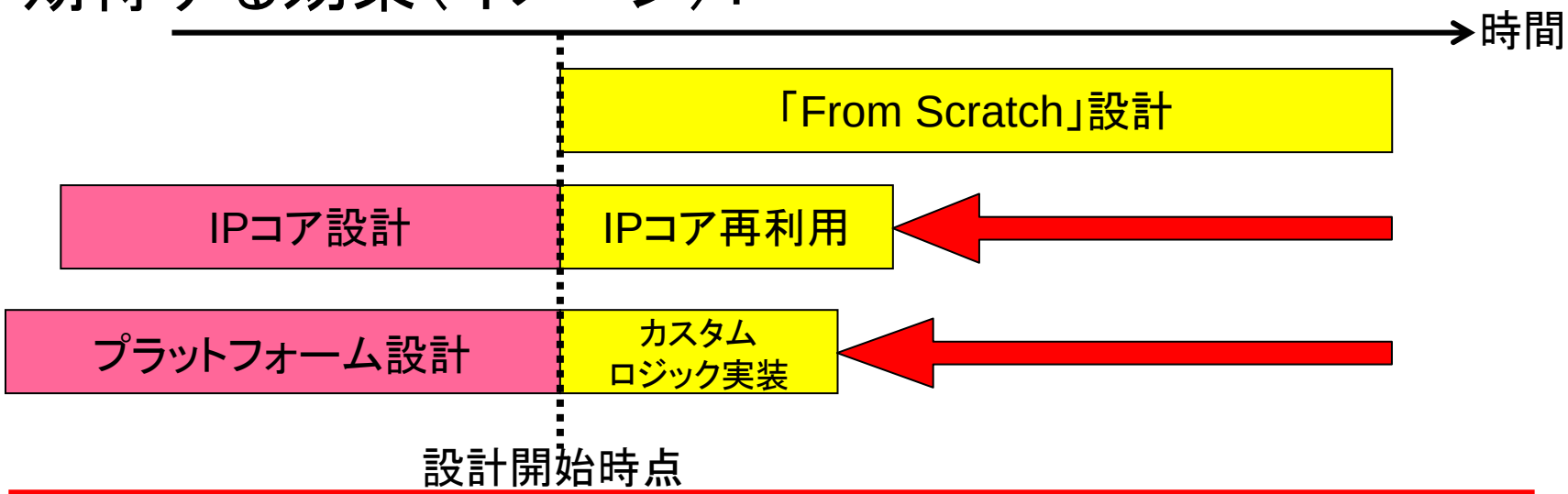
- チップ面積，マスク枚数削減
  - 性能とのトレードオフをどうとるか？
- 歩留まり向上
  - プロセス工程の「揺らぎ」にどう対処するか？
- テスト時間の削減
  - テスト容易化設計技術の向上
- SoCの本質である「多品種少量生産」問題の克服
  - SoCの汎用化，標準化？

Running Cost
（運用コスト）

- 低消費電力化
  - 性能とのトレードオフをどうとるか？
- 高信頼性化，高可用性化，高耐故障性化
  - オンライン・セルフチェッキング技術の向上
  - 製造コストとのトレードオフをどうとるか？
- SoC製品寿命の長命（延命？）化
  - 製品出荷後の仕様変更，バグフィックスをどう行うか？

プラットフォームとは？

- 異なる「カスタム・ロジック」を実現するのに利用可能な，「カスタム・ロジック」の違いに拠らない「共通の土台」
  - アナロジー：自動車の共通車台
- プラットフォーム・ベース設計：上記プラットフォーム上で「カスタム・ロジック」を設計する手法
  - 他の選択肢：「From Scratch」設計，IPコア・ベース設計
- 期待する効果（イメージ）：



カスタム・ロジックとは？

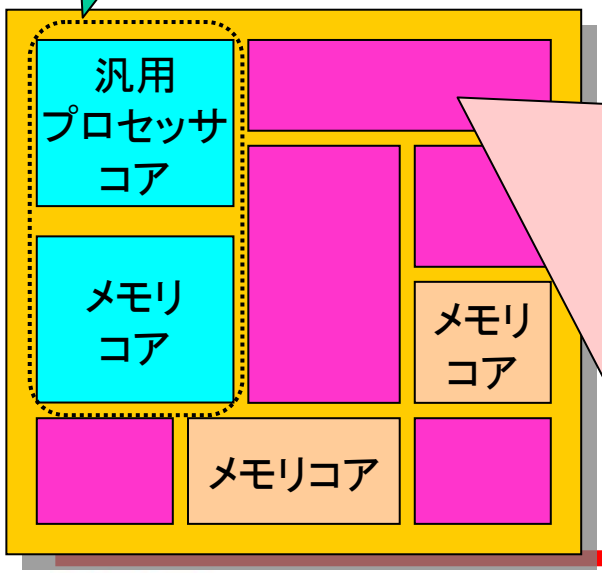
- システムを構成する各種機能のうち，「汎用プロセッサ」上では性能上の理由で実現しない／出来ない機能
  - 反対語：汎用機能，汎用ロジック
  - 同義語：専用ロジック
  - 類似語：専用ハードウェア，ハードウェア・ロジック
- 例：MPEG4の場合
  - 動き予測（ME），動き補償（MC）
  - DCT（離散コサイン変換），IDCT（逆変換）
- 専用機能の存在が「SoCがSoCたる」所以

プラットフォーム・ベース設計 vs. 他の設計手法

- 「From Scratch」
  - 毎回，ハードウェアの新規設計を実施
- IPコア・ベース
  - 既存の「ハードウェア設計（IPコア）」を再利用
  - SoC設計上の最重要課題
    - ①最適なIPコアの選択
    - ②同IPコアの特定半導体テクノロジ上での実装
  - IPコアの種類
    - ソフトIPコア
    - ハードIPコア
- プラットフォーム・ベース
  - 「プラットフォーム」上でカスタム・ロジックをハードウェアまたはソフトウェアとして実現
  - SoC設計上の最重要課題
    - ①最適なプラットフォームの選択
    - ②同プラットフォームの特定半導体テクノロジ上での実装
    - ③同プラットフォーム上でのカスタム・ロジックの実現
  - プラットフォームの種類
    - プロセッサ
    - コンフィギュラブル・プロセッサ
    - リコンフィギュラブル・プロセッサ
    - リコンフィギュラブル・ハードウェア

SoC設計への諸アプローチ

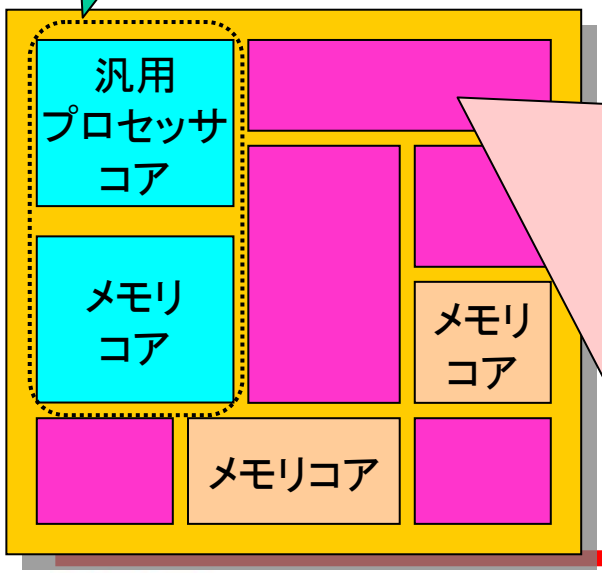
汎用機能の実現法
- 汎用プロセッサ＋ソフトウェア



カスタム・ロジック（専用機能）の実現法
- 「From Scratch」
  - 新規設計のハードウェア
- IPコア・ベース
  - 既存設計のハードウェア
- プラットフォーム・ベース
  - プロセッサ＋ソフトウェア
    - SONY/Toshiba/IBM Cell
    - QuickSilver ACM
  - コンフィギュラブル・プロセッサ＋ソフトウェア
    - Tensilica Xtensa
    - PDI VUPU
  - リコンフィギュラブル・プロセッサ＋ソフトウェア
    - IP Flex DAP/DNA
    - Stretch
    - CLUSS Redefis
  - リコンフィギュラブル・ハードウェア
    - FPGA (Xilinx, Altera, etc.)
    - NEC DRP

標準/汎用SoCプラットフォーム

汎用機能の実現法
- 汎用プロセッサ＋ソフトウェア

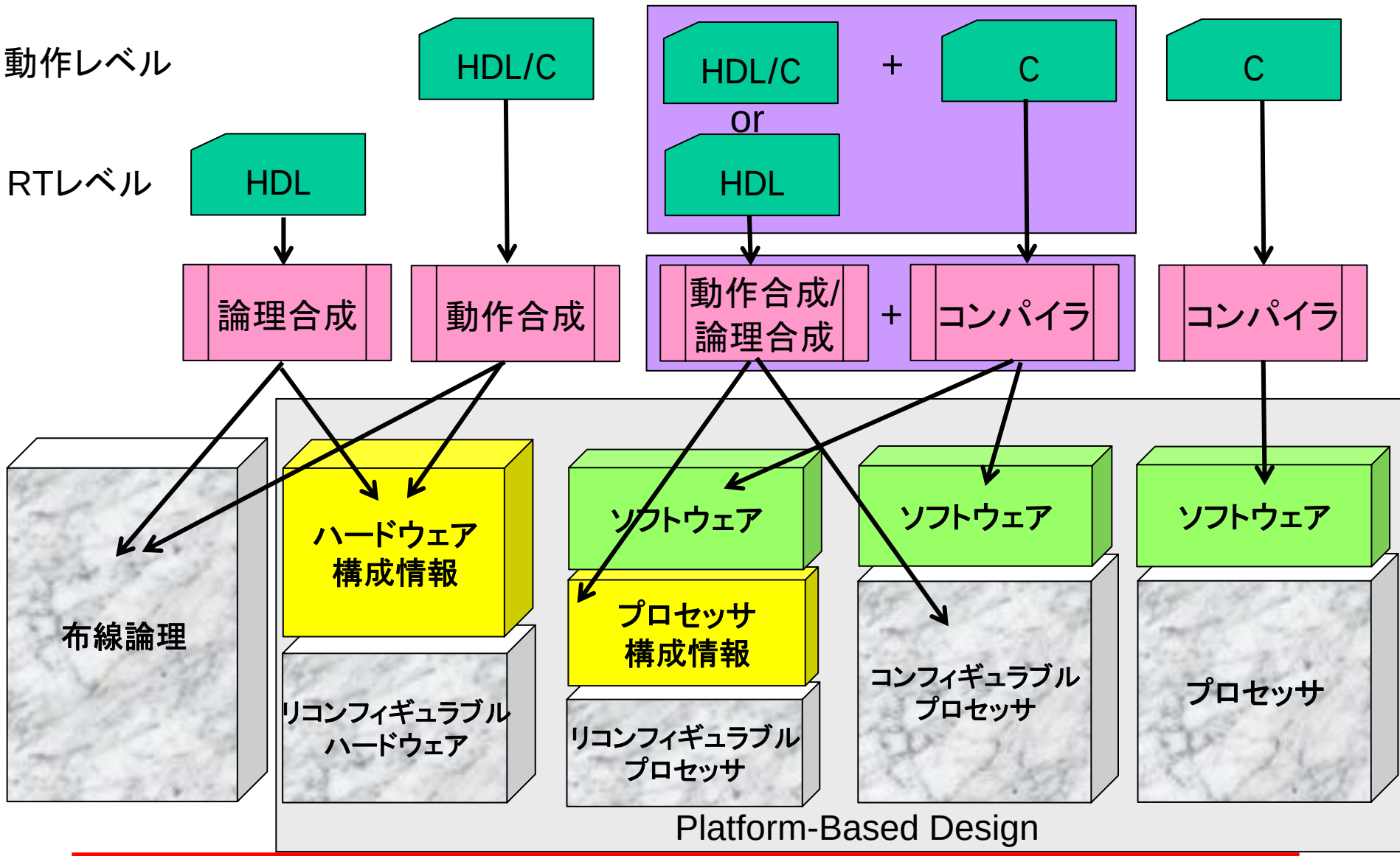


カスタム・ロジック（専用機能）の実現法
- 「From Scratch」
  - 新規設計のハードウェア
- IPコア・ベース
  - 既存設計のハードウェア
- プラットフォーム・ベース
  - プロセッサ＋ソフトウェア
    - SONY/Toshiba/IBM Cell
    - QuickSilver ACM
  - コンフィギュラブル・プロセッサ＋ソフトウェア
    - Tensilica Xtensa
    - PDI VUPU
  - リコンフィギュラブル・プロセッサ＋ソフトウェア
    - IP Flex DAP/DNA
    - Stretch
    - CLUSS Redefis
  - リコンフィギュラブル・ハードウェア
    - FPGA (Xilinx, Altera, etc.)
    - NEC DRP

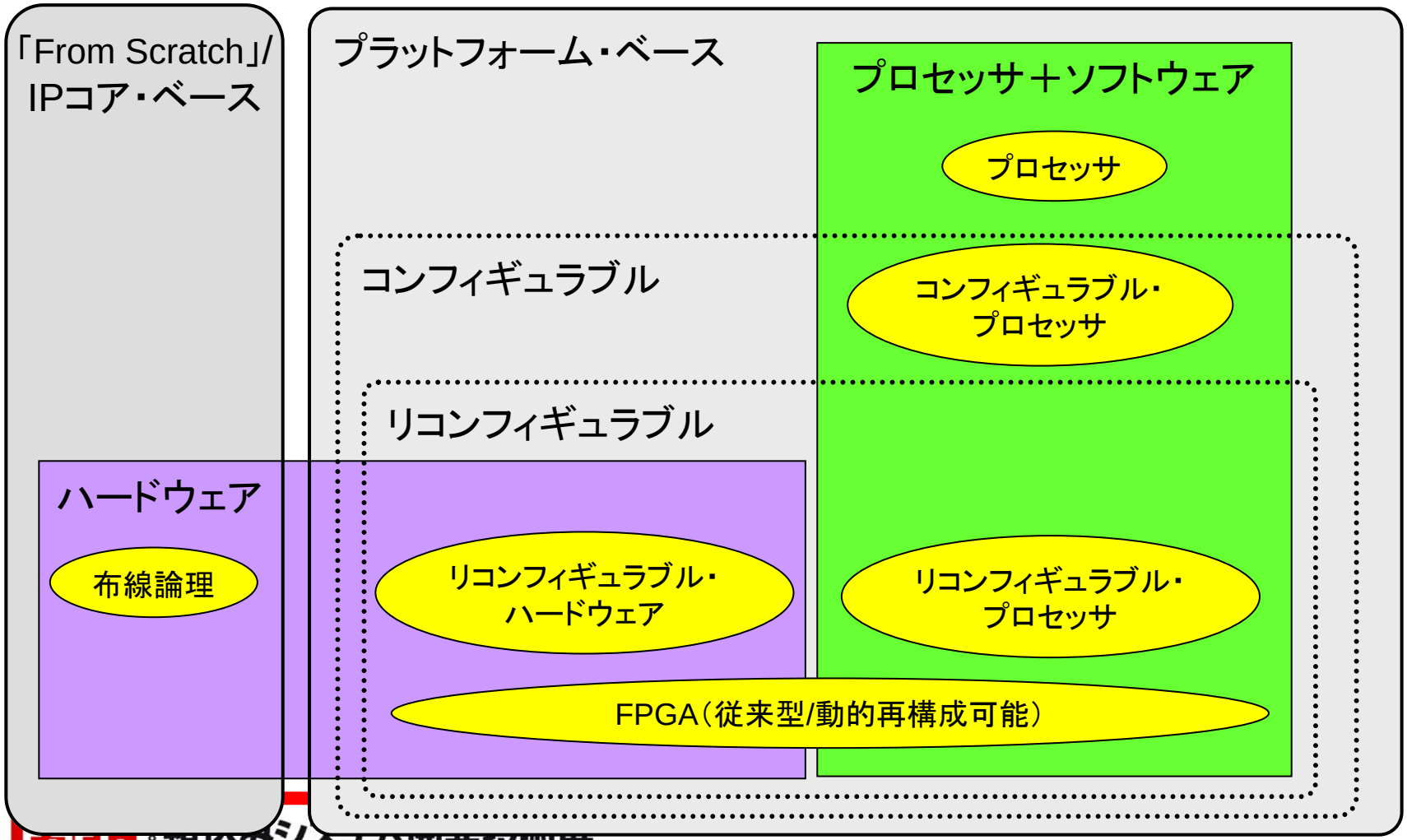
リファレンス・リスト

- プロセッサ＋ソフトウェア
  - 従来型の組込みプロセッサ/DSP
  - 均質型マルチプロセッサ
    - SONY/Toshiba/IBM Cell ➔
  - 非均質型マルチプロセッサ
    - QuickSilver ACM ➔ http://www.quicksilvertech.com/
- コンフィギュラブル・プロセッサ＋ソフトウェア
    - Tensilica Xtensa ➔ http://www.tensilica.com/
    - PDI VUPU ➔ http://www.pdi.co.jp/
- リコンフィギュラブル・プロセッサ＋ソフトウェア
    - IP Flex DAP/DNA ➔ http://www.ipflex.com/
    - Stretch ➔ http://www.stretchinc.com/
    - CLUSS Redefis ➔ http://www.fleets.jp/
- リコンフィギュラブル・プロセッサ
    - NEC DRP ➔ http://www.necel.com/ja/techhighlights/drp/

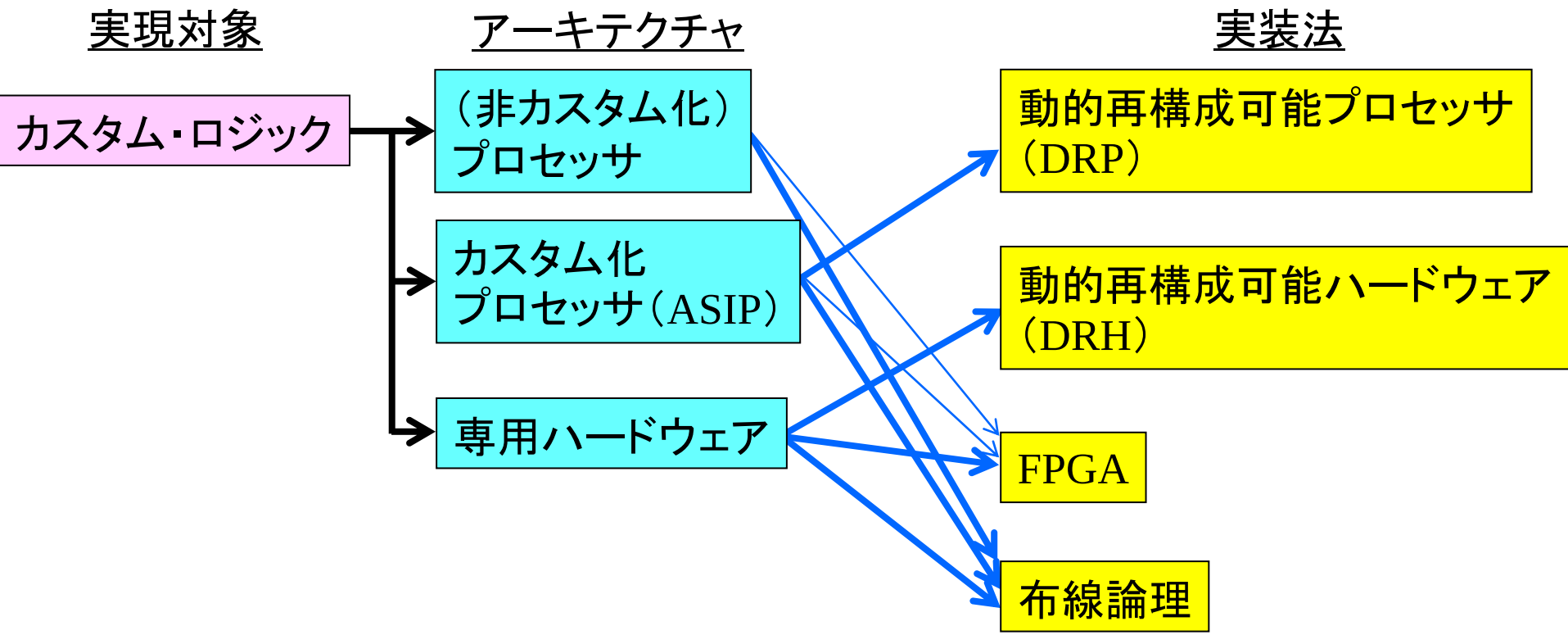
SoC設計への諸アプローチ



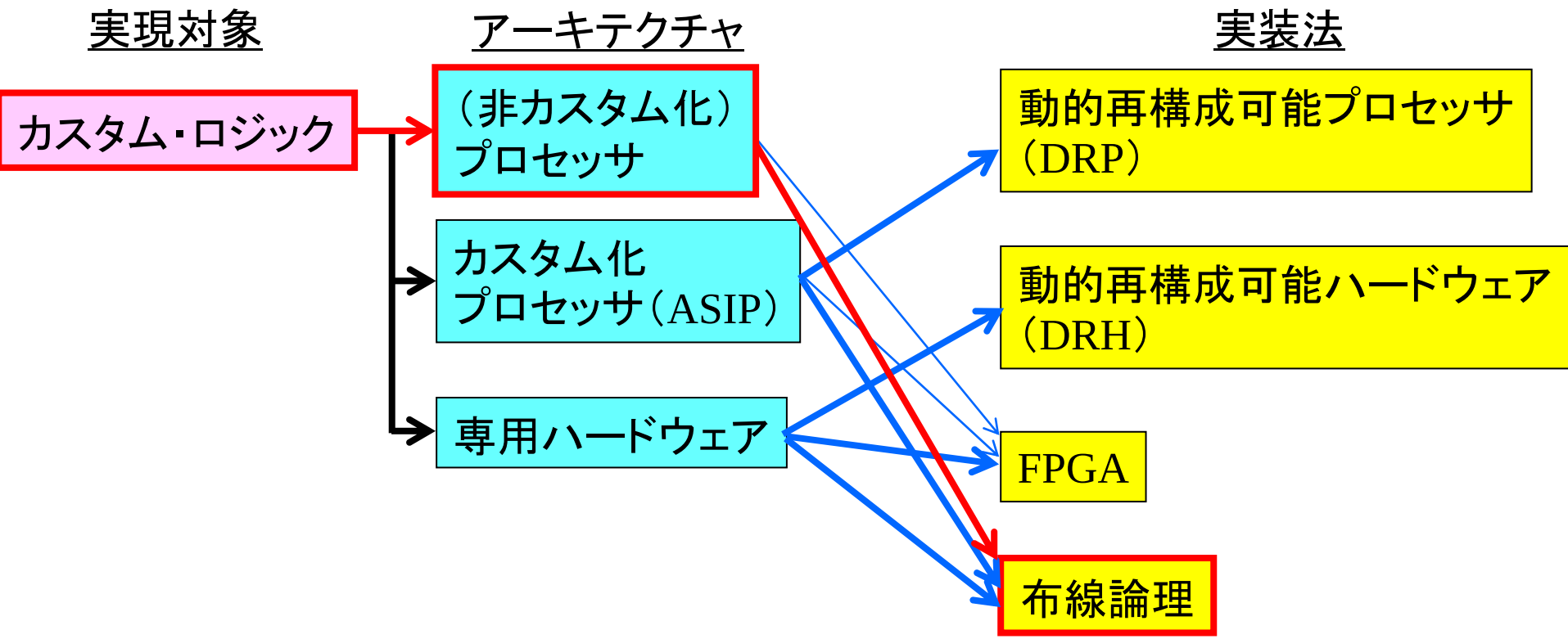
SoC設計への諸アプローチ



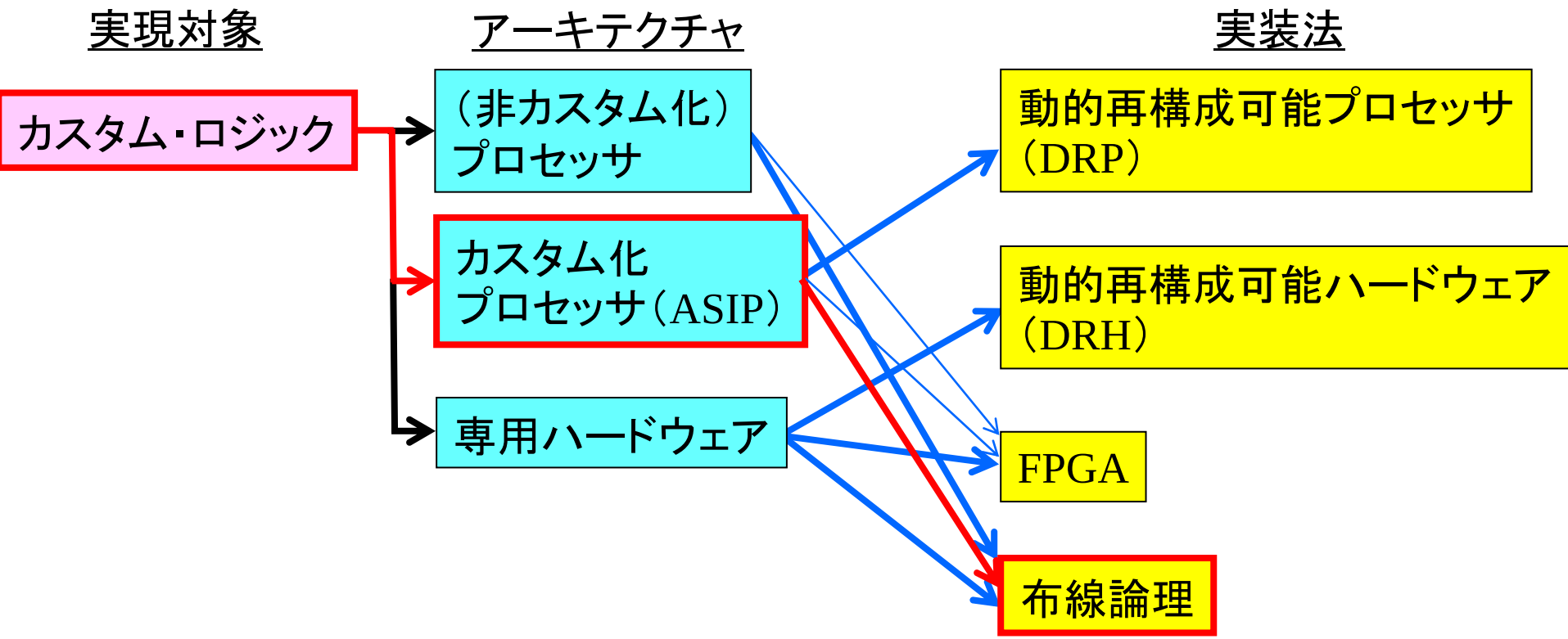
カスタム・ロジックの実現方法



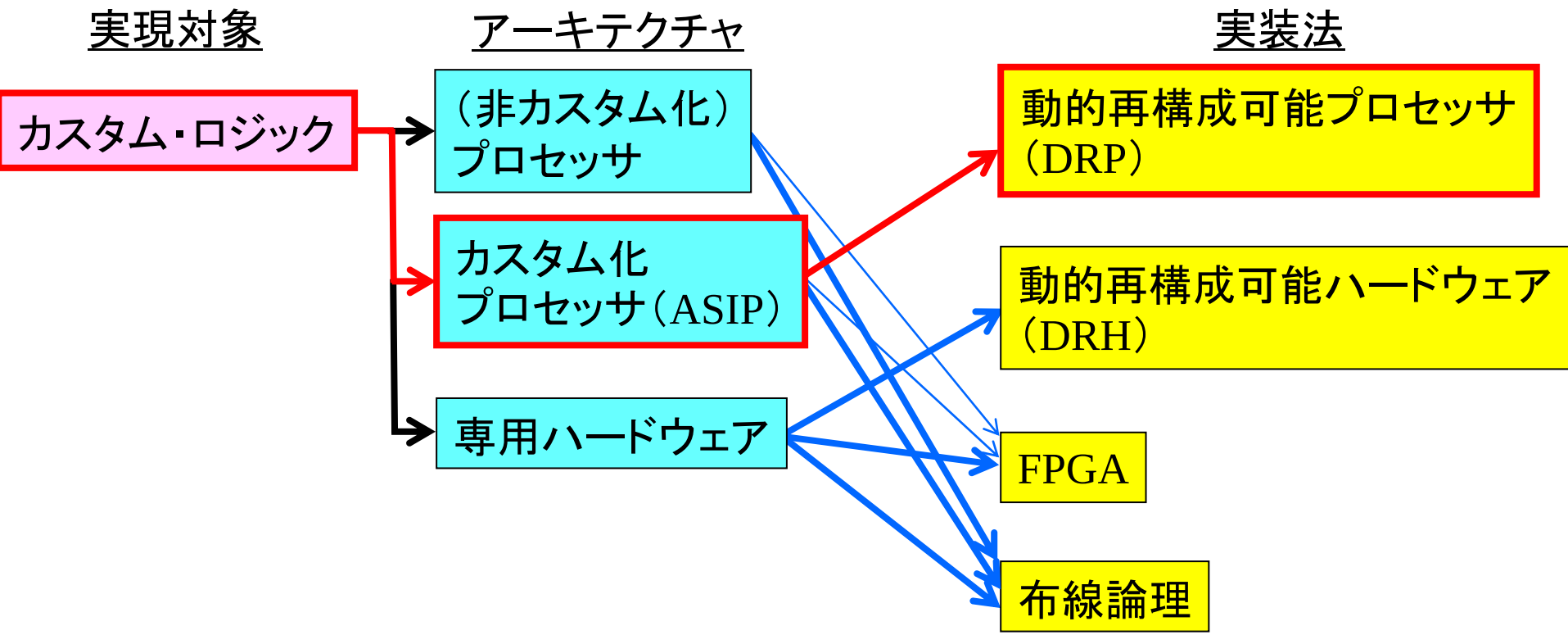
カスタム・ロジックの実現方法
～プロセッサ＋ソフトウェア～



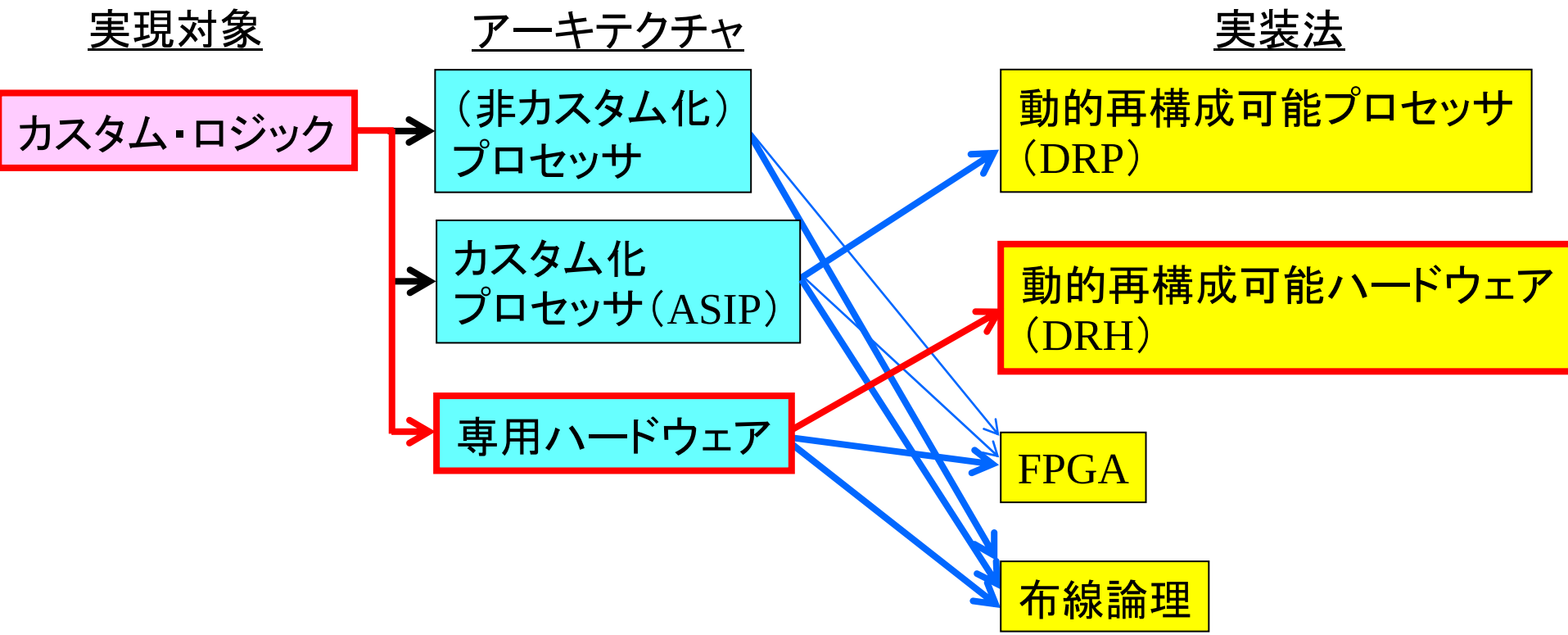
カスタム・ロジックの実現方法
～コンフィギュラブル・プロセッサ＋ソフトウェア～



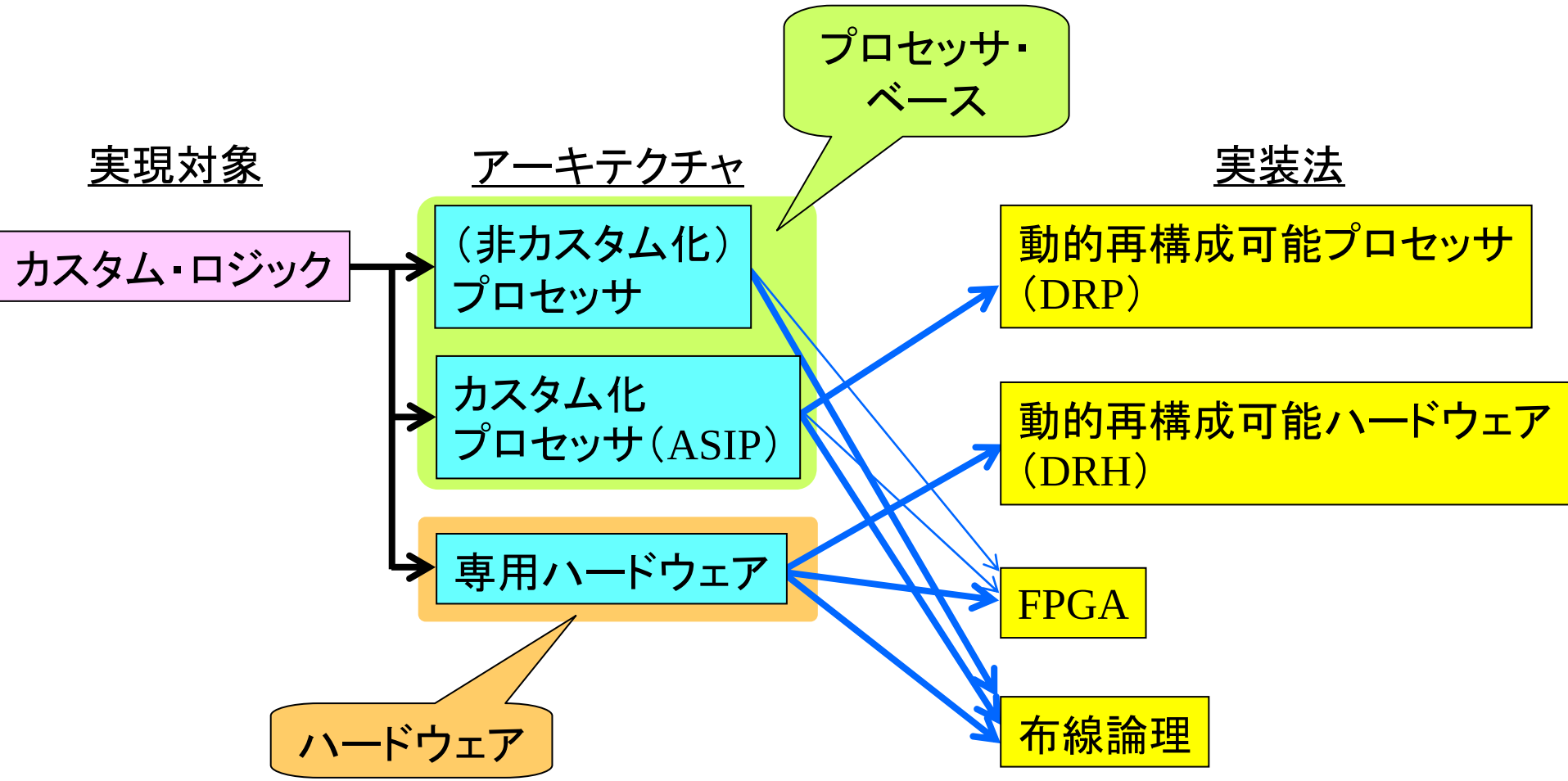
カスタム・ロジックの実現方法
～リコンフィギュラブル・プロセッサ＋ソフトウェア～



カスタム・ロジックの実現方法
～リコンフィギュラブル・ハードウェア～

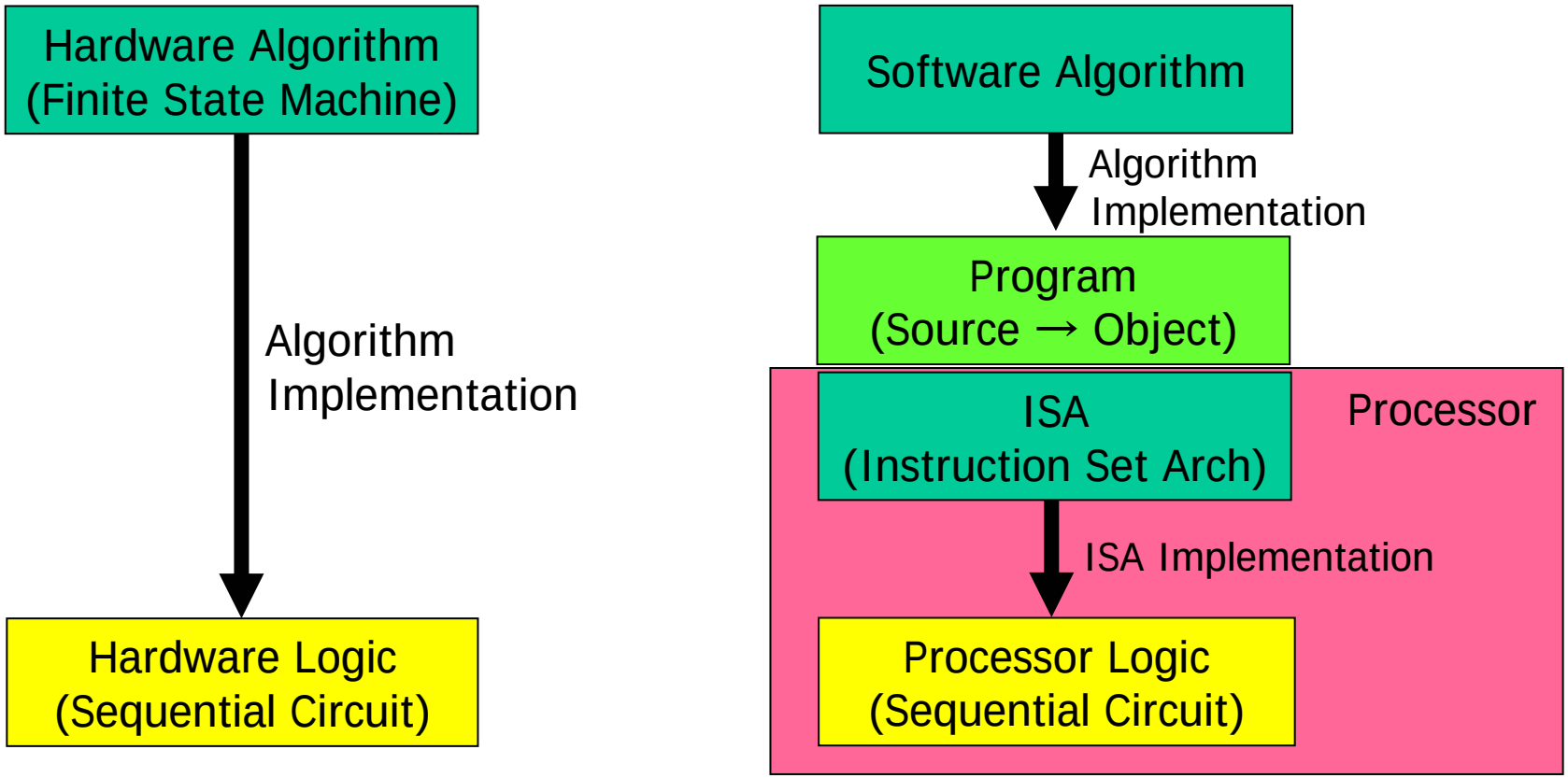


ハードウェア対プロセッサ・ベース



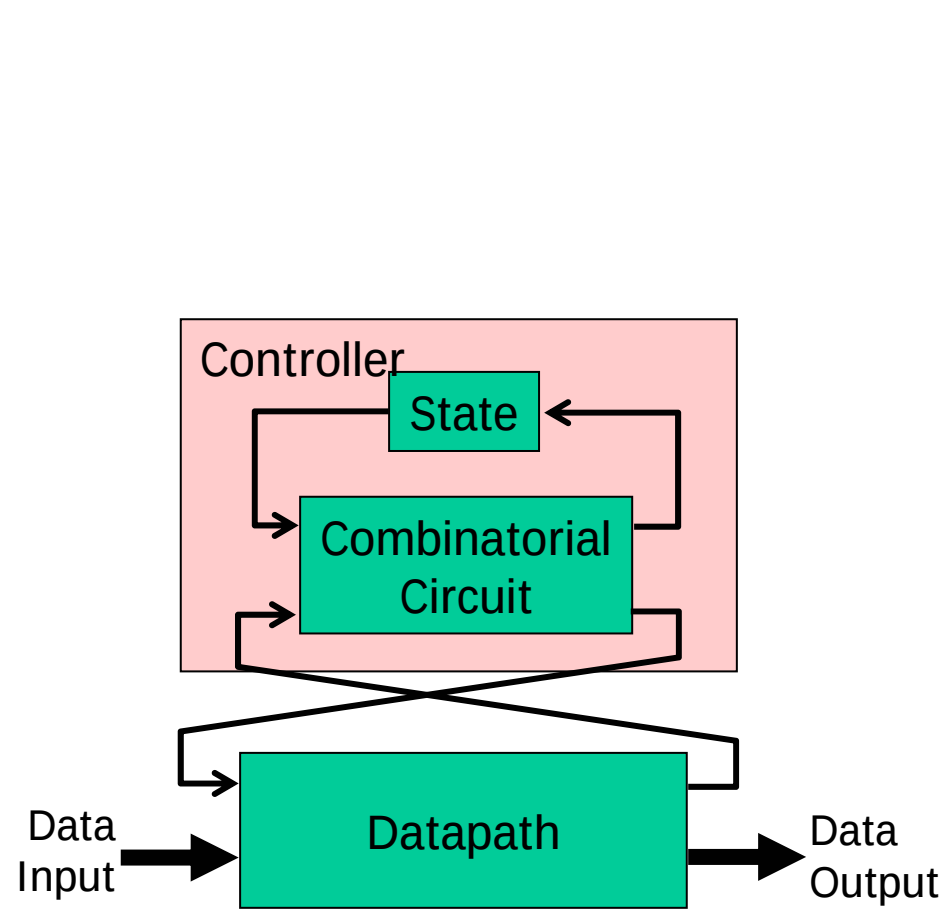
ハードウェア対プロセッサ・ベース

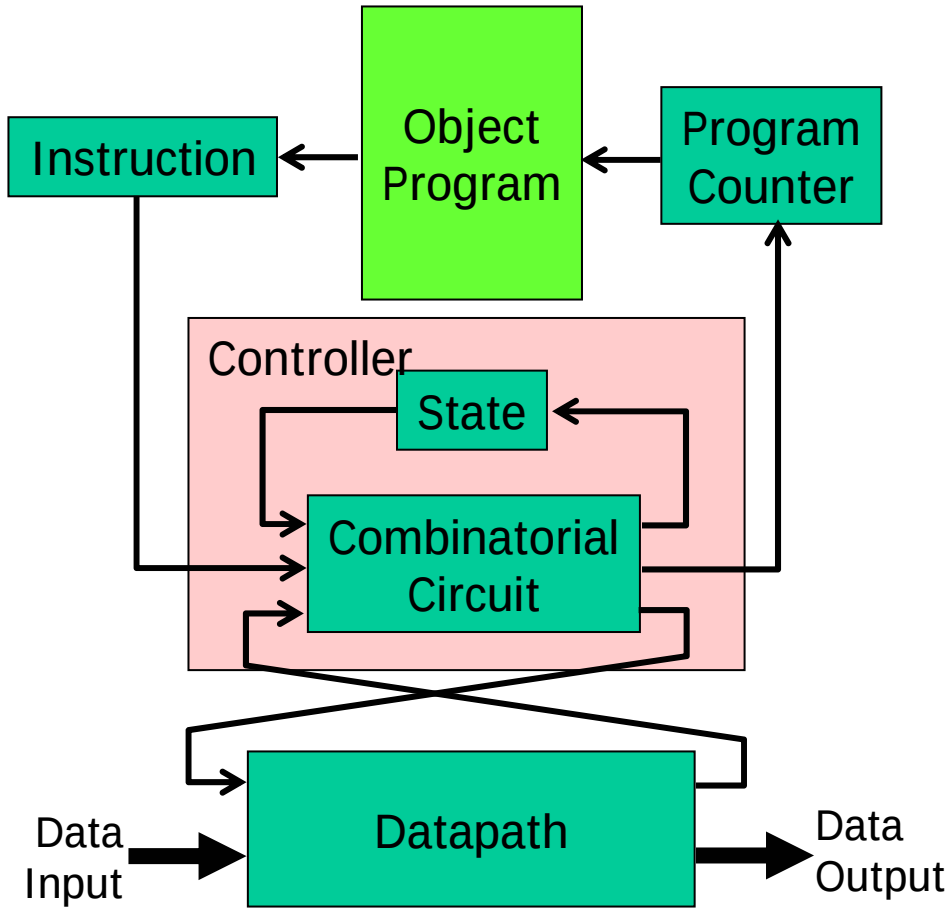
- ハードウェア
- プロセッサ＋ソフトウェア



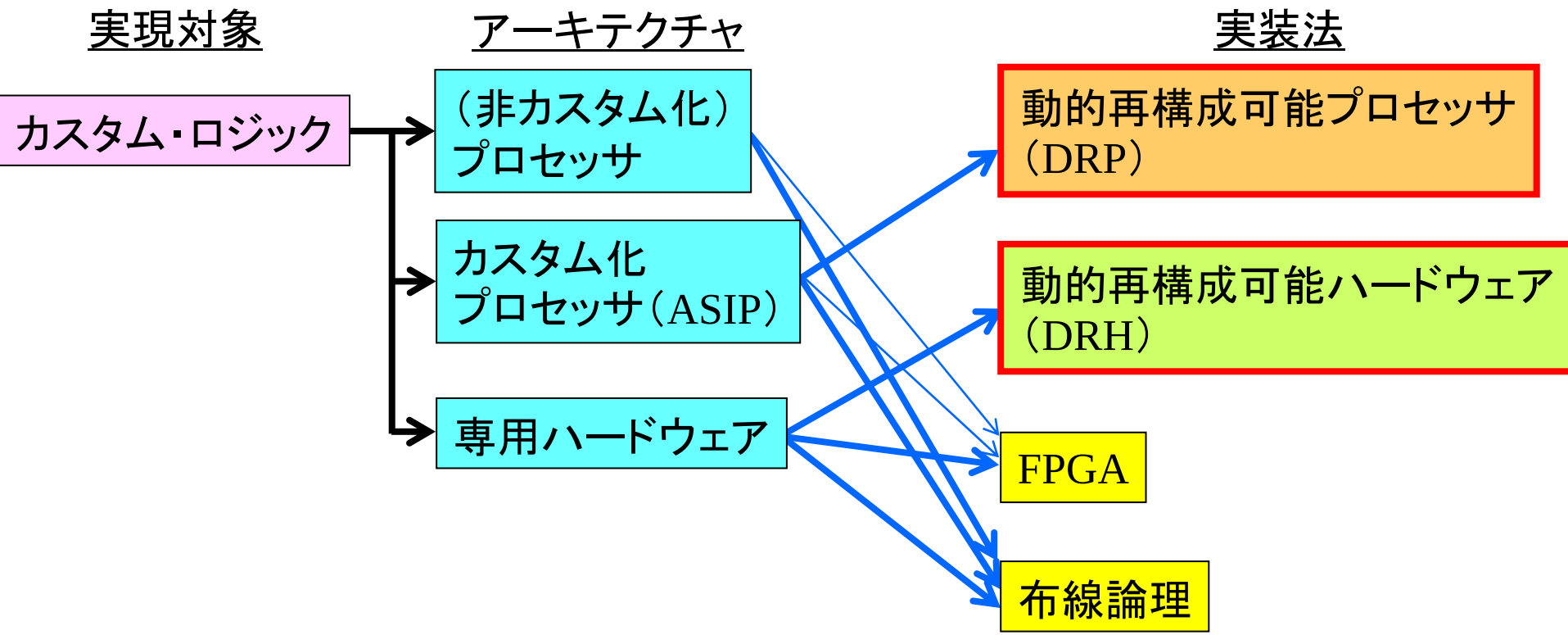
ハードウェア対プロセッサ・ベース

- ハードウェア
- プロセッサ＋ソフトウェア



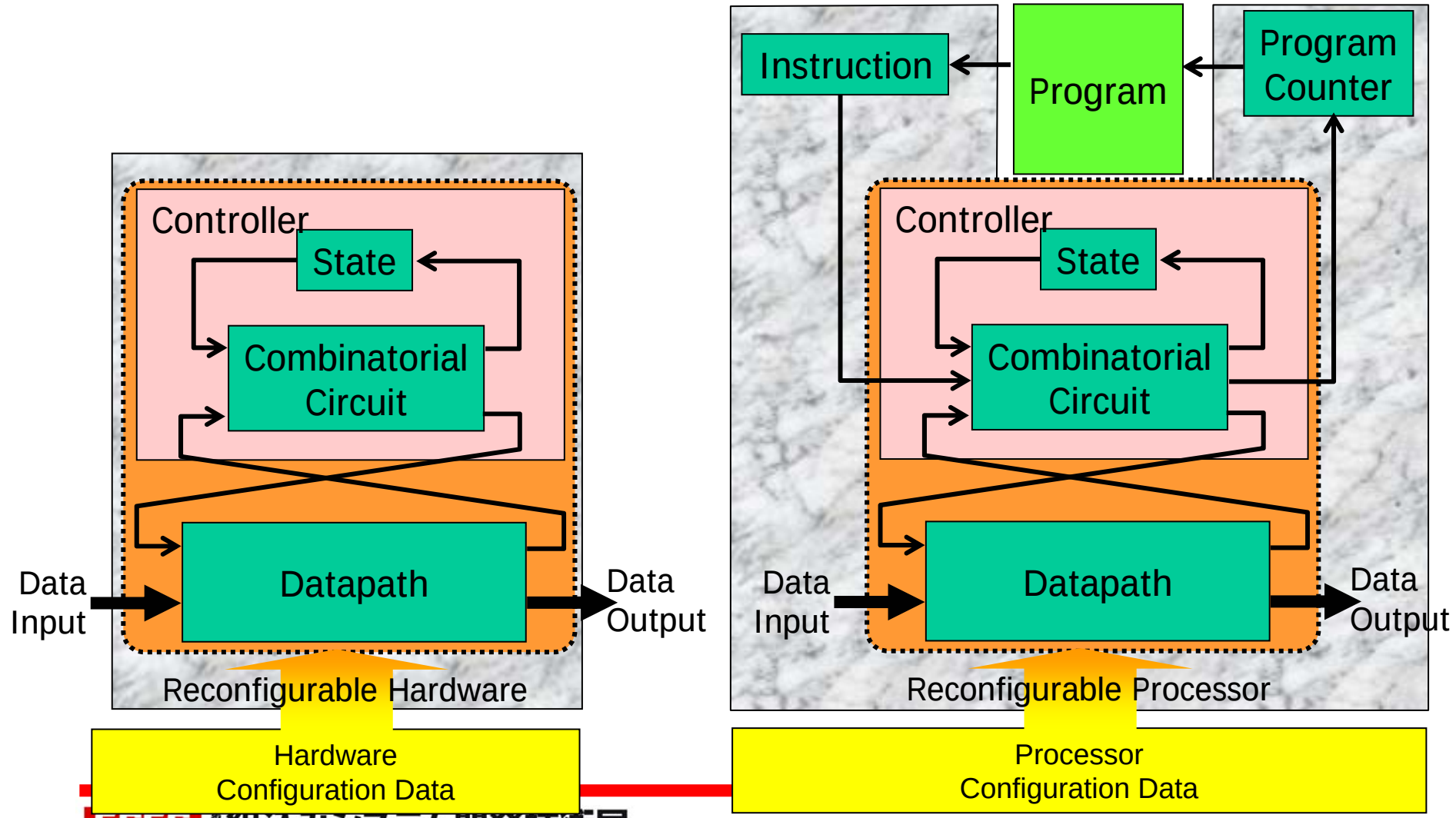


動的再構成可能ハードウェア対 動的再構成可能プロセッサ



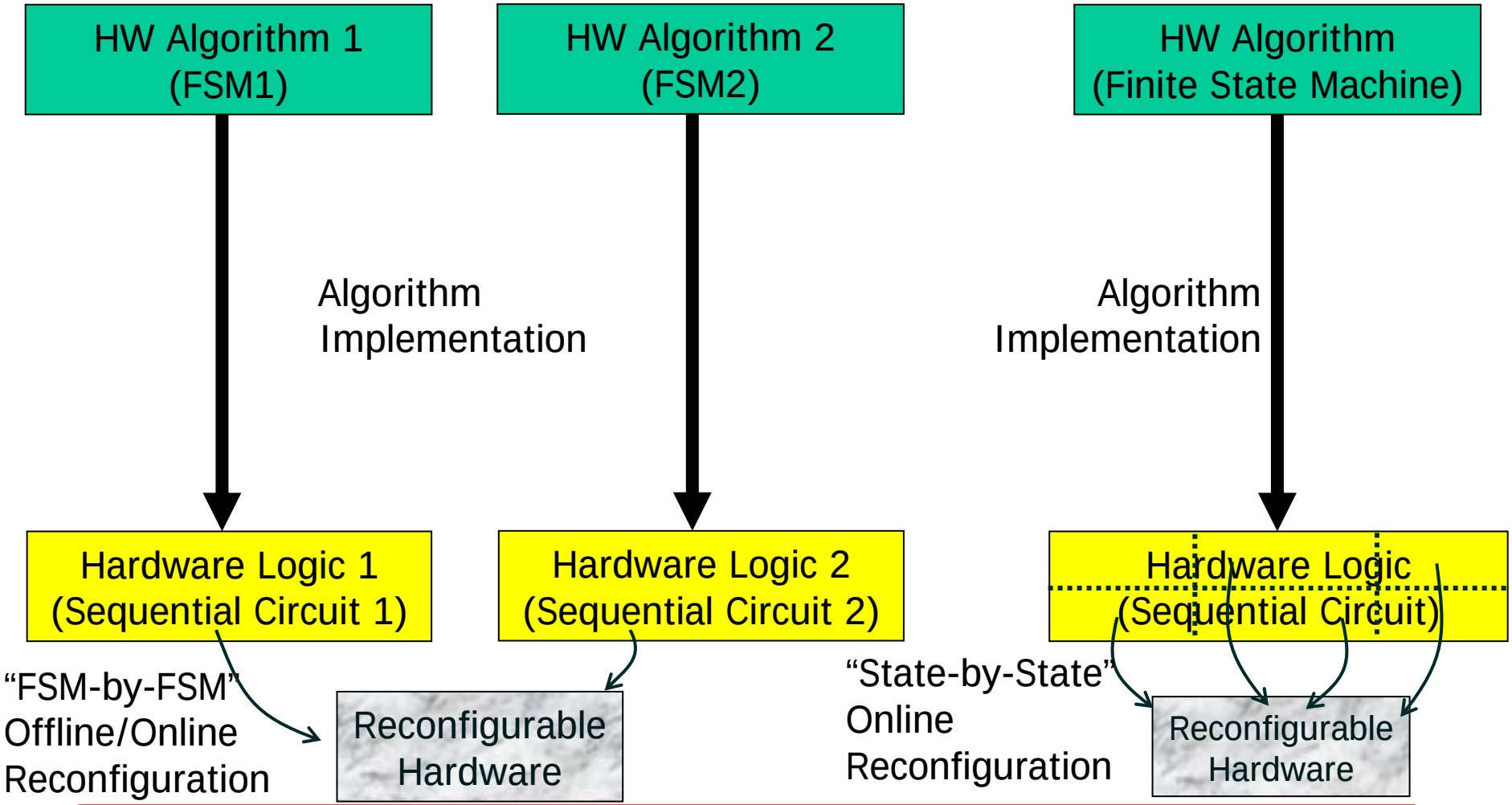
動的再構成可能ハードウェア対
動的再構成可能プロセッサ

- 動的再構成可能ハードウェア
- 動的再構成可能プロセッサ



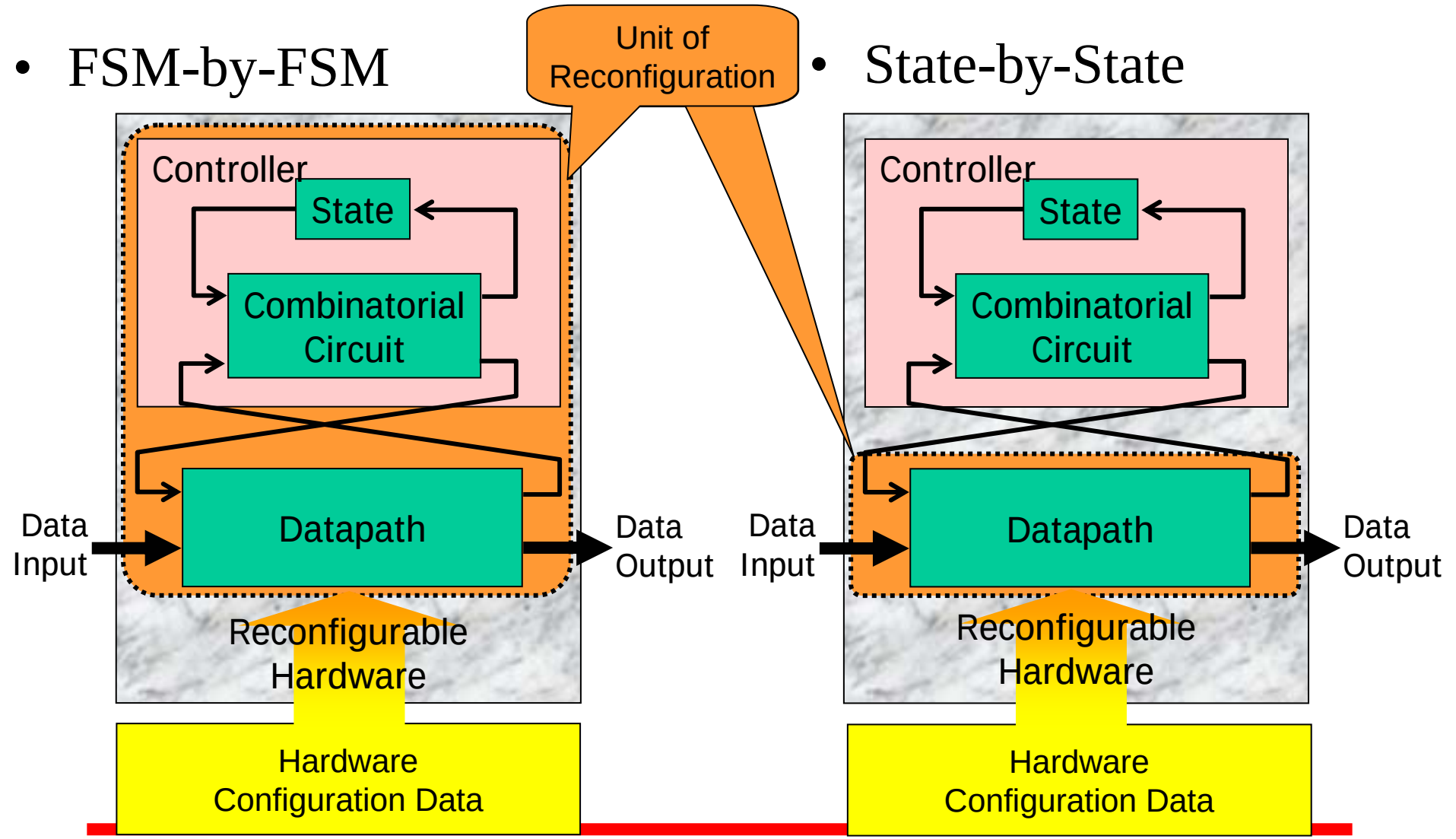
動的再構成可能ハードウェア
～時間的粒度～

- FSM-by-FSM
- State-by-State



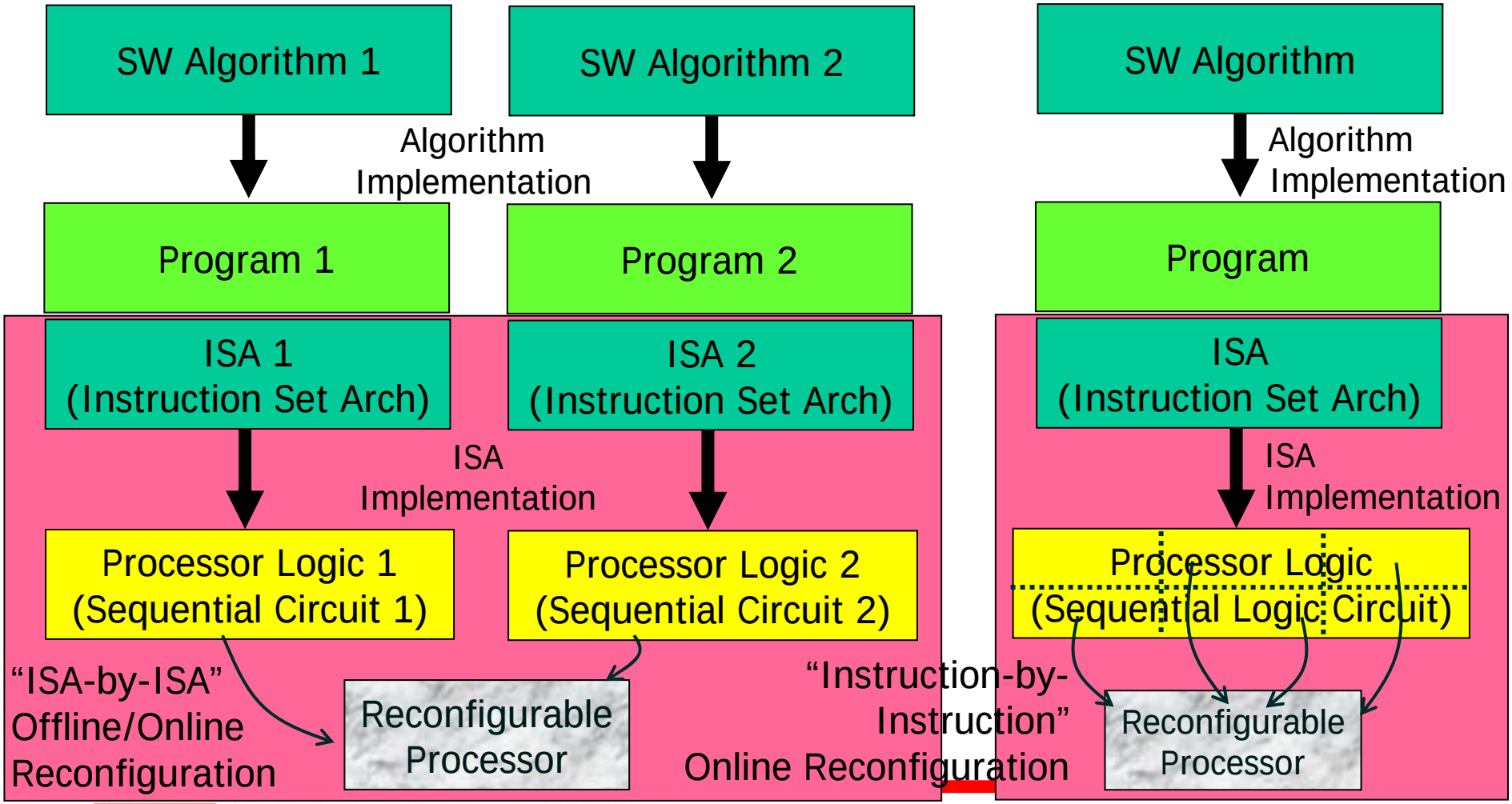
動的再構成可能ハードウェア
～時間的粒度～

- FSM-by-FSM
- State-by-State



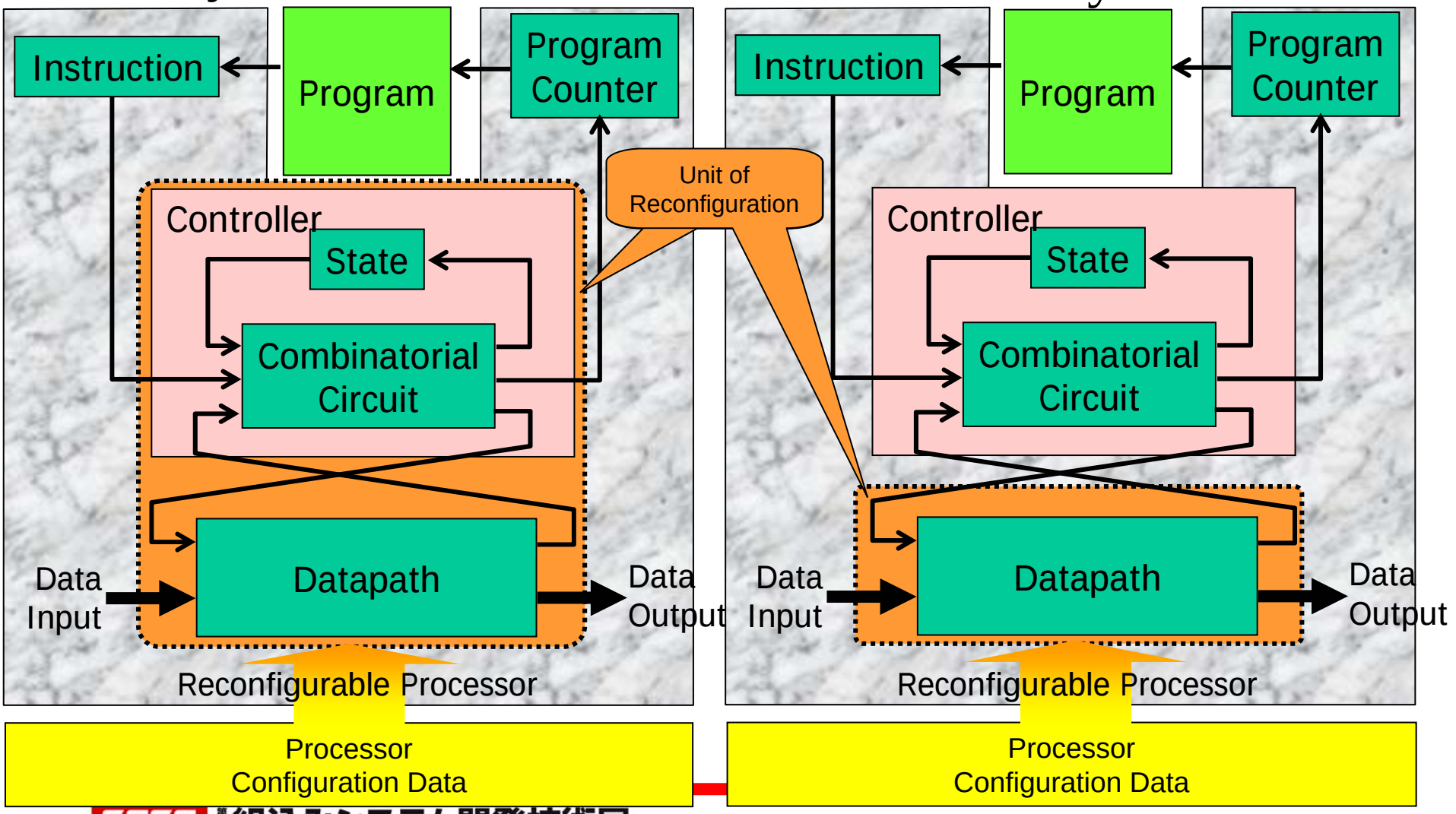
動的再構成可能プロセッサ ～時間的粒度～

- ISA-by-ISA
- Instruction-by-Instruction



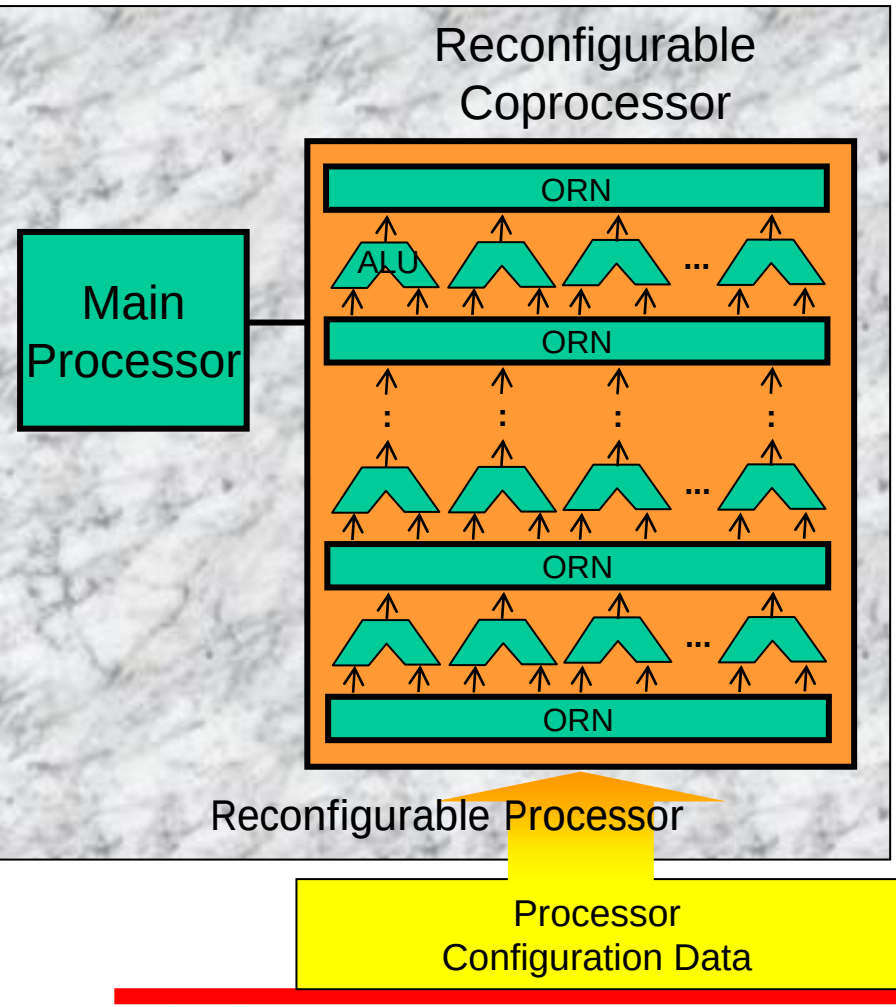
動的再構成可能プロセッサ
～時間的粒度～

- ISA-by-ISA
- Instruction-by-Instruction

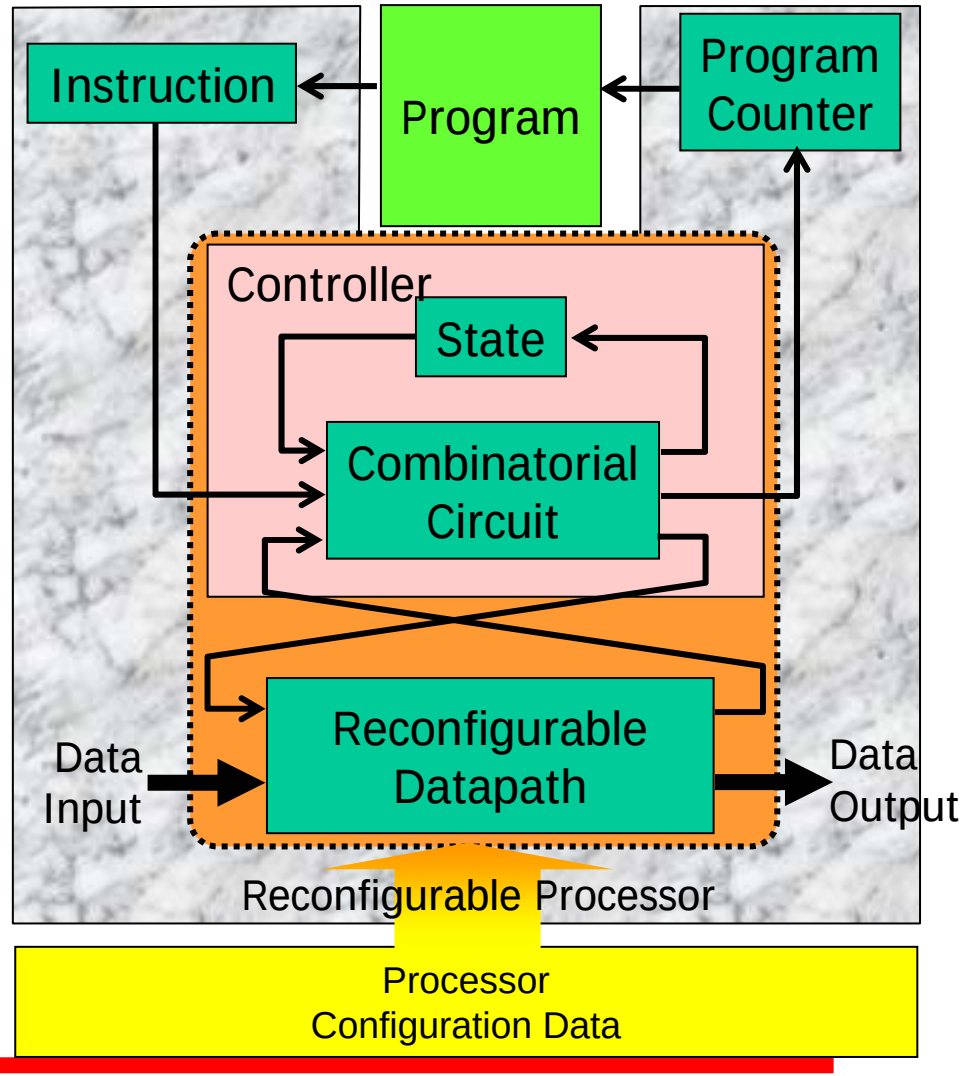


動的再構成可能プロセッサ・アーキテクチャ

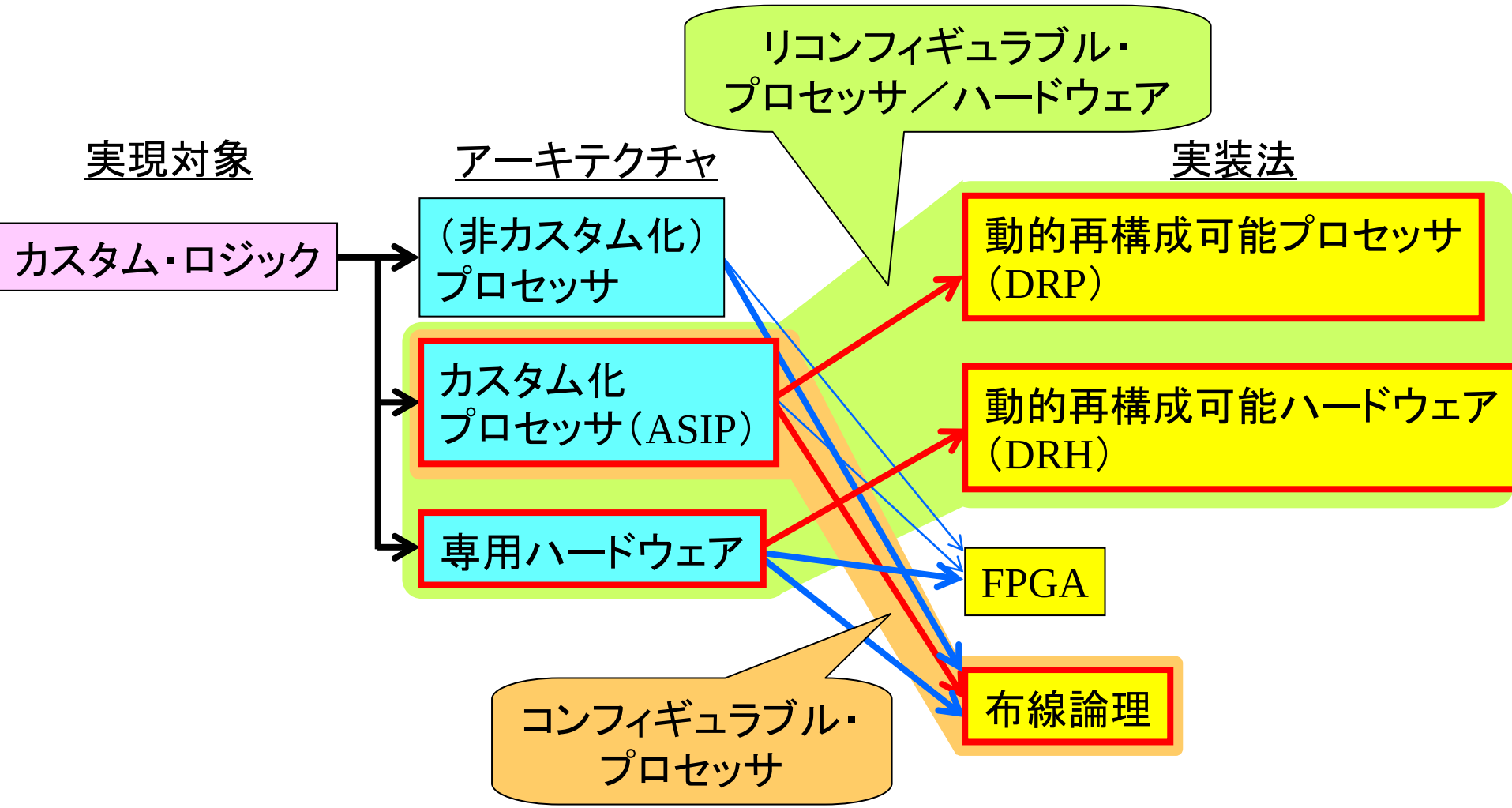
- コプロセッサ型



- プロセッサ内蔵型



コンフィギュラブル 対 リコンフィギュラブル

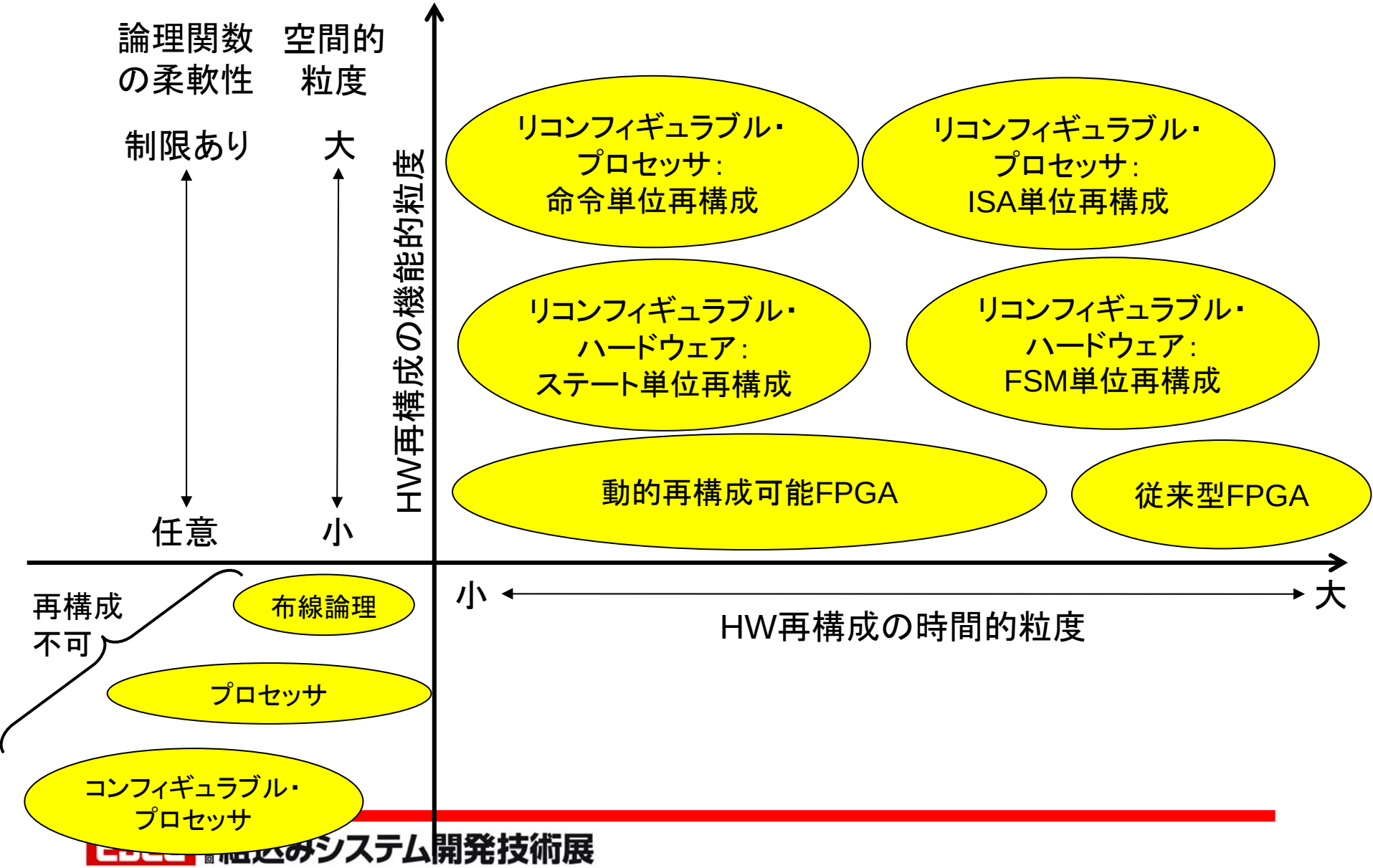


コンフィギュラブル 対 リコンフィギュラブル

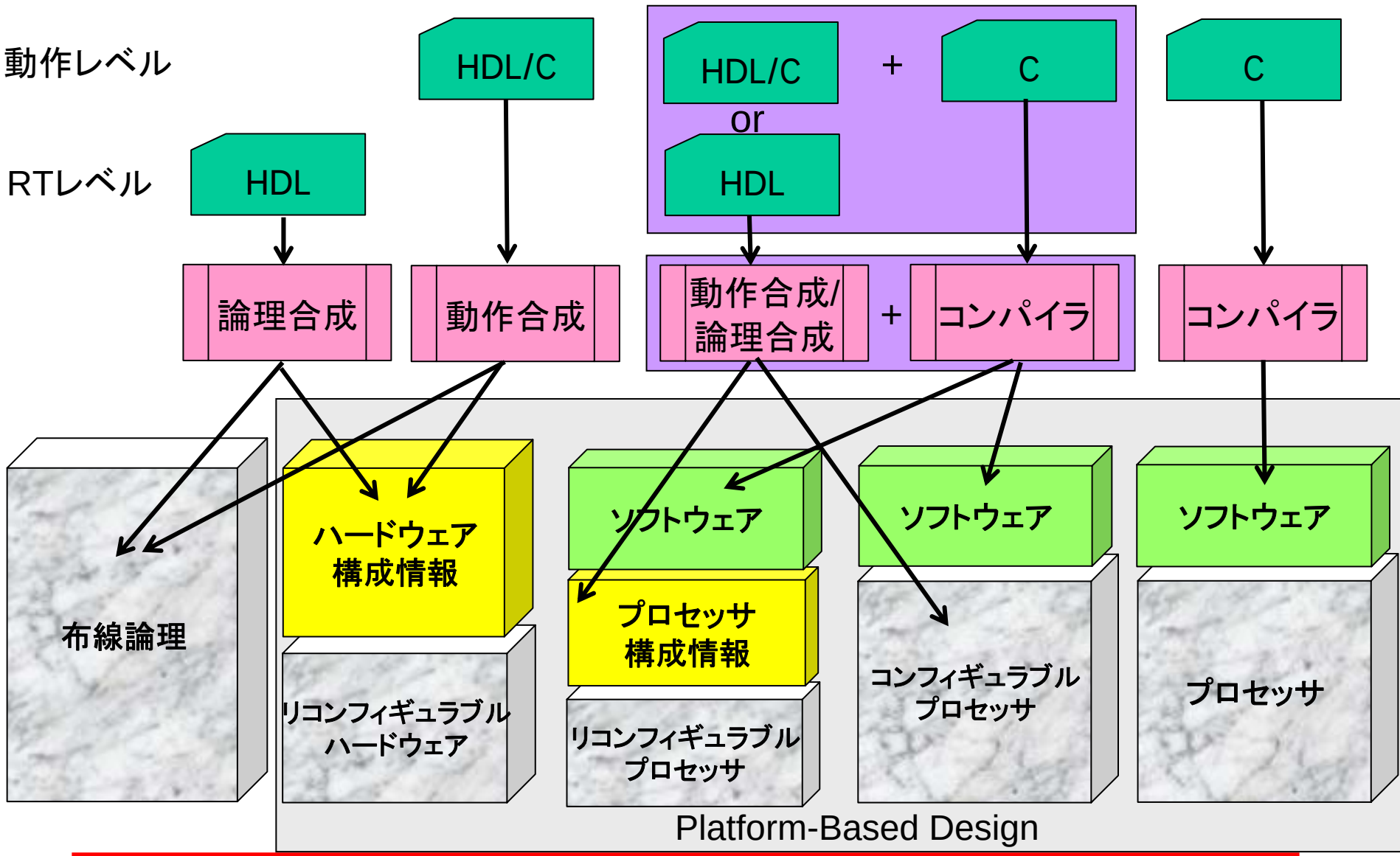
- 共通の性質：ハードウェアまたはプロセッサとして実現する機能が（SoC設計者によって）設定可能
  - Configurable（構成可能）：上記設定が「一度だけ」可能
  - Reconfigurable（再構成可能）：上記設定が「何度でも」可能

	ハードウェア	プロセッサ
非コンフィギュラブル	―	通常のプロセッサ
コンフィギュラブル	通常のハードウェア設計	コンフィギュラブル・プロセッサ
リコンフィギュラブル	リコンフィギュラブル・ハードウェア	リコンフィギュラブル・プロセッサ

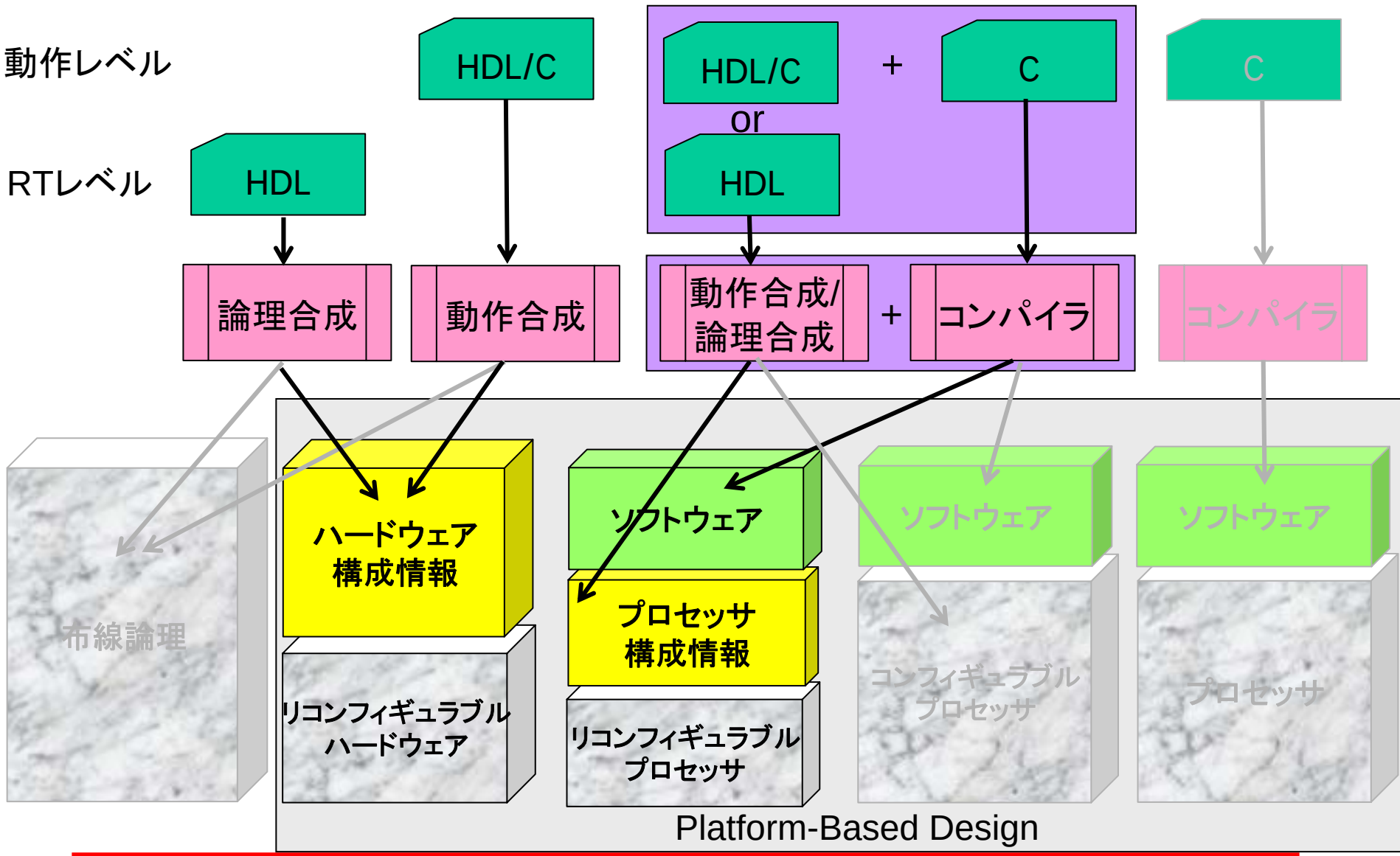
HW再構成可能性に関する設計空間



SoC設計への諸アプローチ

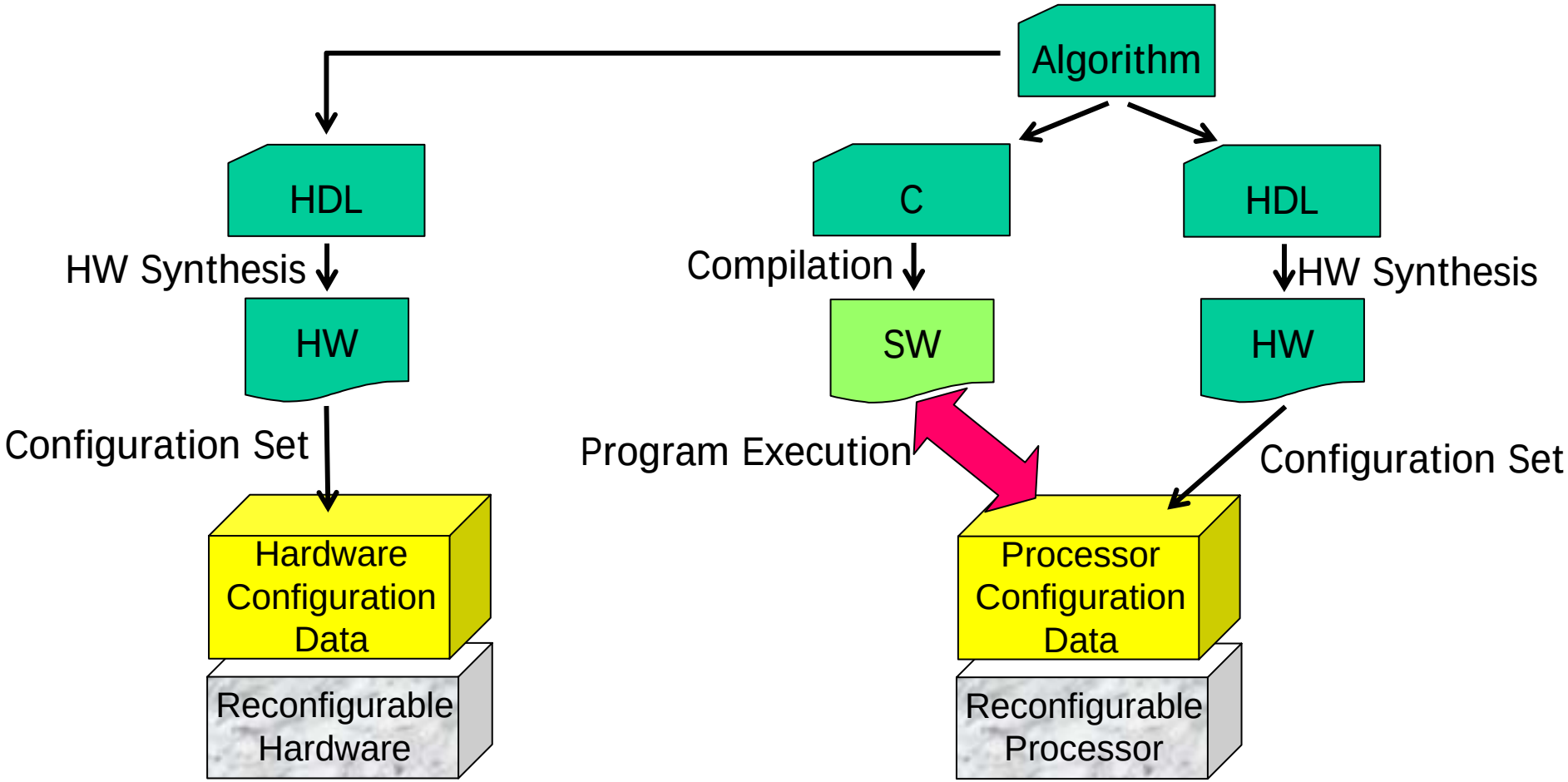


SoC設計への諸アプローチ

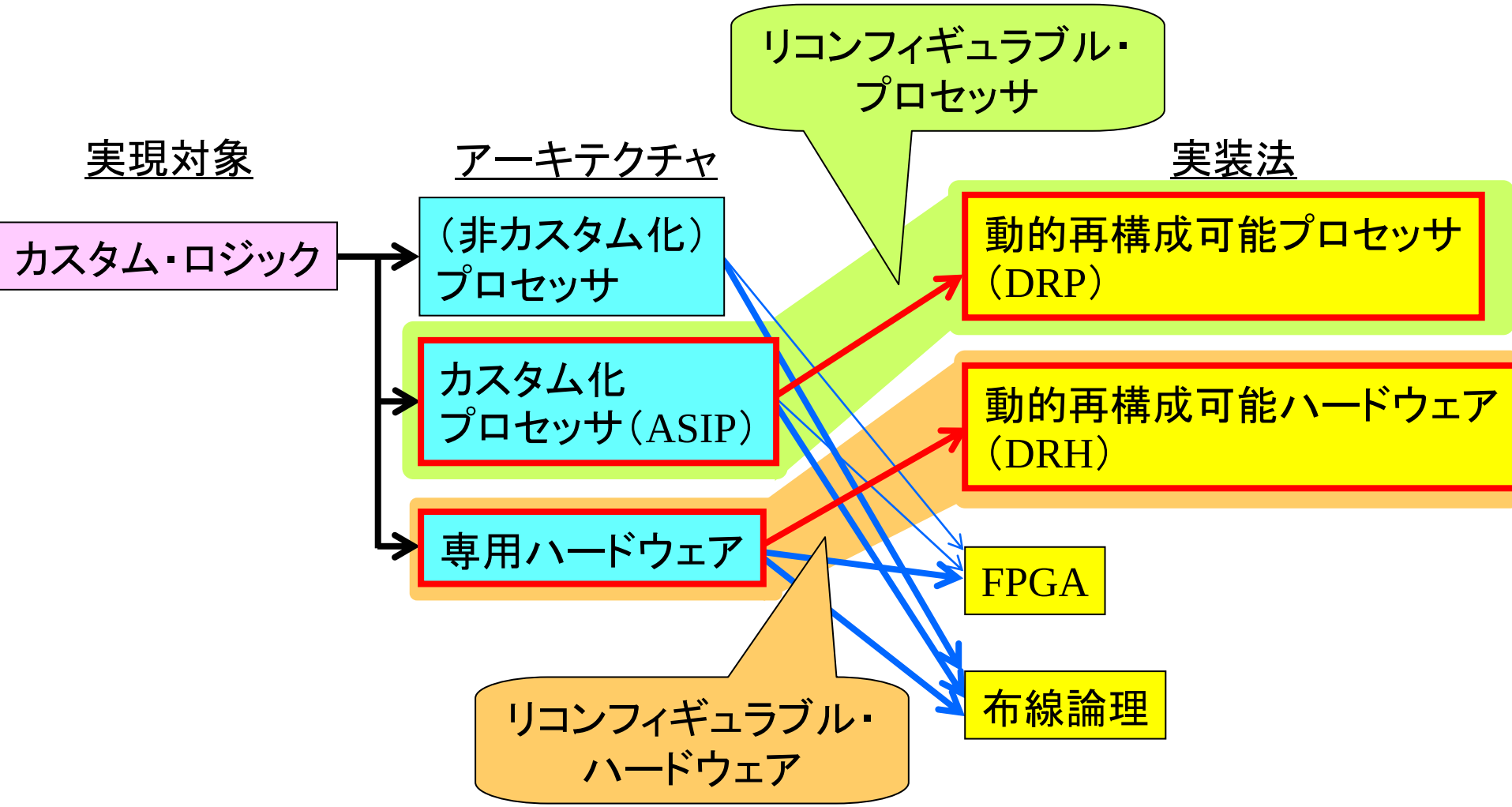


リコンフィギュラブル・ハードウェア 対 リコンフィギュラブル・プロセッサ

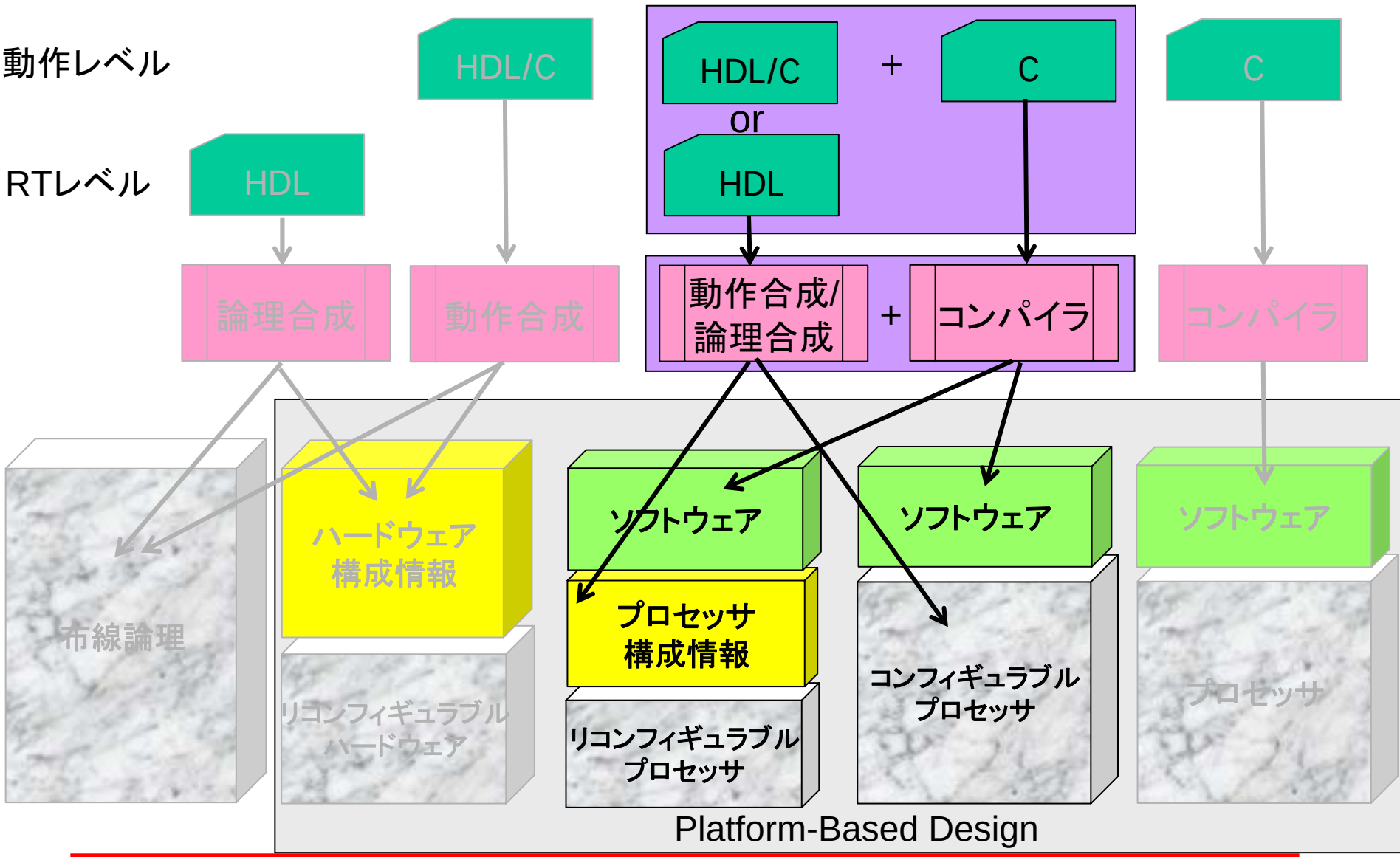
• リコンフィギュラブル・ハードウェア　• リコンフィギュラブル・プロセッサ



リコンフィギュラブル・ハードウェア 対 リコンフィギュラブル・プロセッサ

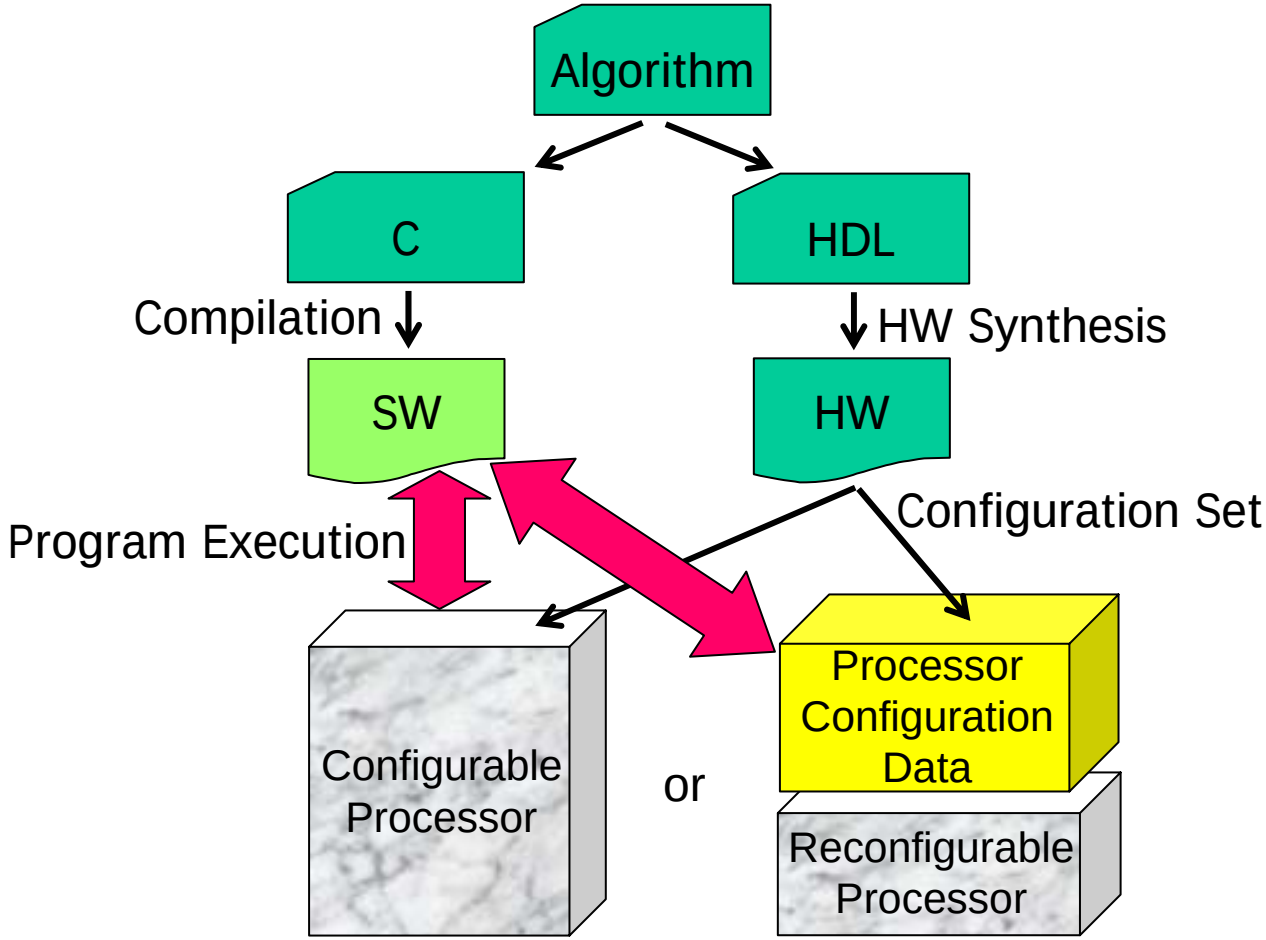


SoC設計への諸アプローチ

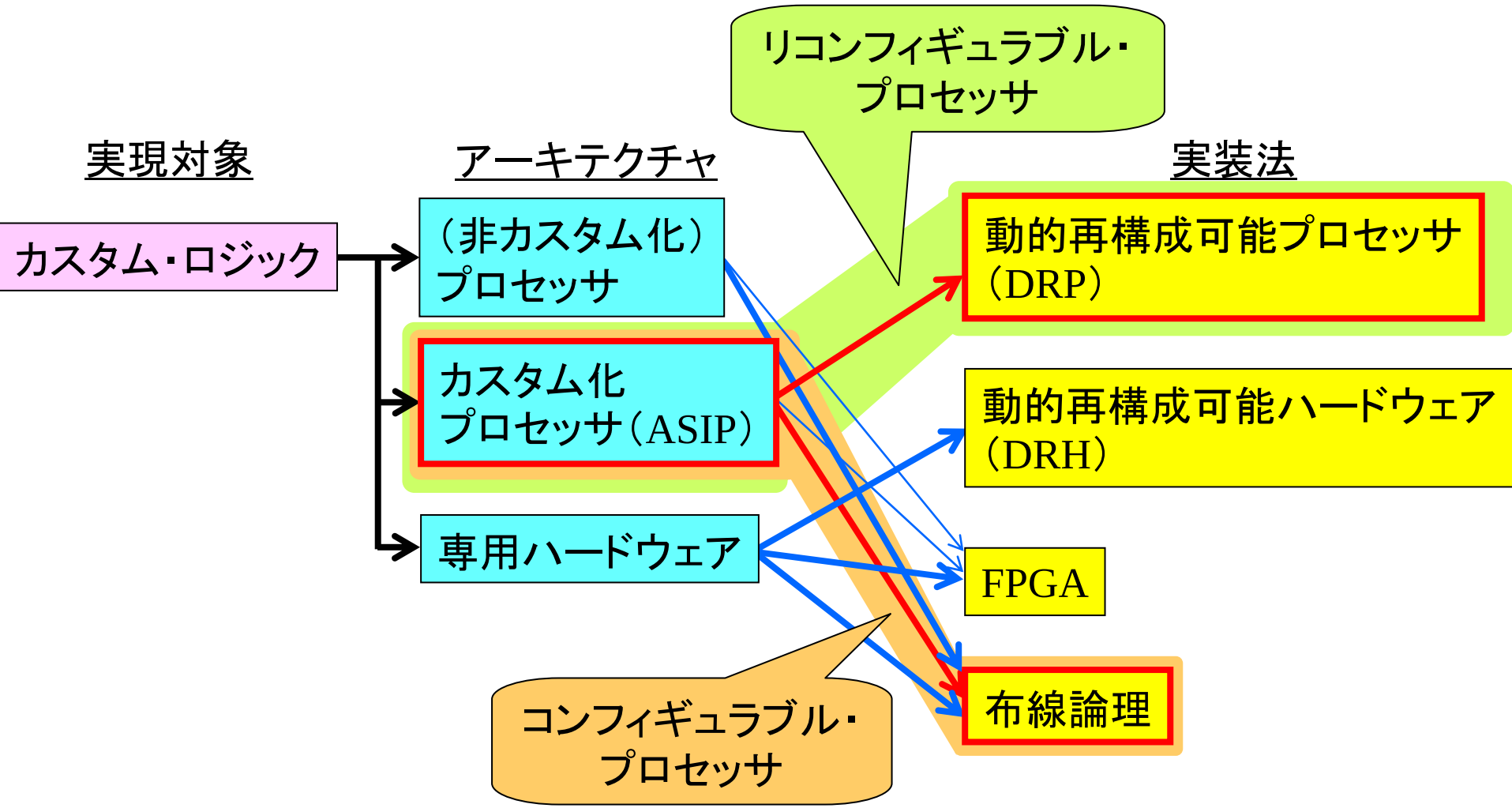


コンフィギュラブル・プロセッサ 対 リコンフィギュラブル・プロセッサ

- コンフィギュラブル・プロセッサ
- リコンフィギュラブル・プロセッサ

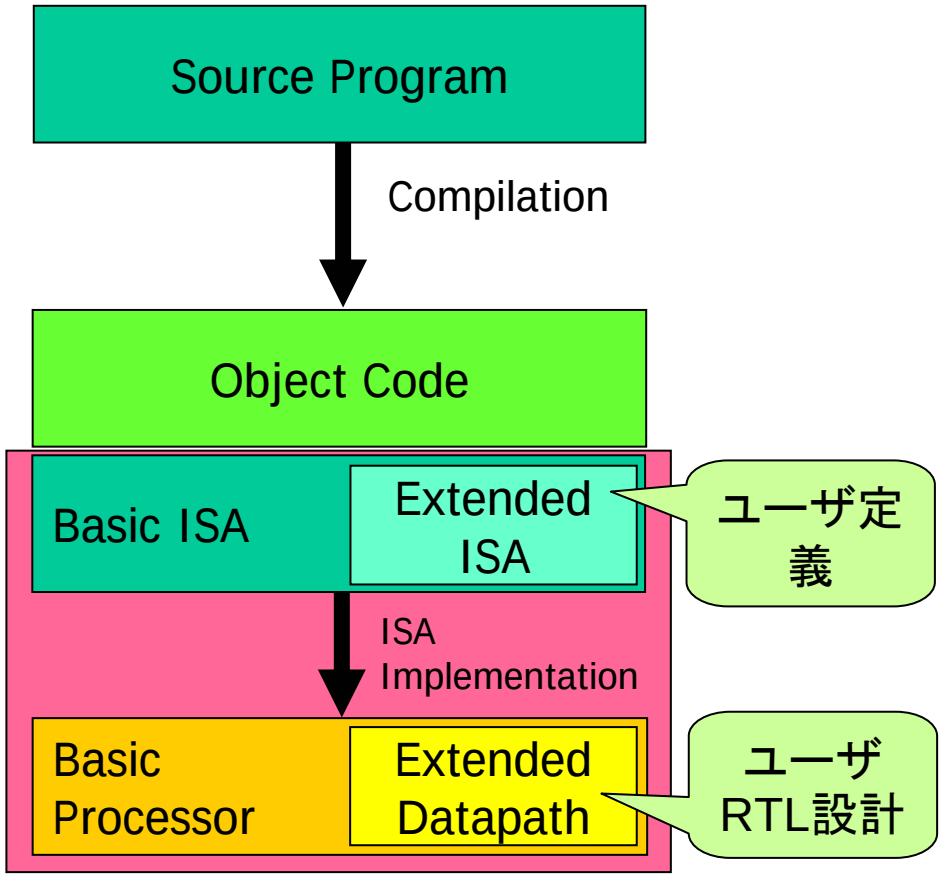


コンフィギュラブル・プロセッサ 対 リコンフィギュラブル・プロセッサ

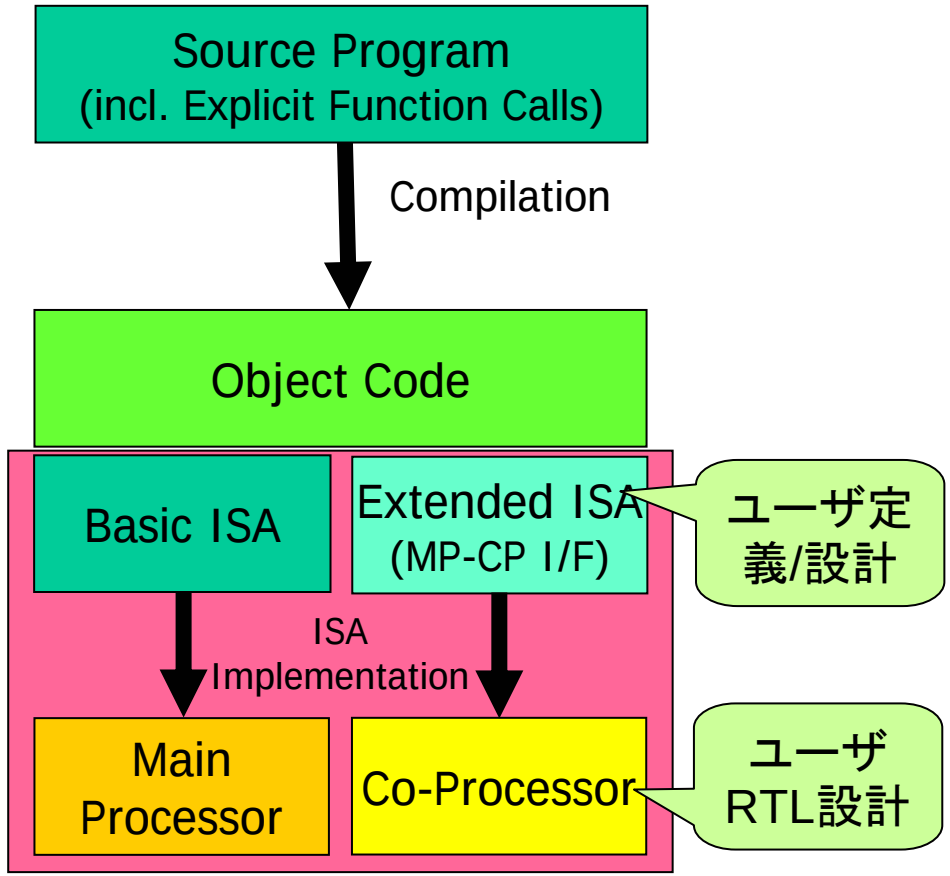


コンフィギュラブル・プロセッサ

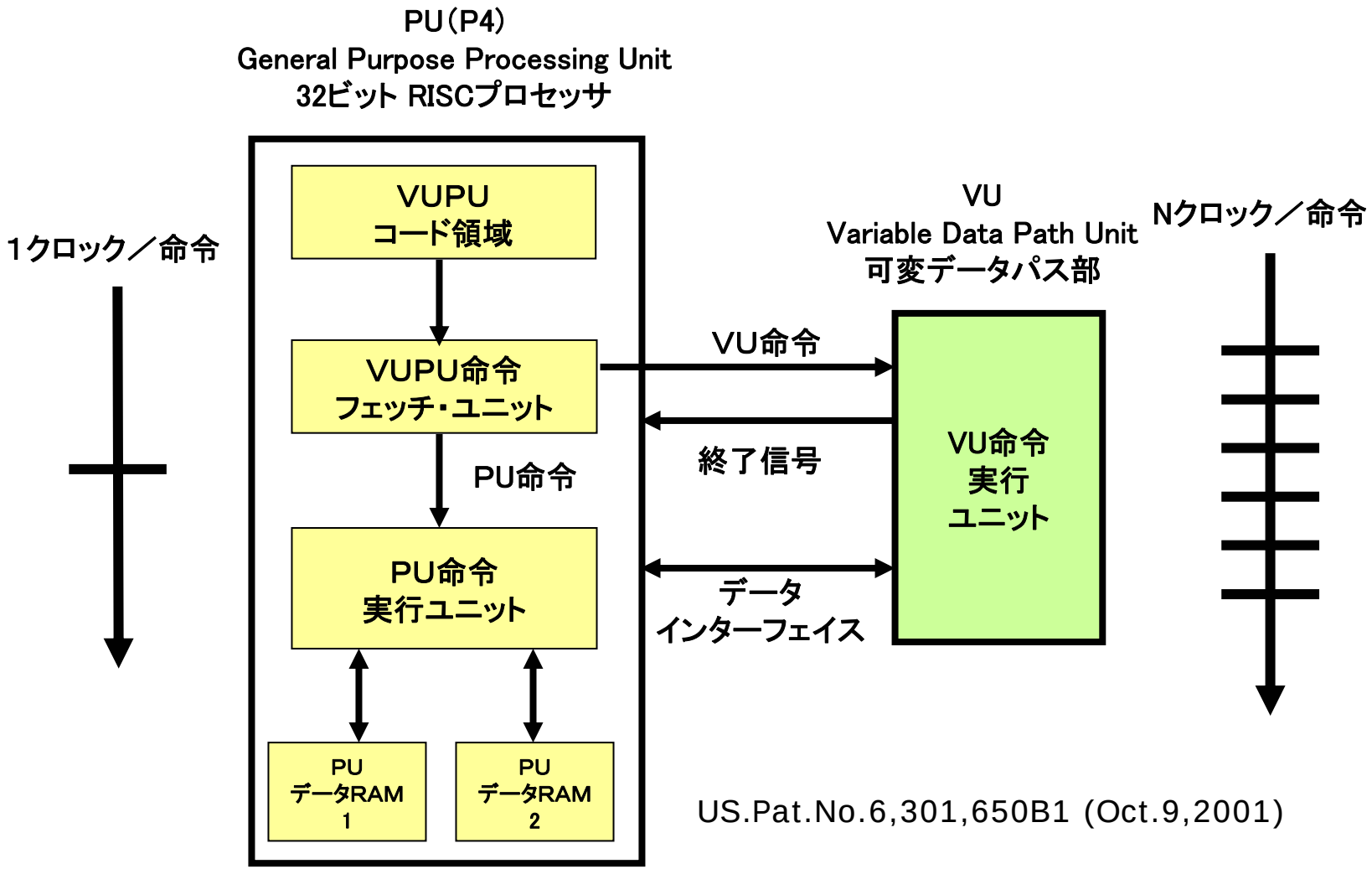
- Tensilica Xtensa



- PDI VUPU



VUPU基本アーキテクチャ



US.Pat.No.6,301,650B1 (Oct.9,2001)

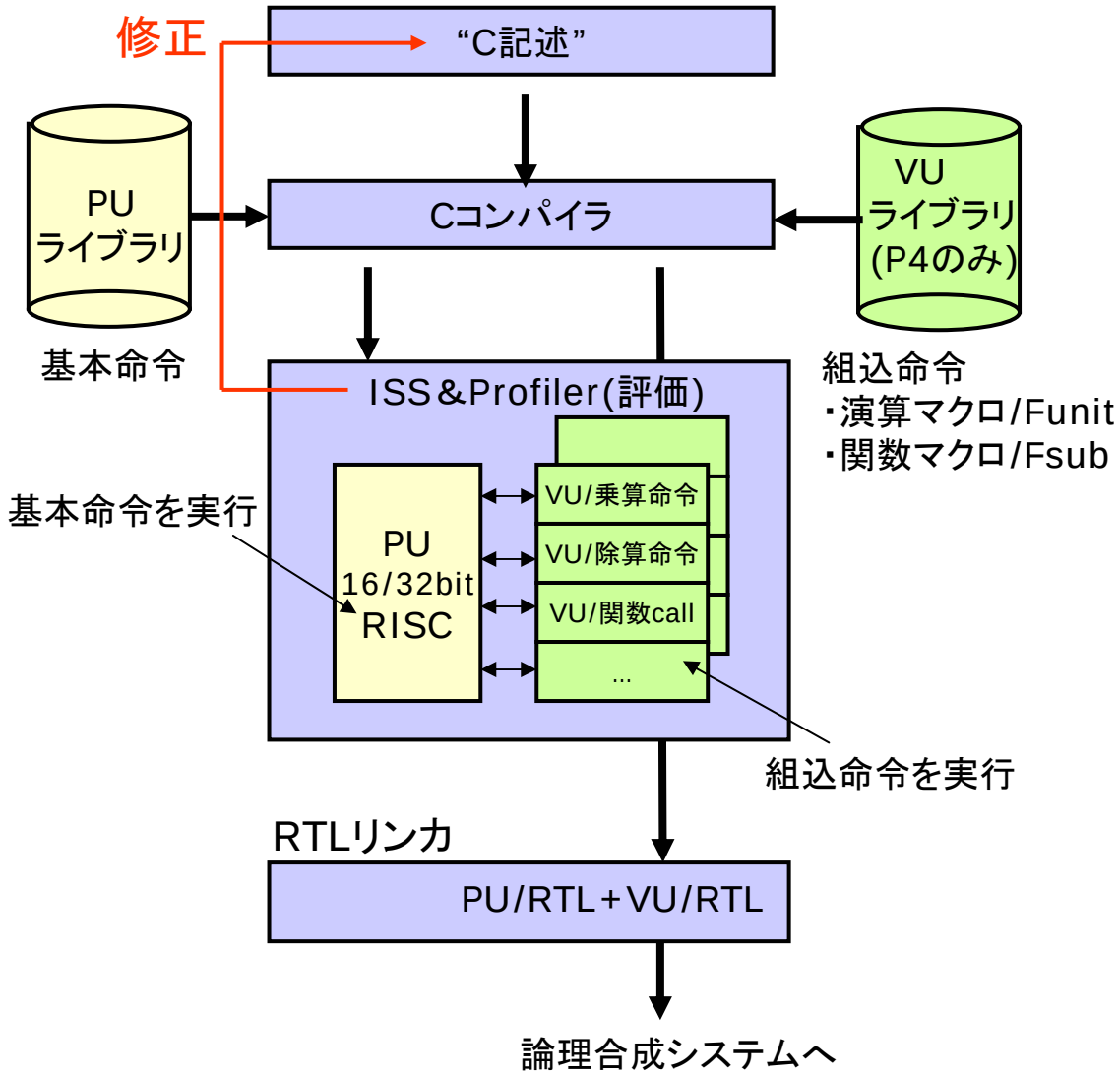
VUPU プロセッサ・コンパイル・システム

ご提供物:

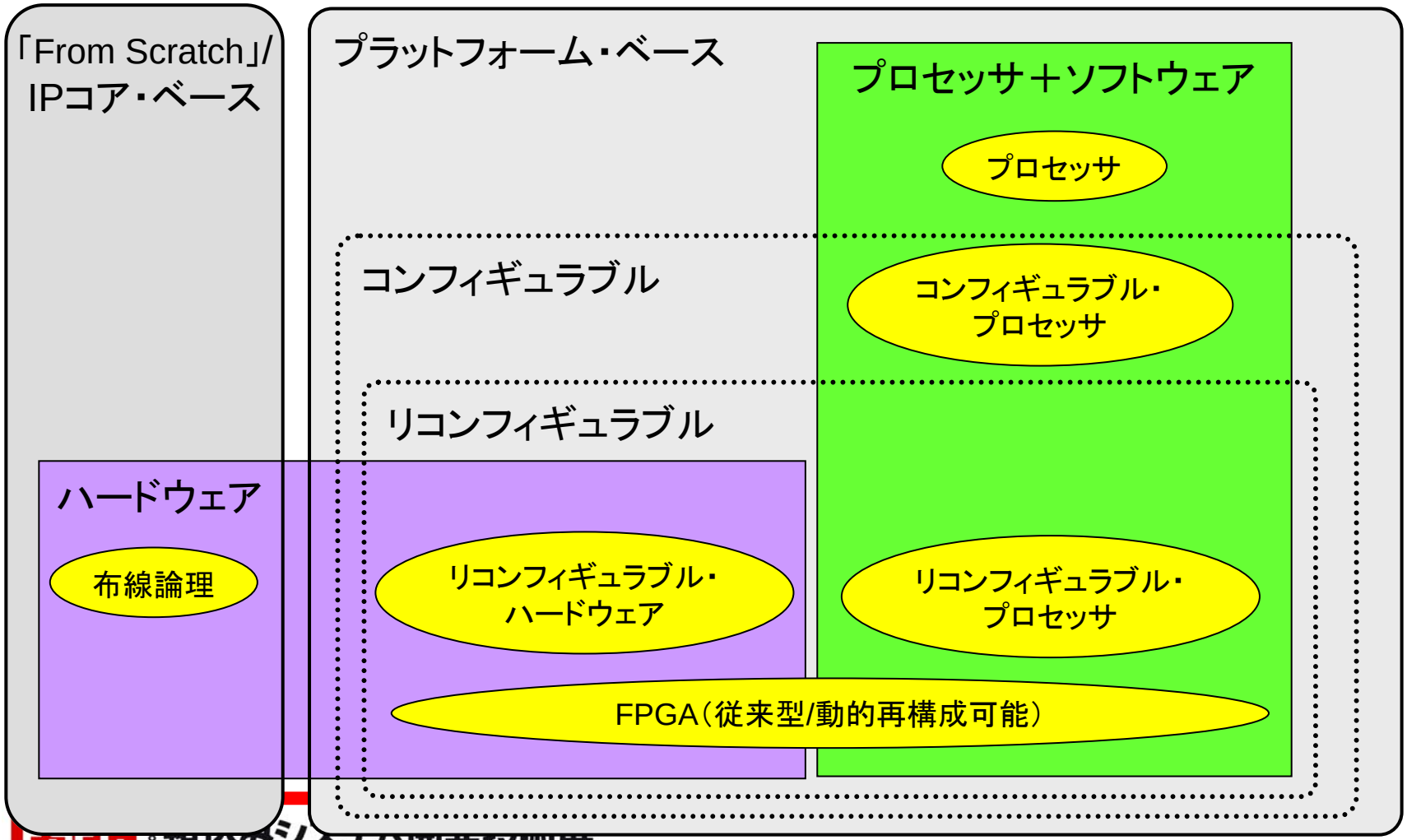
1.Cコンパイラ
2.ISSシミュレータ
3.ISSプロファイラ
4.アセンブラ
5.RTLリンカ
6.デバッガソフト

7.ISSシミュレーション用
Cライブラリ

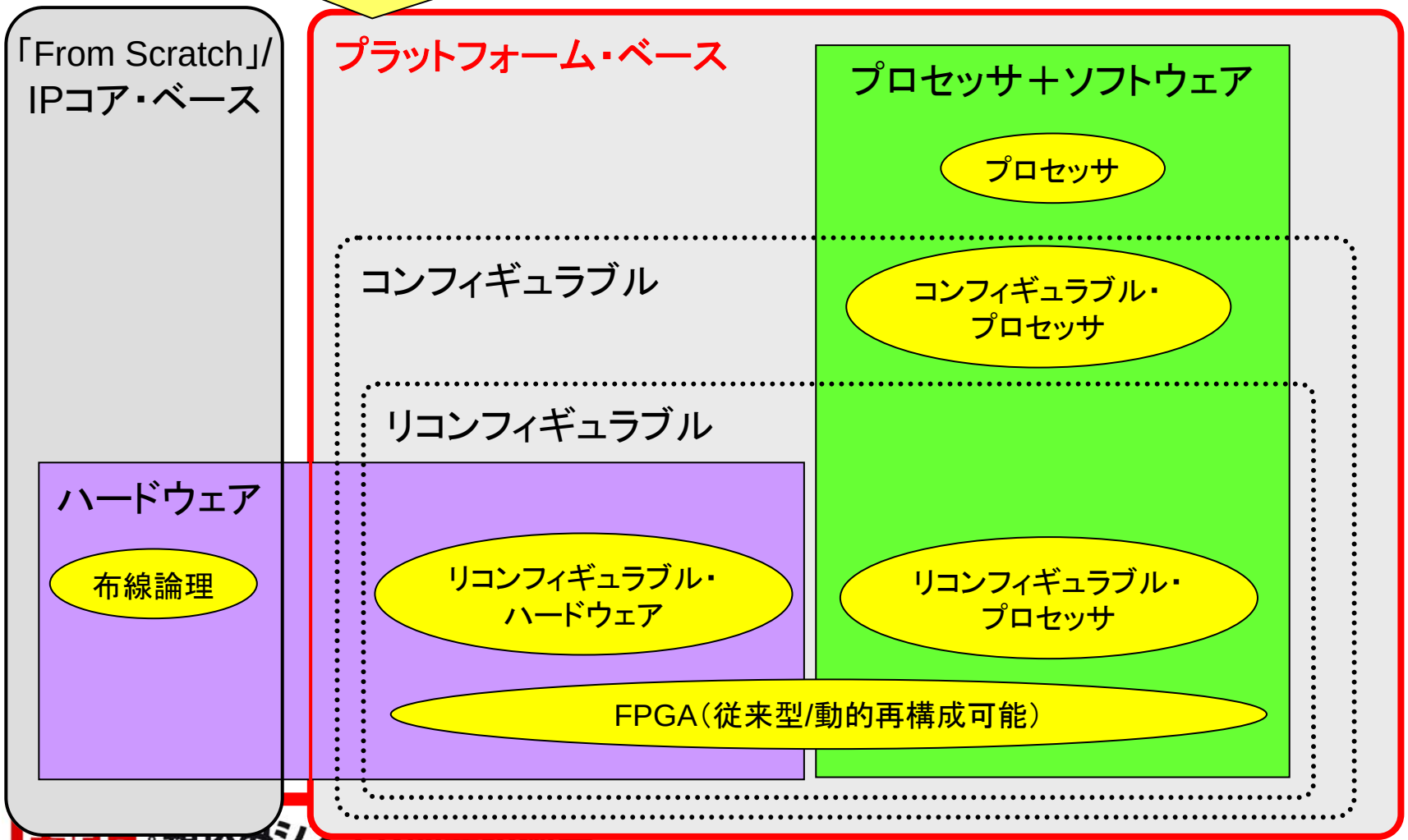
8.プロセッサRTLライブラリ
9.デバッグユニット
RTLライブラリ
10.VU/Funit RTLライブラリ
(P4のみ)
11.VU/Fsub RTLライブラリ
(P4のみ)



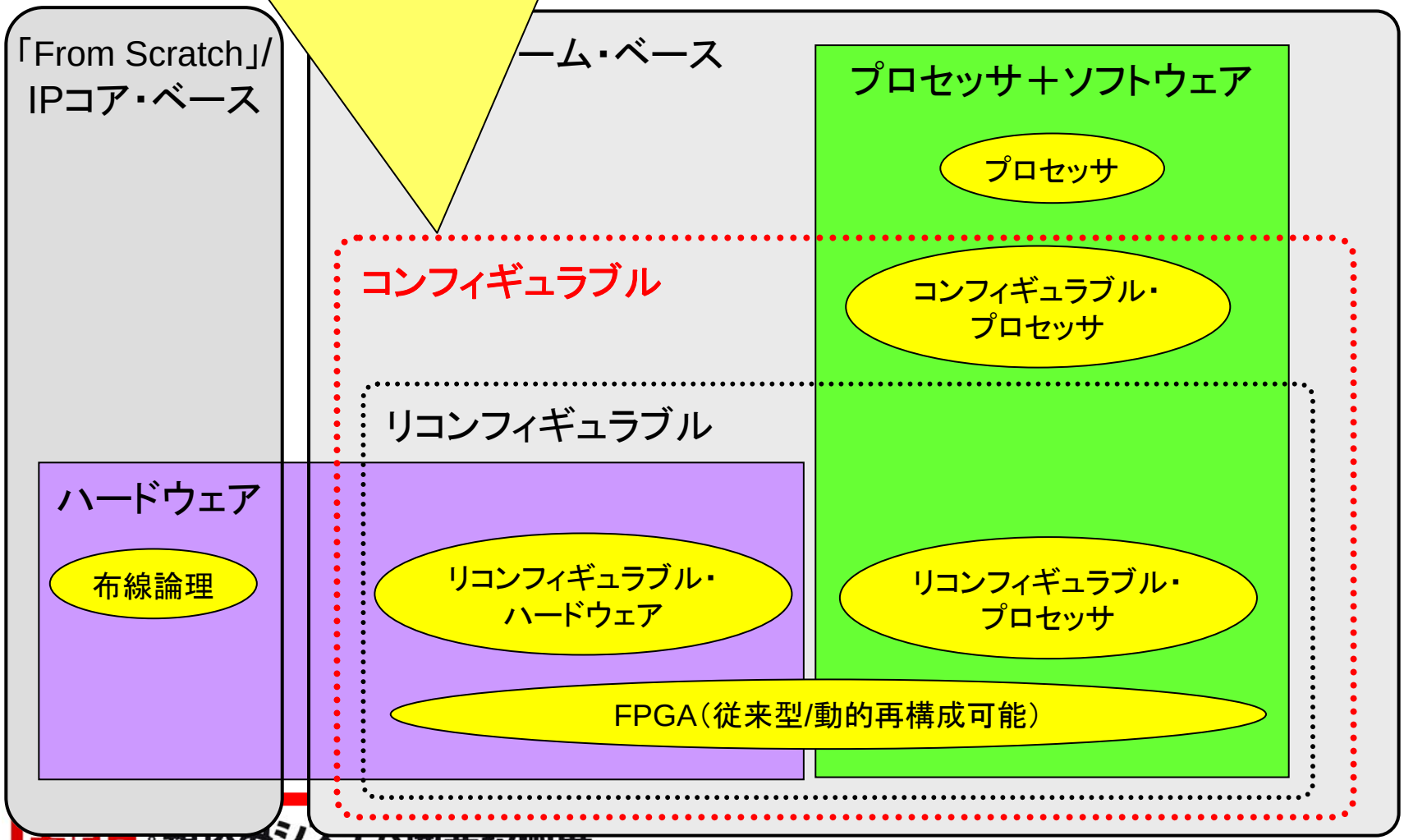
SoC設計への諸アプローチ



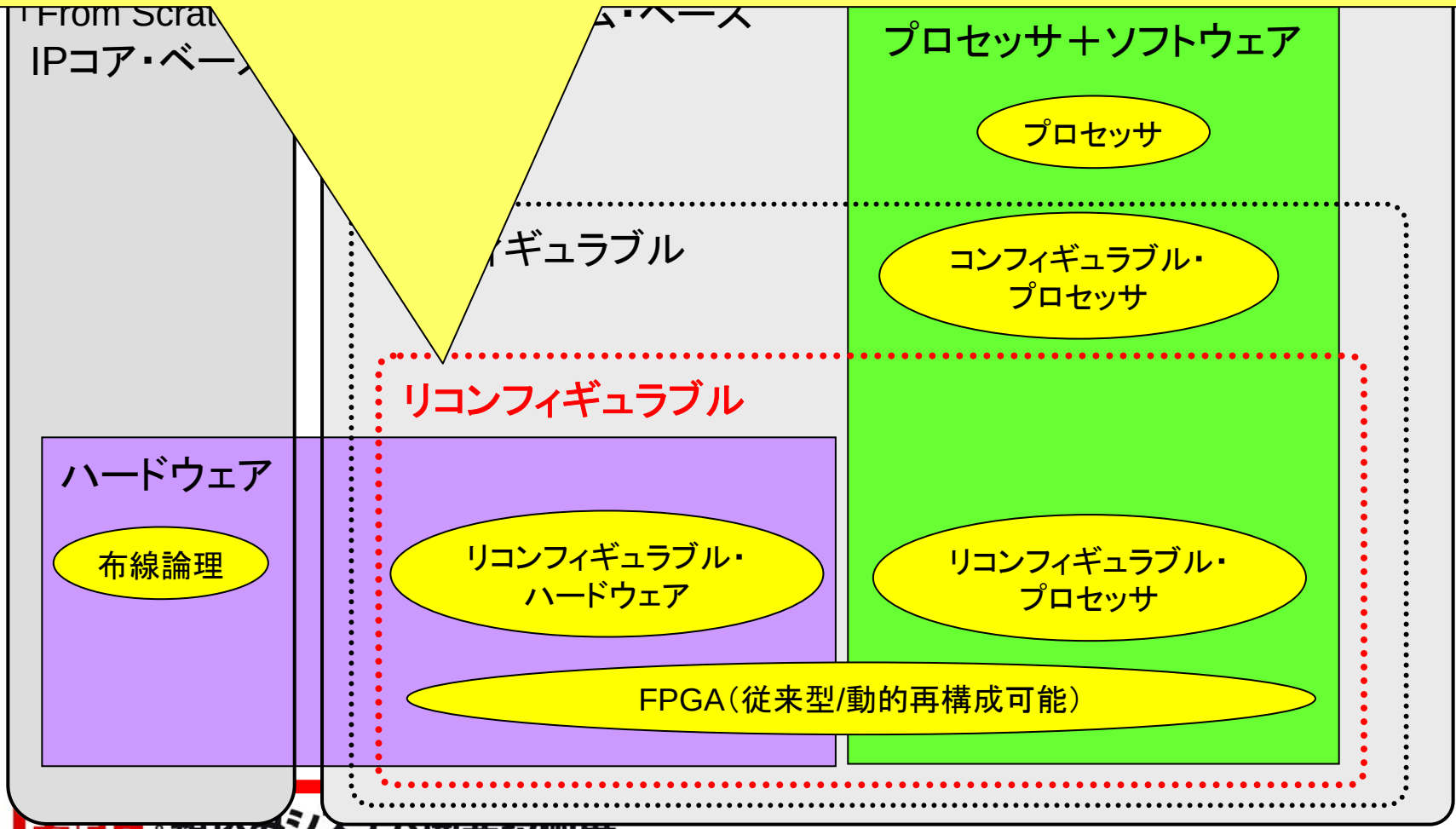
- SoC設計作業を①プラットフォーム実装，および，②その上での専用機能実現に分割！
- プラットフォームを再利用することで，SoC設計作業を②の「専用機能実現」に集中し，SoC設計期間を短縮！
- さらに，プラットフォーム上で多用な専用機能が実現可能なことから，SoC設計の共有＆汎用化が可能→少品種大量生産が可能に！



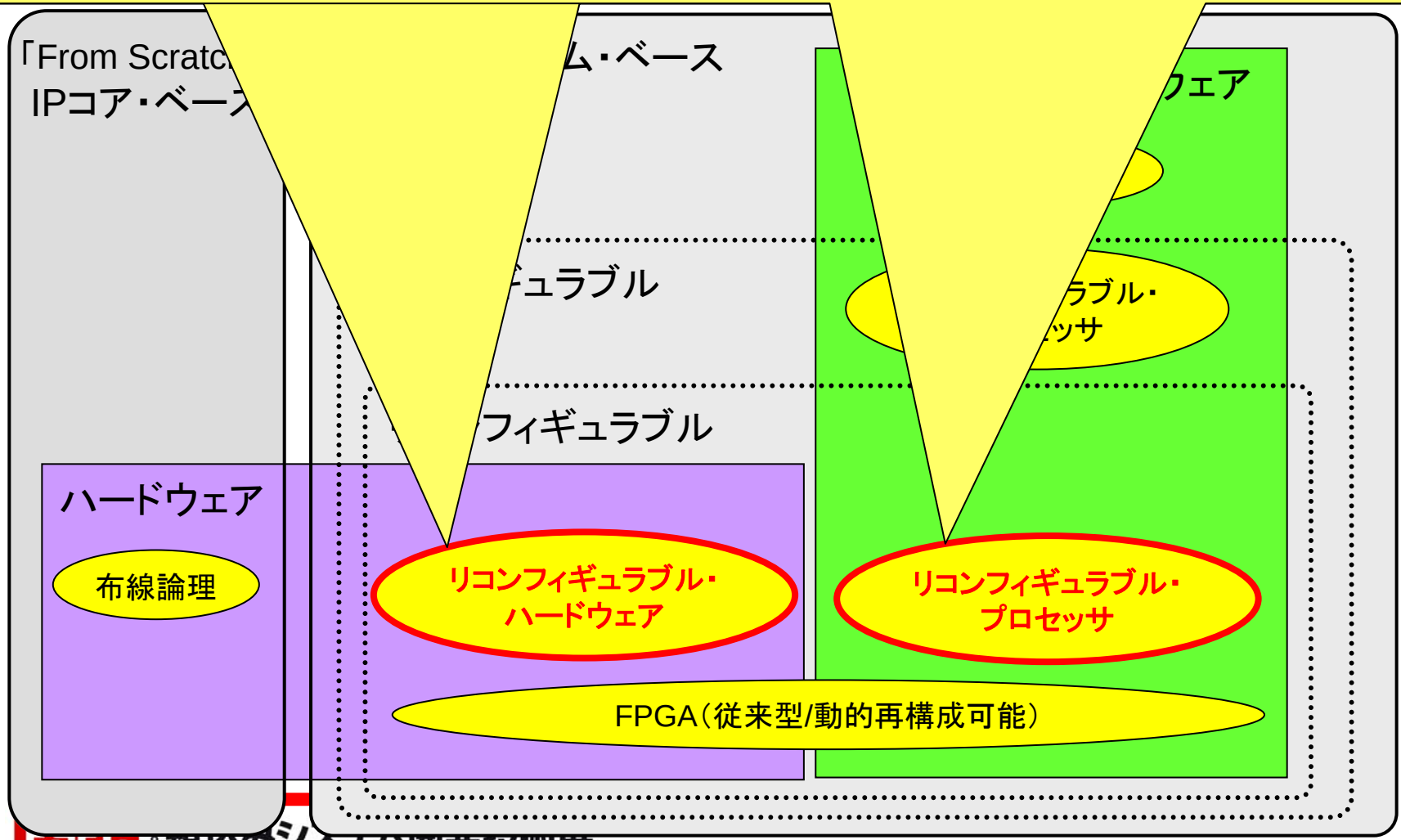
- 実現すべき専用機能に合わせてプラットフォーム（プロセッサ・ロジック）の構成，機能をカスタム化可能！
  ↗従来型プロセッサ/DSPに比べてコスト対性能比が向上
  ↘従来型プロセッサ/DSPに比べて，カスタム化のためにSoC設計期間が増大



- プラットフォーム（ハードウェア・ロジック，または，プロセッサ・ロジック）の構成，機能をカスタム化するタイミングをSoC設計終了後にまで延期可能！
  ↗コンフィギュラブル・プロセッサに比べてSoC設計期間を短縮
- 再構成のためのオーバヘッド（面積，時間）が不可避
- ただし，再構成可能性を活用することで，実装すべき機能を（布線論理や従来型プロセッサのように）ナイーブに実装するより，必要面積を削減することが可能
  – コンフィギュラブル・プロセッサに比べたコスト対性能比は？



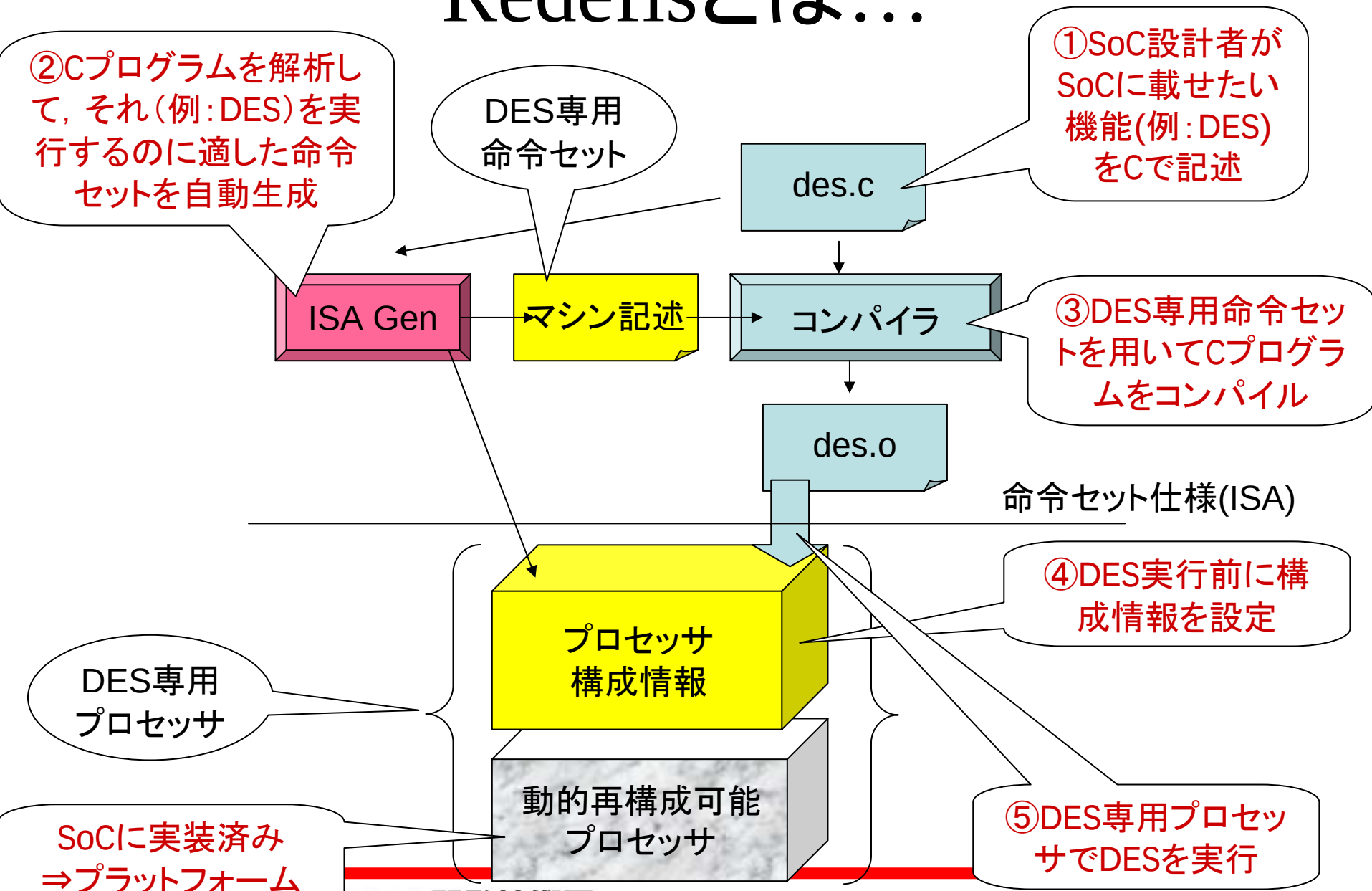
- コスト対性能比：
  ＋リコンフィギュラブル・ハードウェア・ロジック
  －リコンフィギュラブル・プロセッサ
- 設計生産性，Cレベル設計との親和性：
  －リコンフィギュラブル・ハードウェア・ロジック
  ＋リコンフィギュラブル・プロセッサ



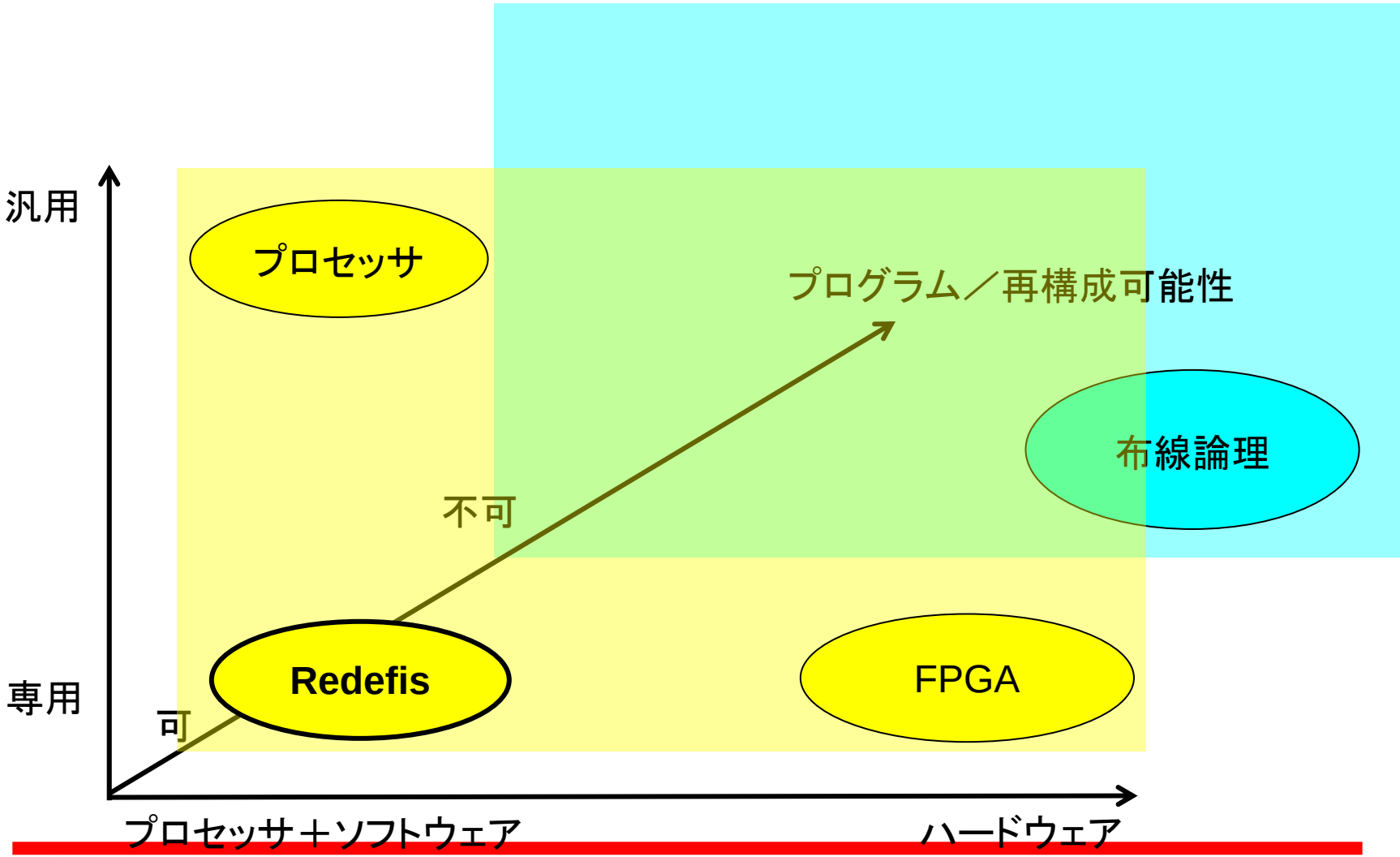
目次

- SoC設計への諸アプローチ
  - カスタム・ロジックの実現方法
    - プロセッサ＋ソフトウェア
    - コンフィギュラブル・プロセッサ＋ソフトウェア
    - リコンフィギュラブル・プロセッサ＋ソフトウェア
    - リコンフィギュラブル・ハードウェア
- ダイナミック・リコンフィギュラブル・プロセッサの具体例
  - CLUSS Redefis

Redefisとは…

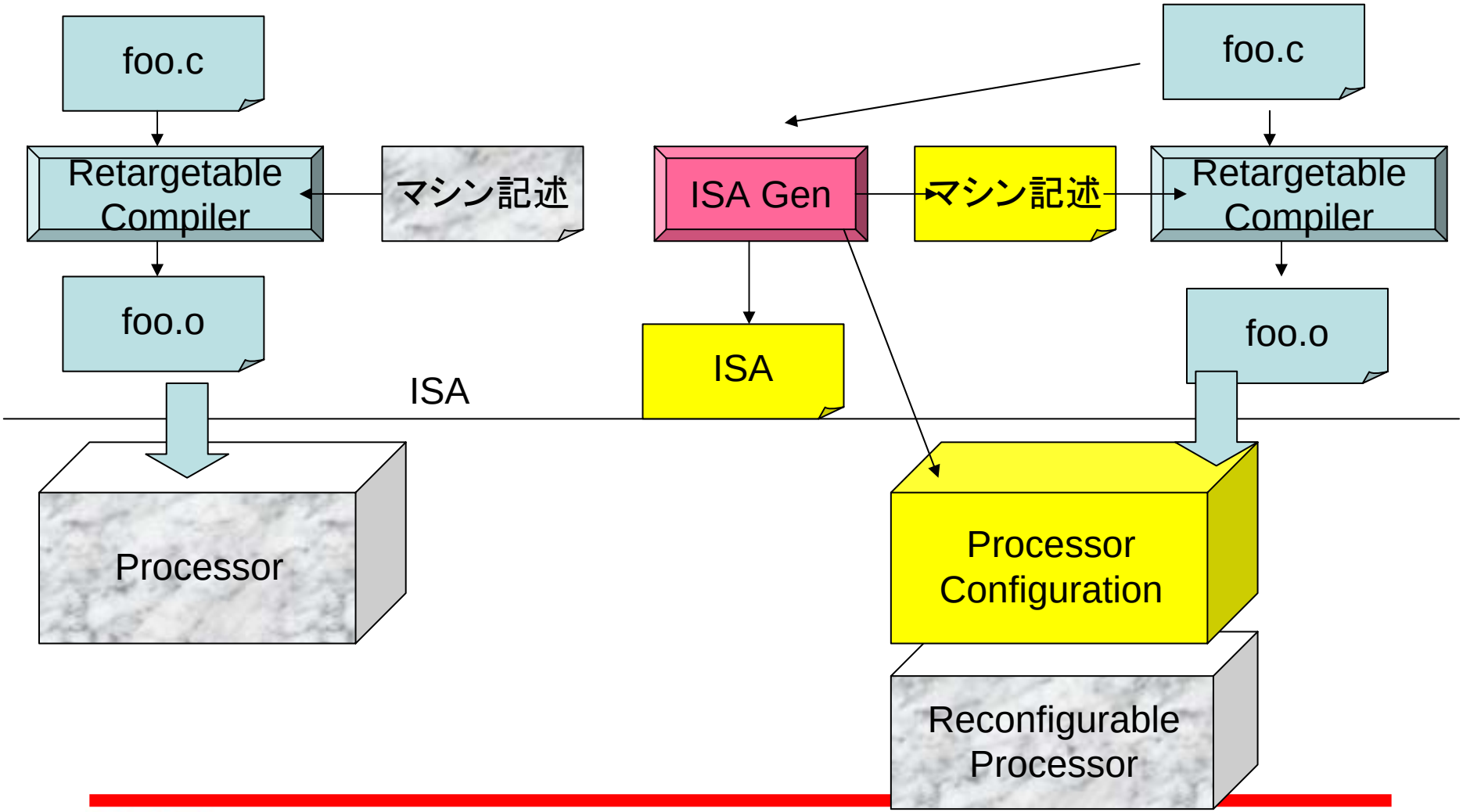

②Cプログラムを解析して，それ（例：DES）を実行するのに適した命令セットを自動生成
DES専用
命令セット
①SoC設計者がSoCに載せたい機能(例：DES)をCで記述
des.c
ISA Gen
マシン記述
コンパイラ
③DES専用命令セットを用いてCプログラムをコンパイル
des.o
命令セット仕様(ISA)
④DES実行前に構成情報を設定
プロセッサ
構成情報
DES専用
プロセッサ
動的再構成可能
プロセッサ
SoCに実装済み
⇒プラットフォーム
⑤DES専用プロセッサでDESを実行

Redefisの位置付け



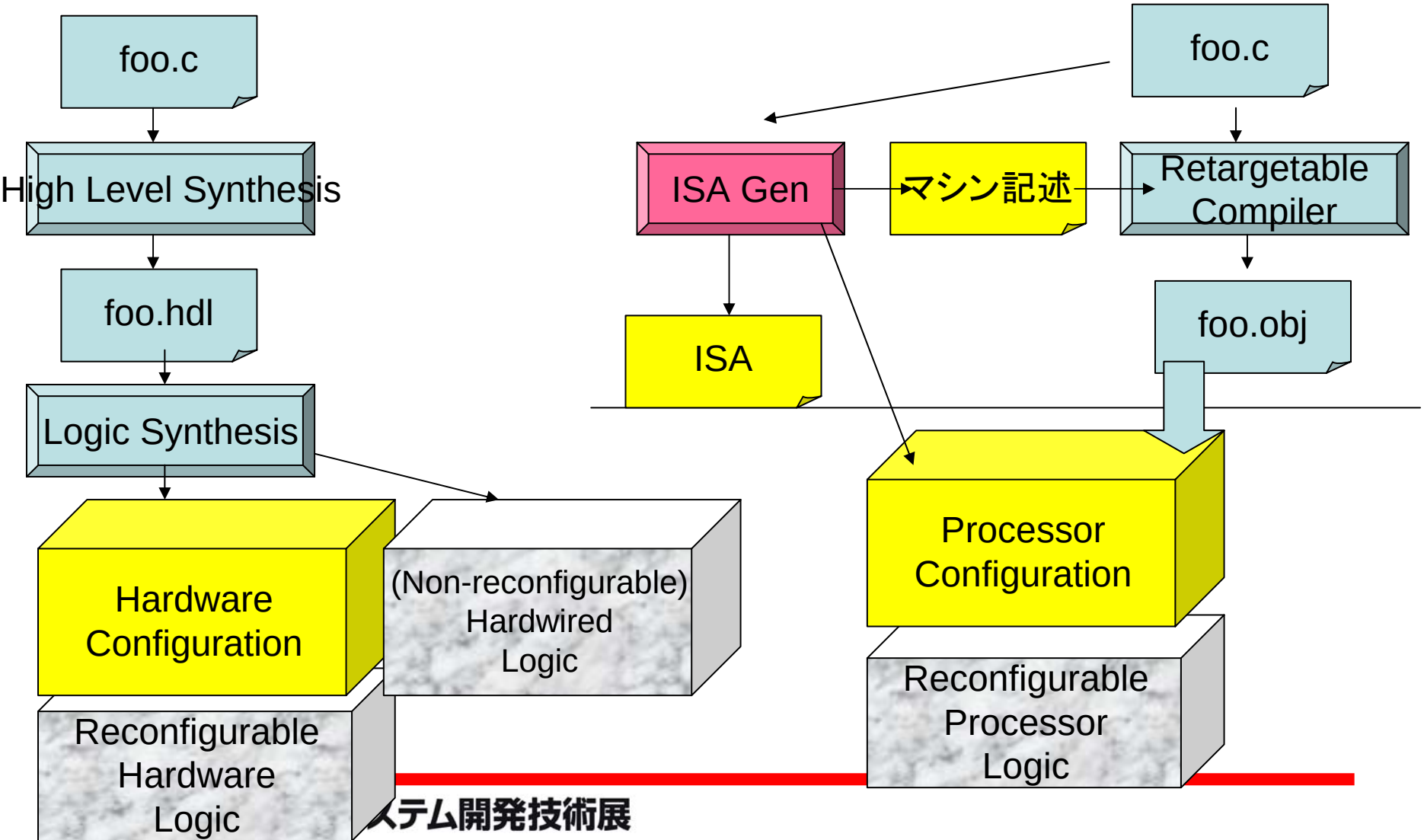
従来型プロセッサ vs Redefis

- 従来プロセッサ上でのソフトウェア開発フロー
- Redefis上での「プロセッサ構成情報＋ソフトウェア」開発フロー



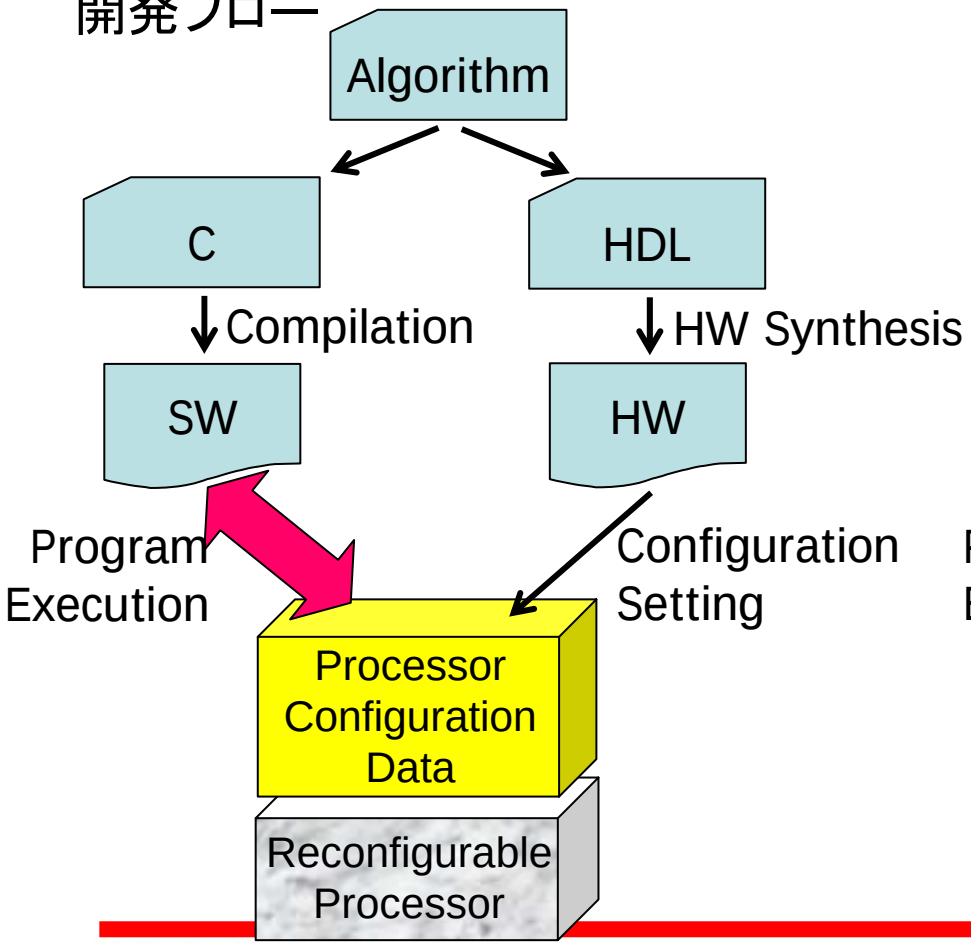
ハードウェア vs Redefis

- 従来のハードウェア設計
- Redefis上での「プロセッサ構成情報＋ソフトウェア」開発フロー

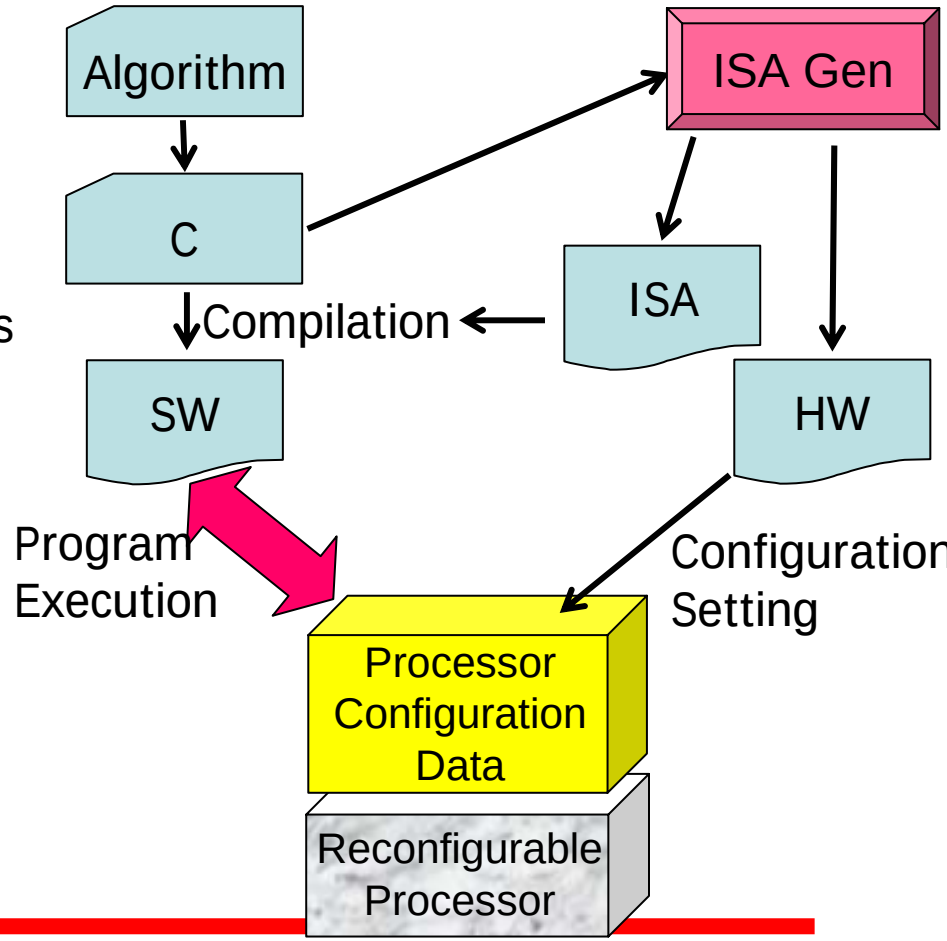


通常の動的再構成可能プロセッサ vs. Redefis

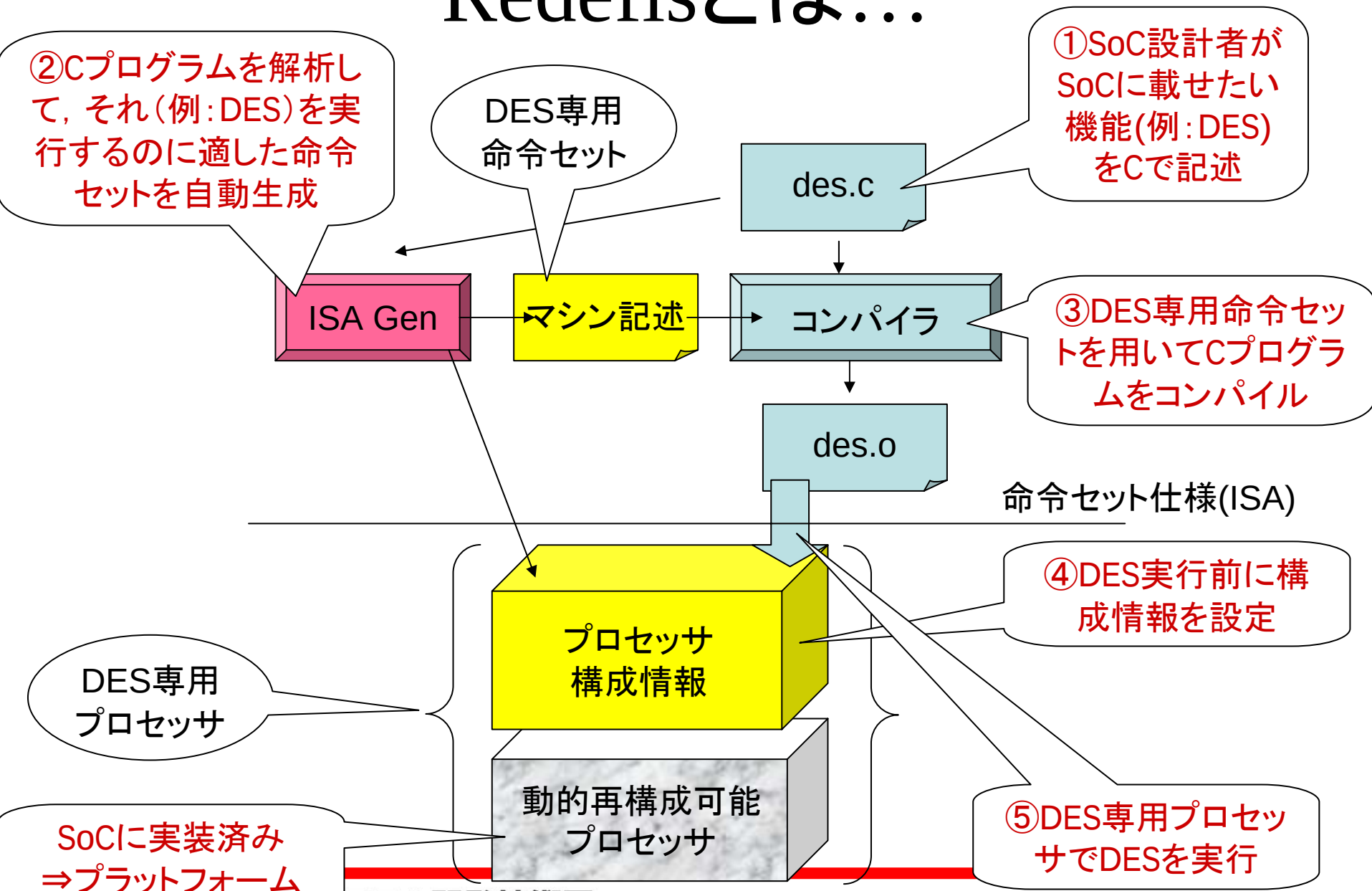
- 通常の動的再構成可能プロセッサ上での「プロセッサ構成情報＋ソフトウェア」開発フロー



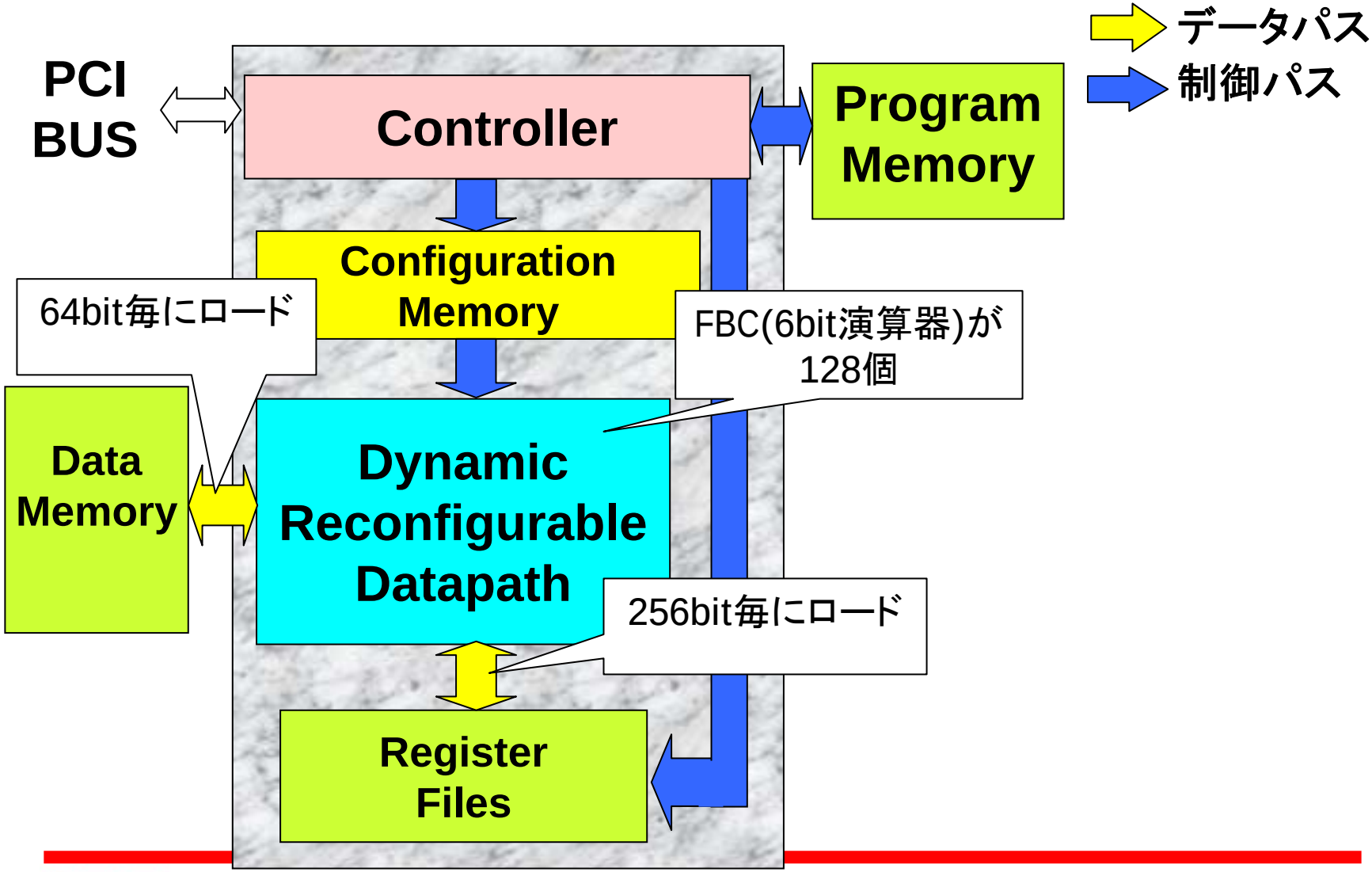
- Redefis上での「プロセッサ構成情報＋ソフトウェア」開発フロー



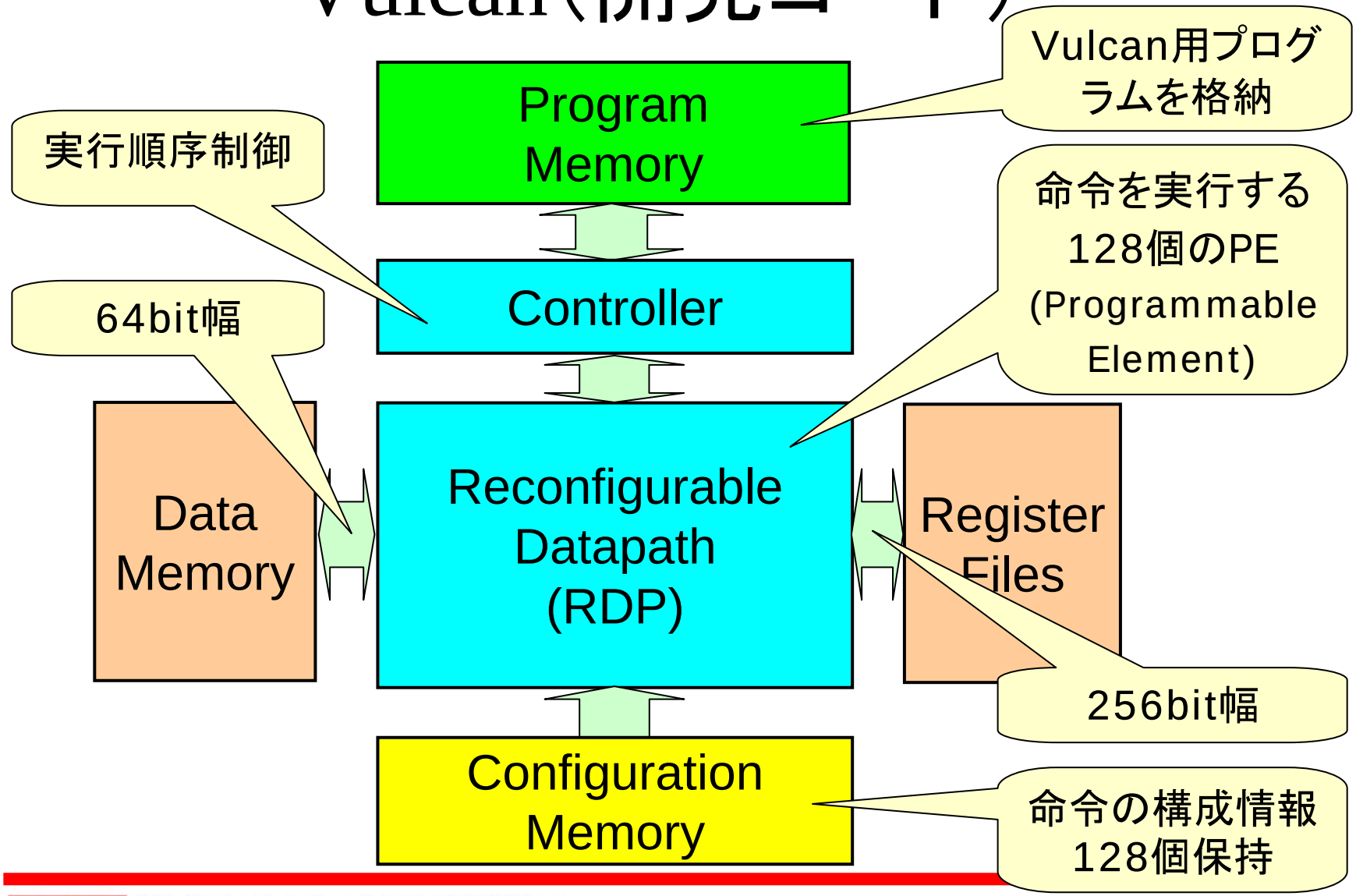
Redefisとは…


②Cプログラムを解析して，それ（例：DES）を実行するのに適した命令セットを自動生成
DES専用
命令セット
①SoC設計者がSoCに載せたい機能(例：DES)をCで記述
des.c
ISA Gen
マシン記述
コンパイラ
③DES専用命令セットを用いてCプログラムをコンパイル
des.o
命令セット仕様(ISA)
④DES実行前に構成情報を設定
プロセッサ
構成情報
DES専用
プロセッサ
動的再構成可能
プロセッサ
SoCに実装済み
⇒プラットフォーム
⑤DES専用プロセッサでDESを実行

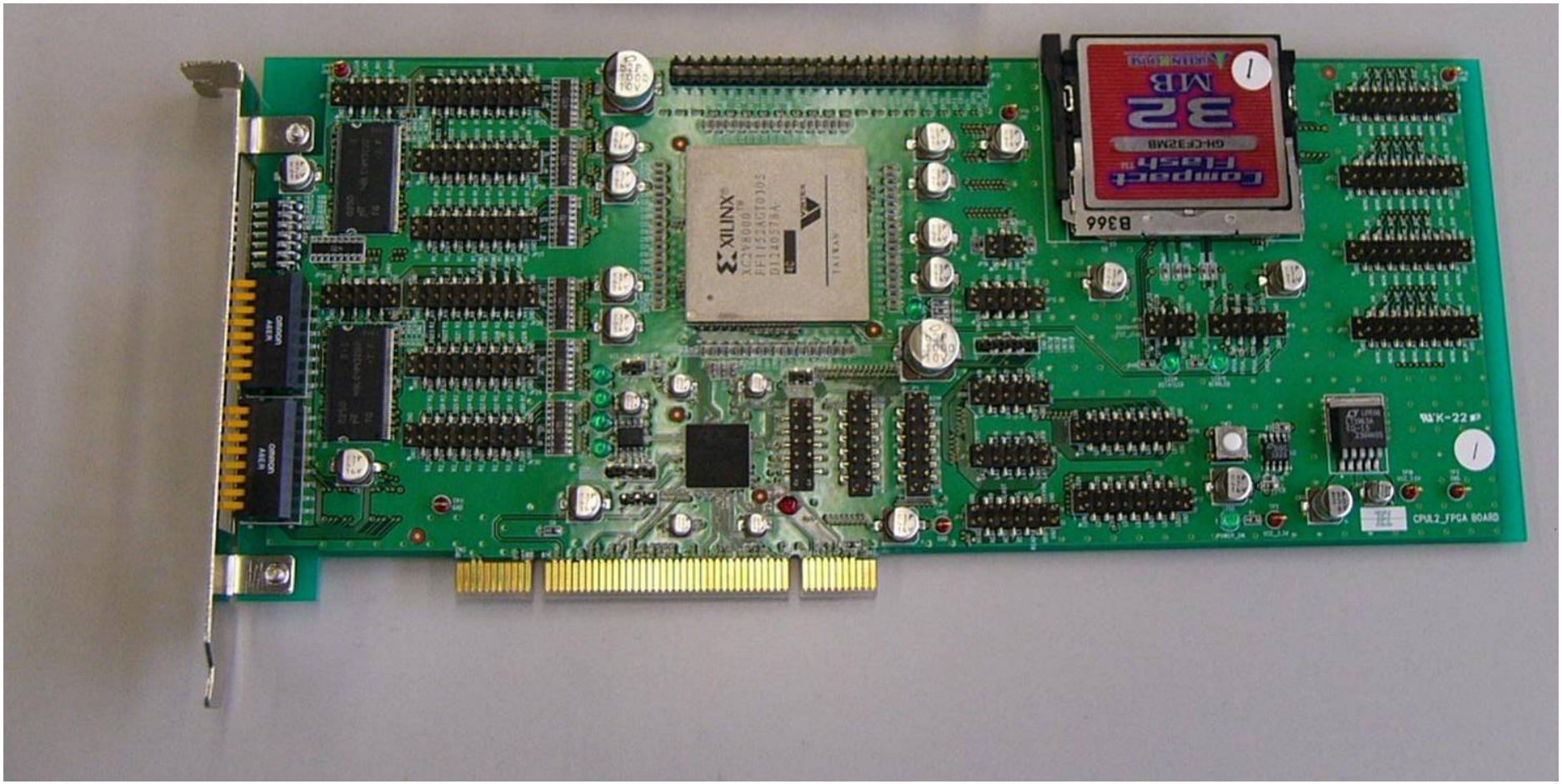
Redefis向け動的再構成可能プロセッサ ～Vulcan（開発コード）～



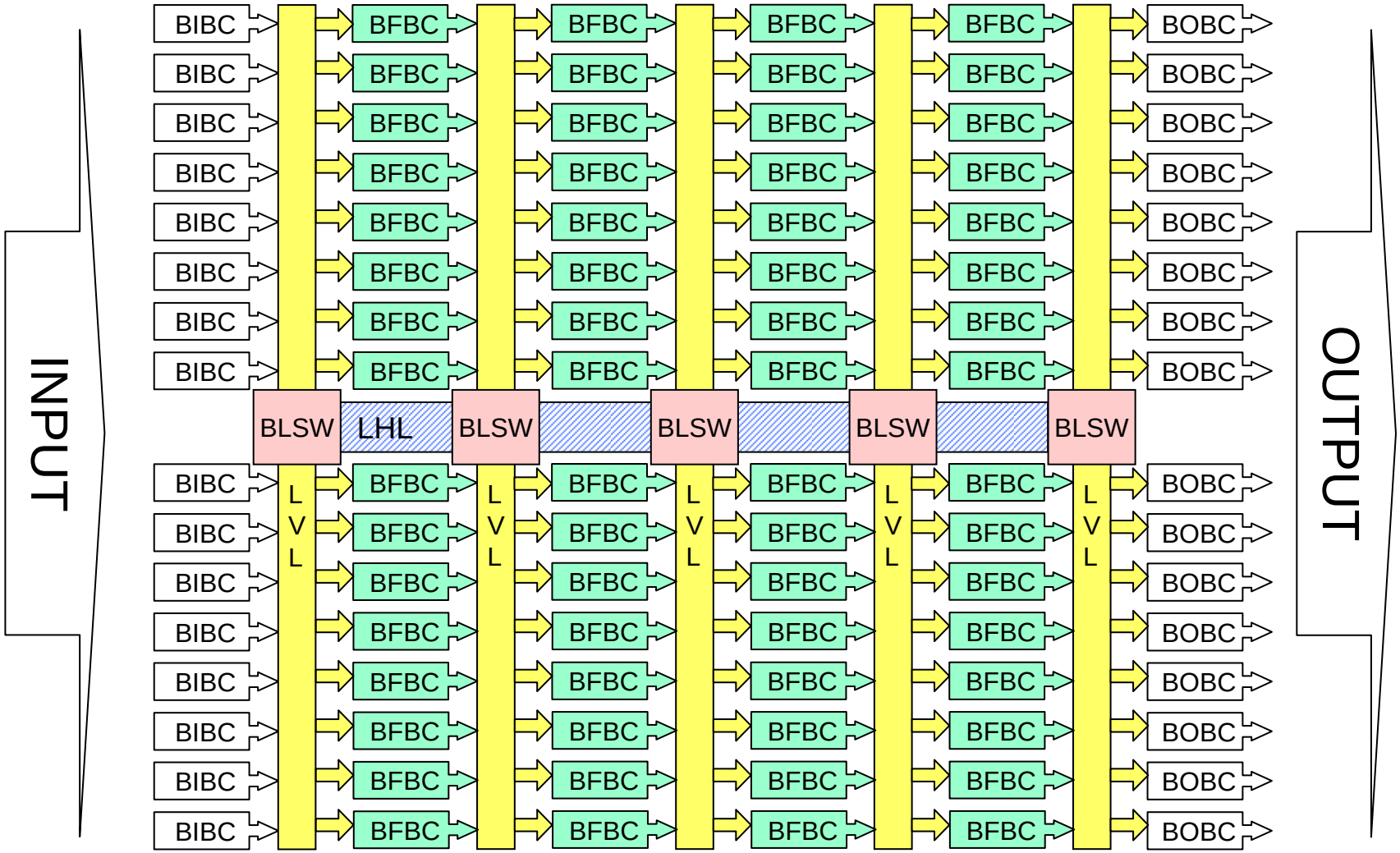
Redefis向け動的再構成可能プロセッサ ～Vulcan（開発コード）～



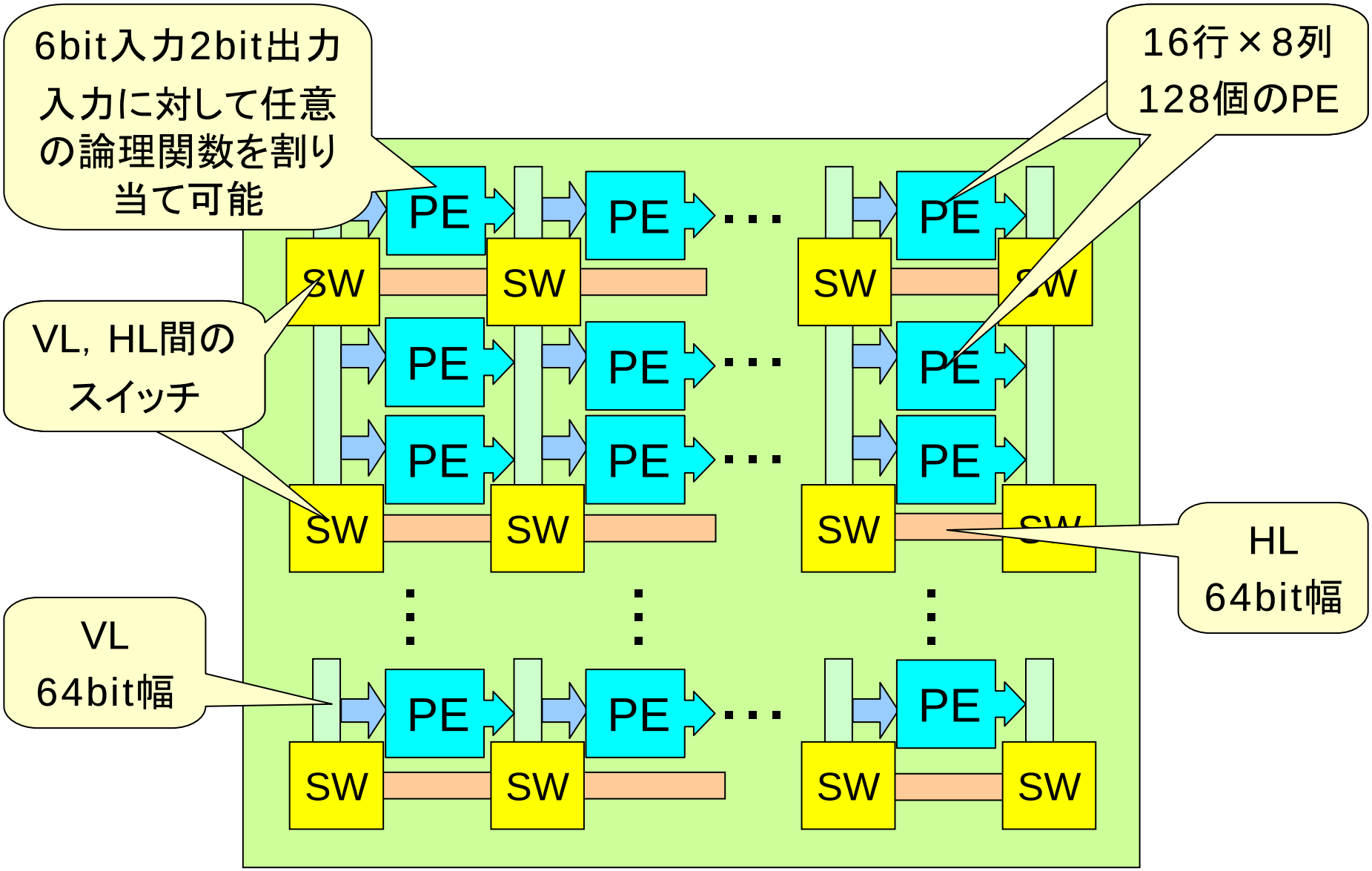
Vulcanを実装したFPGAボード



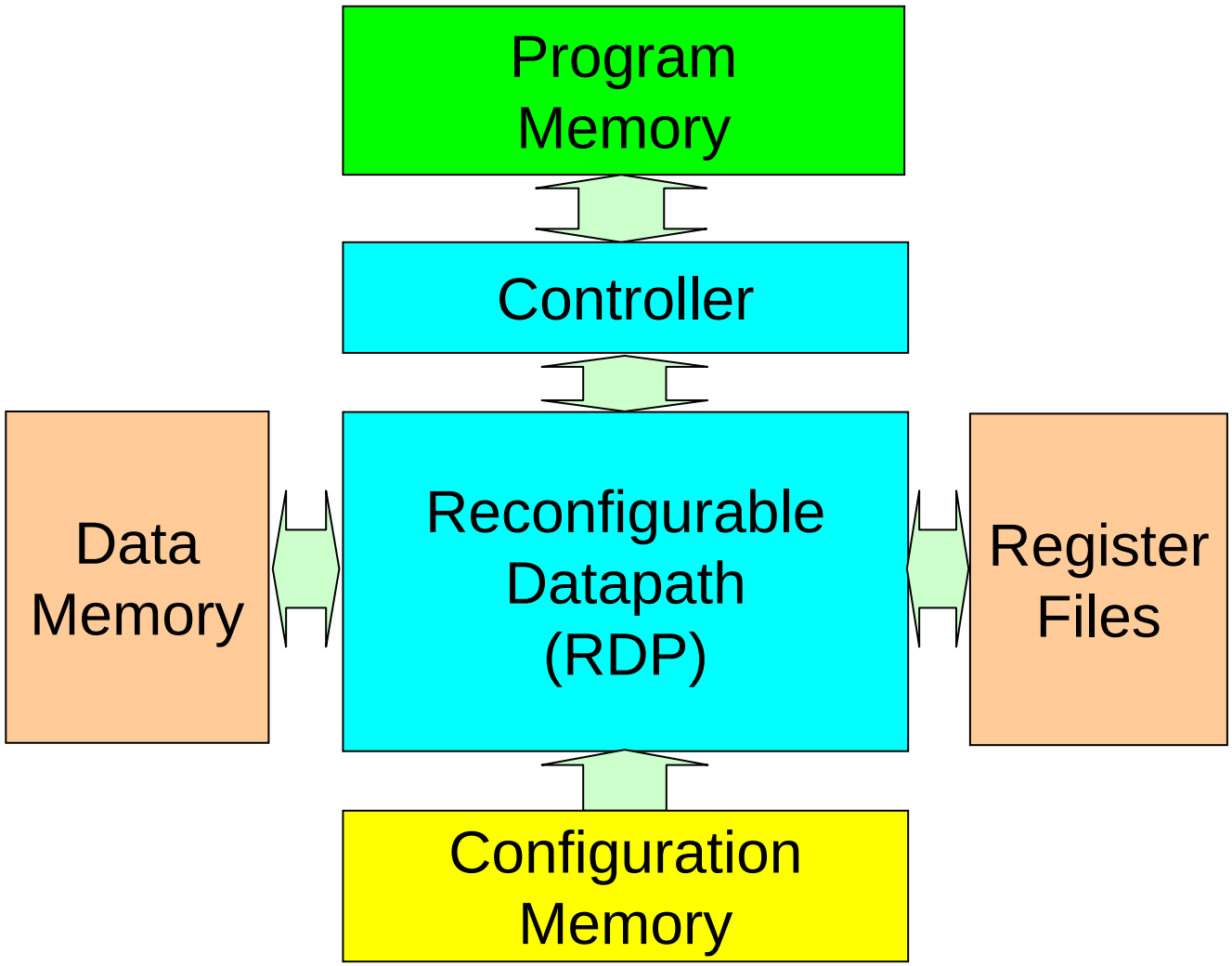
Reconfigurable Datpath (RDP)



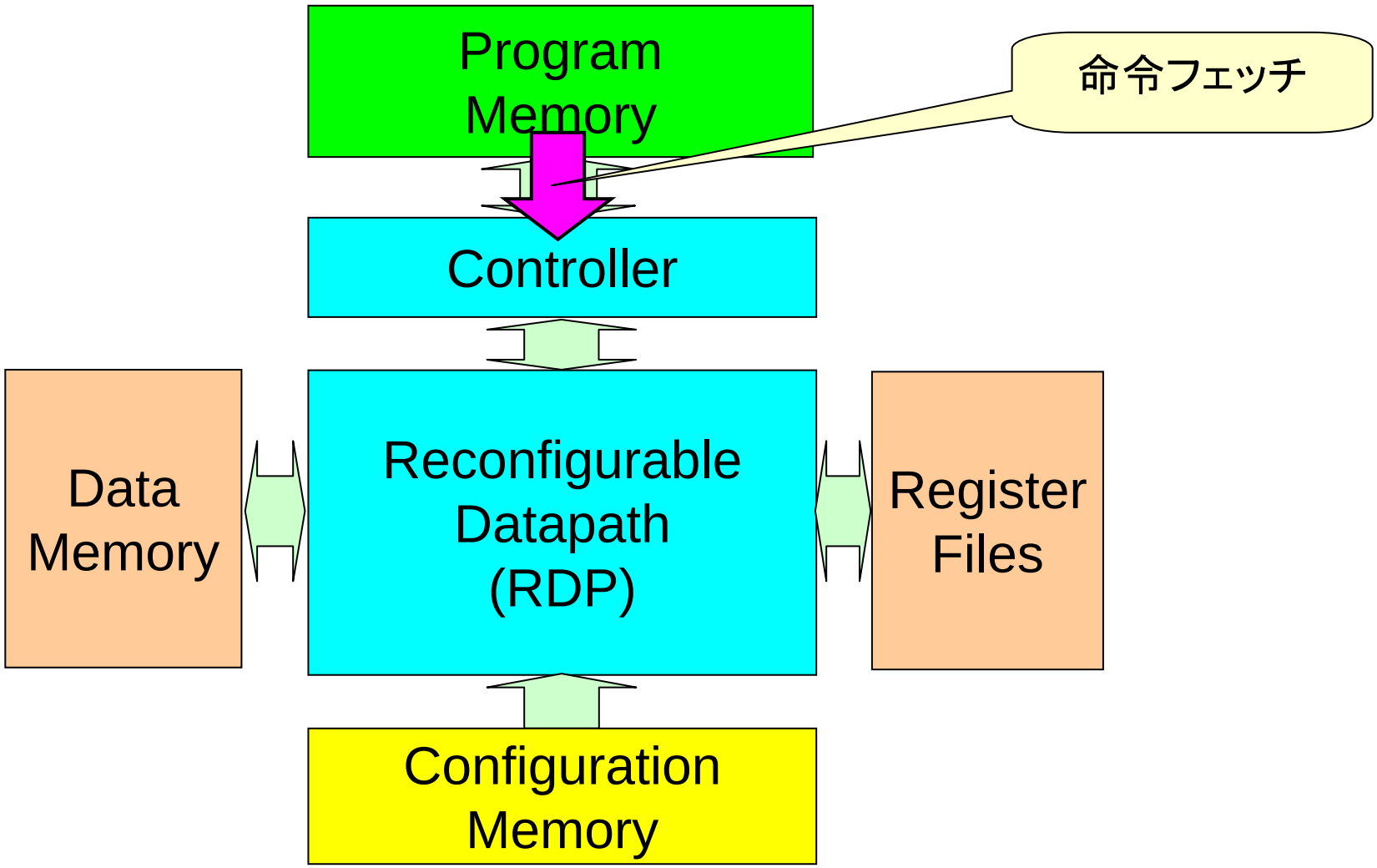
RDP拡大図


6bit入力2bit出力
入力に対して任意の論理関数を割り当て可能
16行×8列
128個のPE
VL，HL間のスイッチ
HL
64bit幅
VL
64bit幅
PE
SW

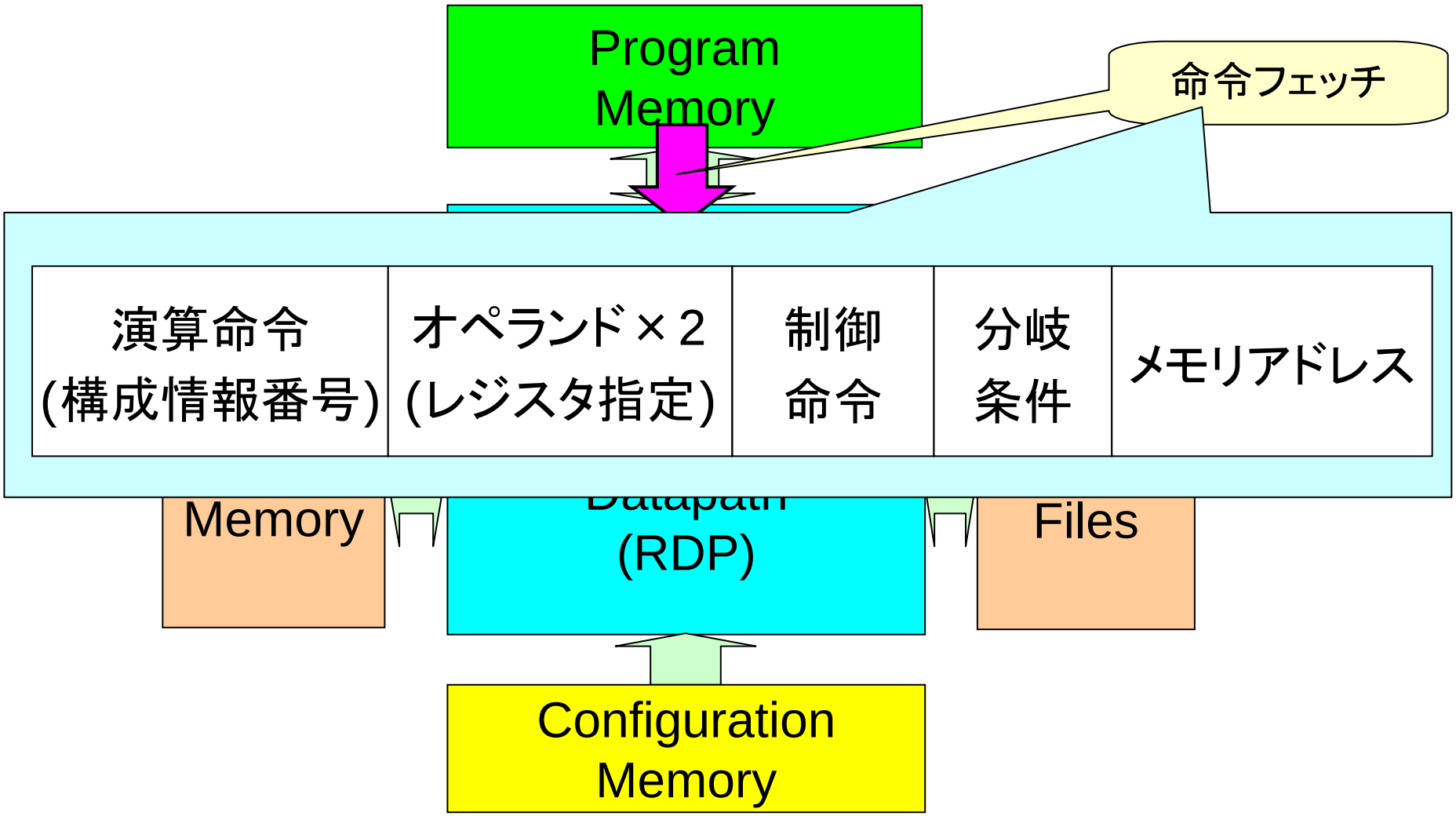
Vulcan実行フロー



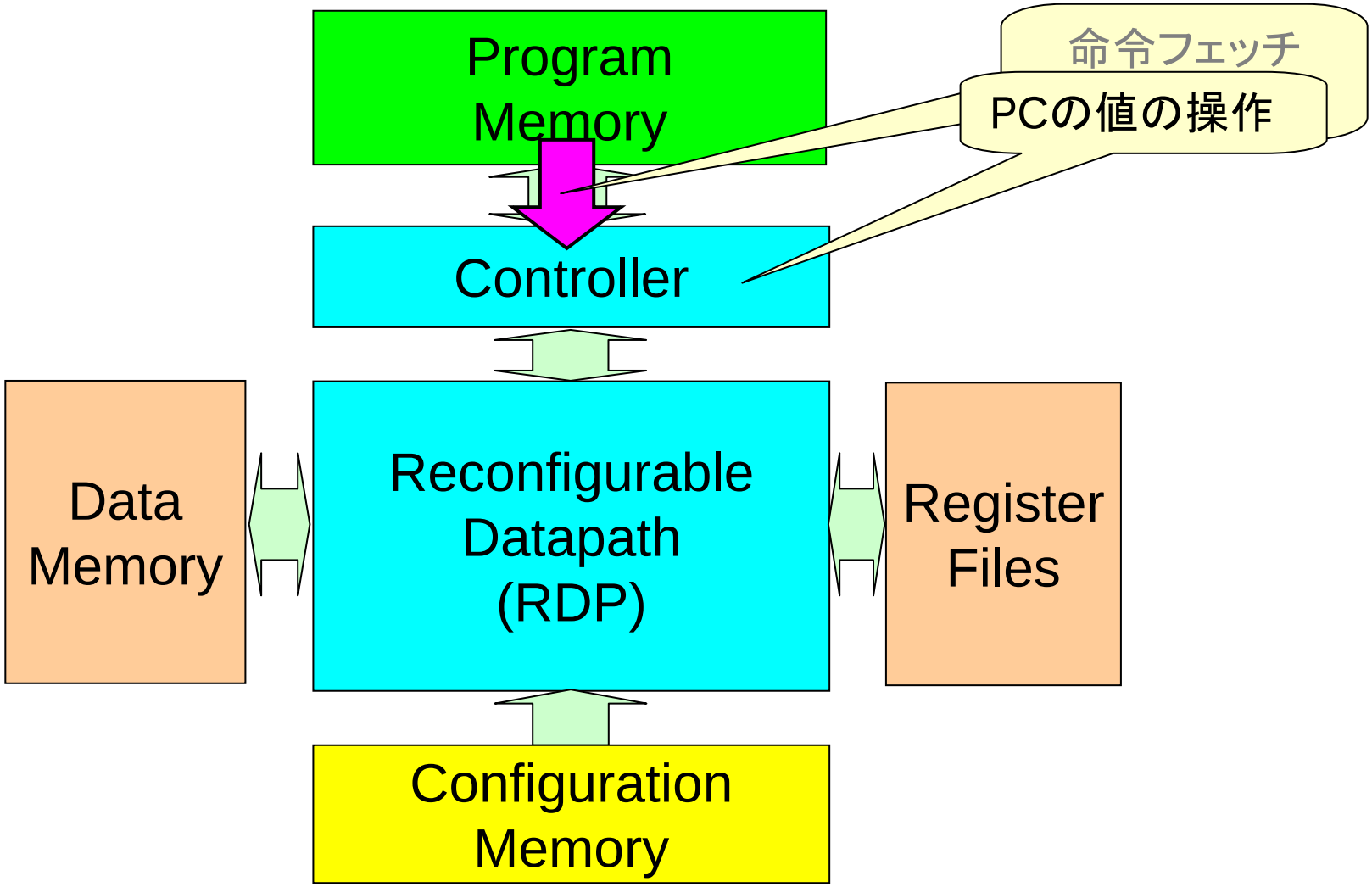
Vulcan実行フロー①
～命令フェッチ～



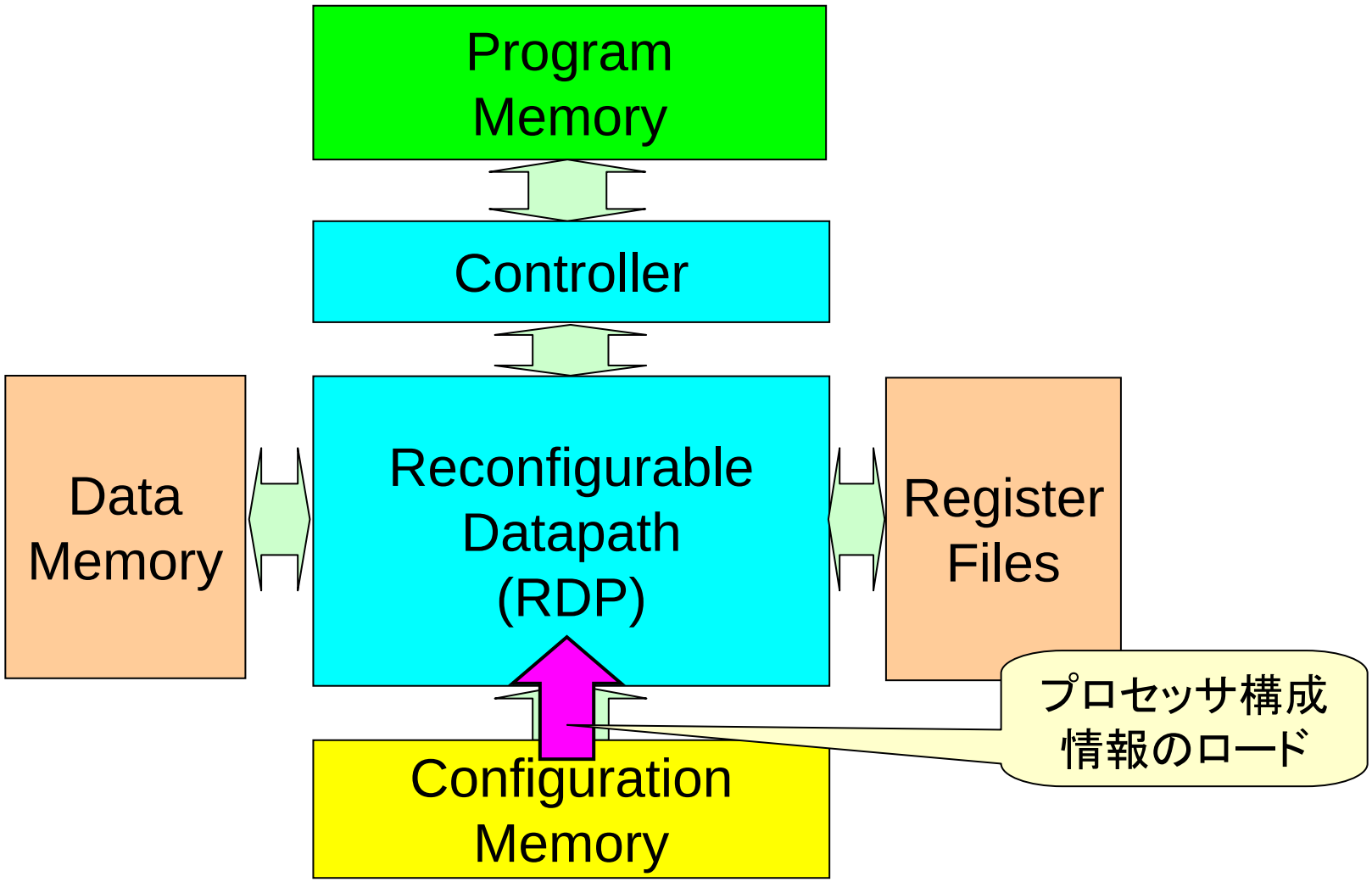
Vulcan実行フロー①
～命令フォーマット～



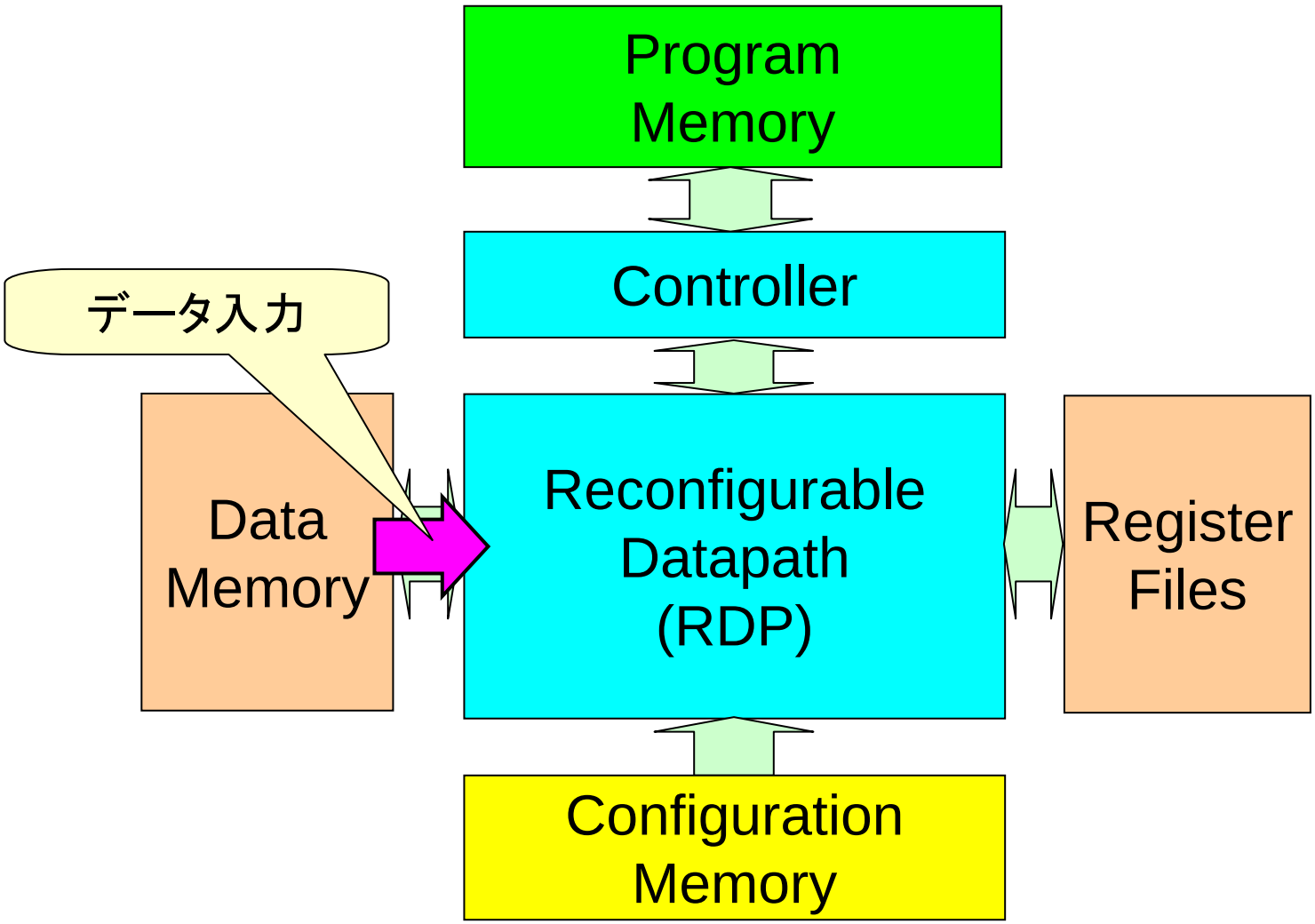
Vulcan実行フロー①
～プログラムカウンタPC更新～



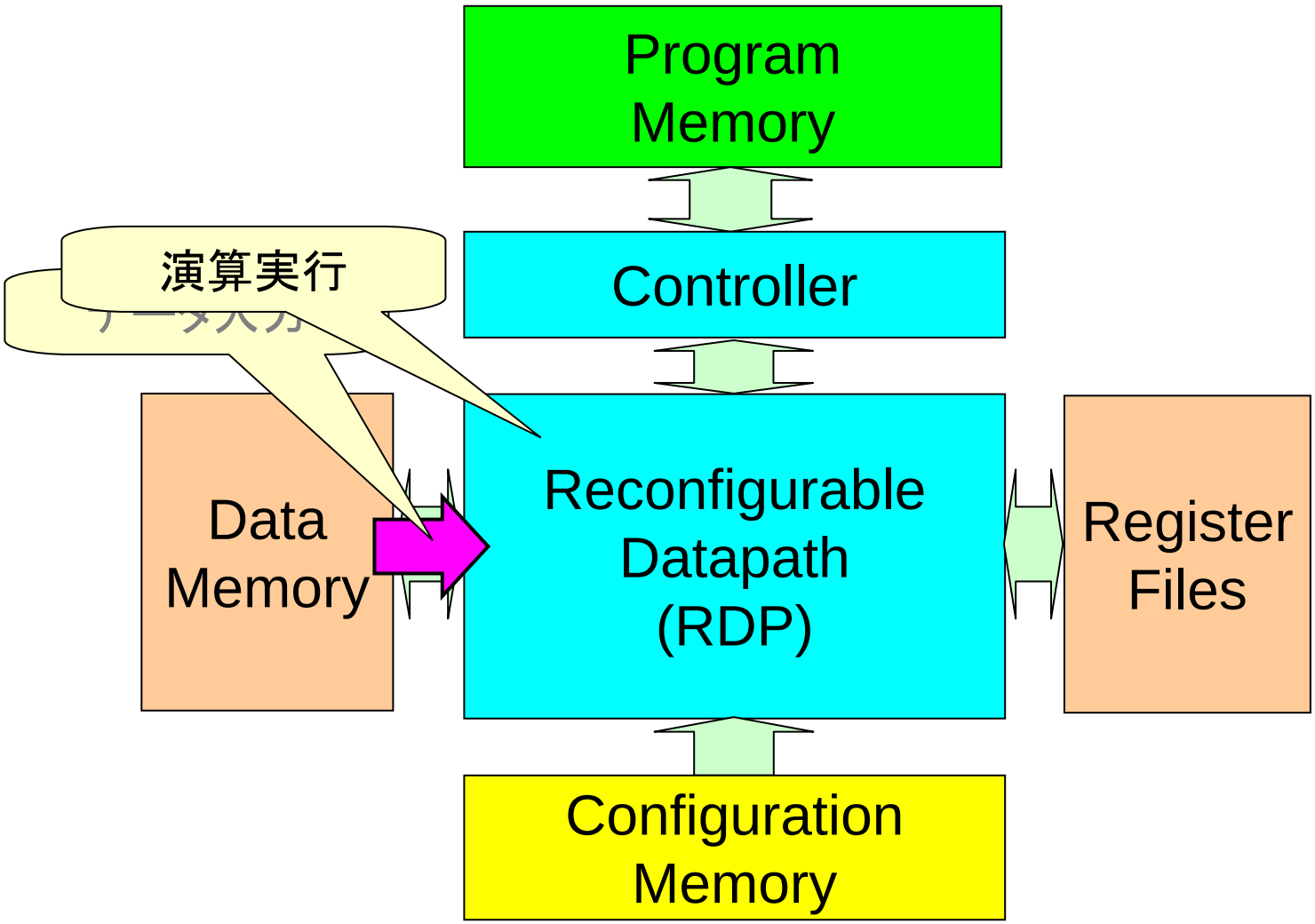
Vulcan実行フロー②
～プロセッサ構成情報のロード～



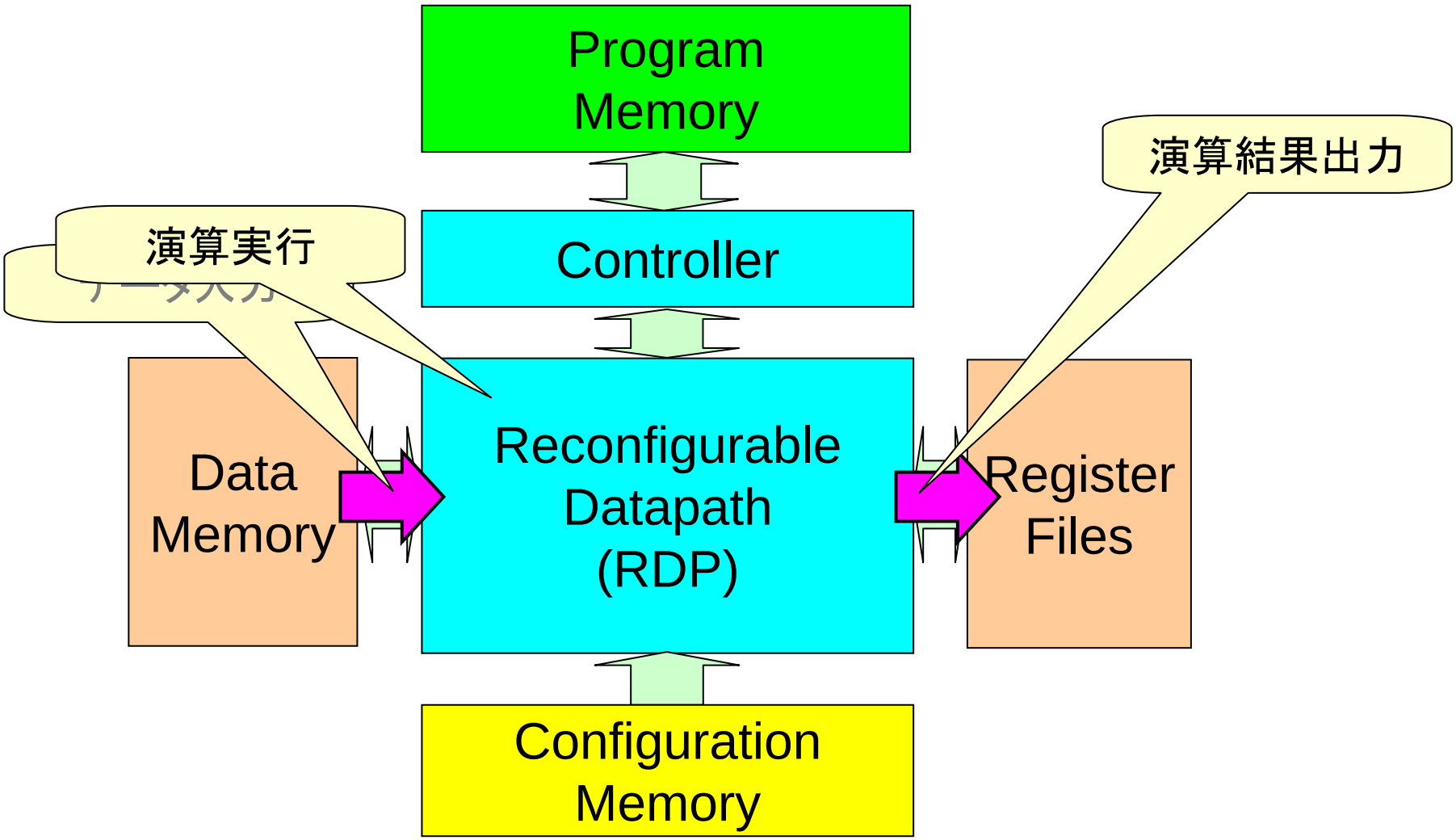
Vulcan実行フロー③
～データ入力～



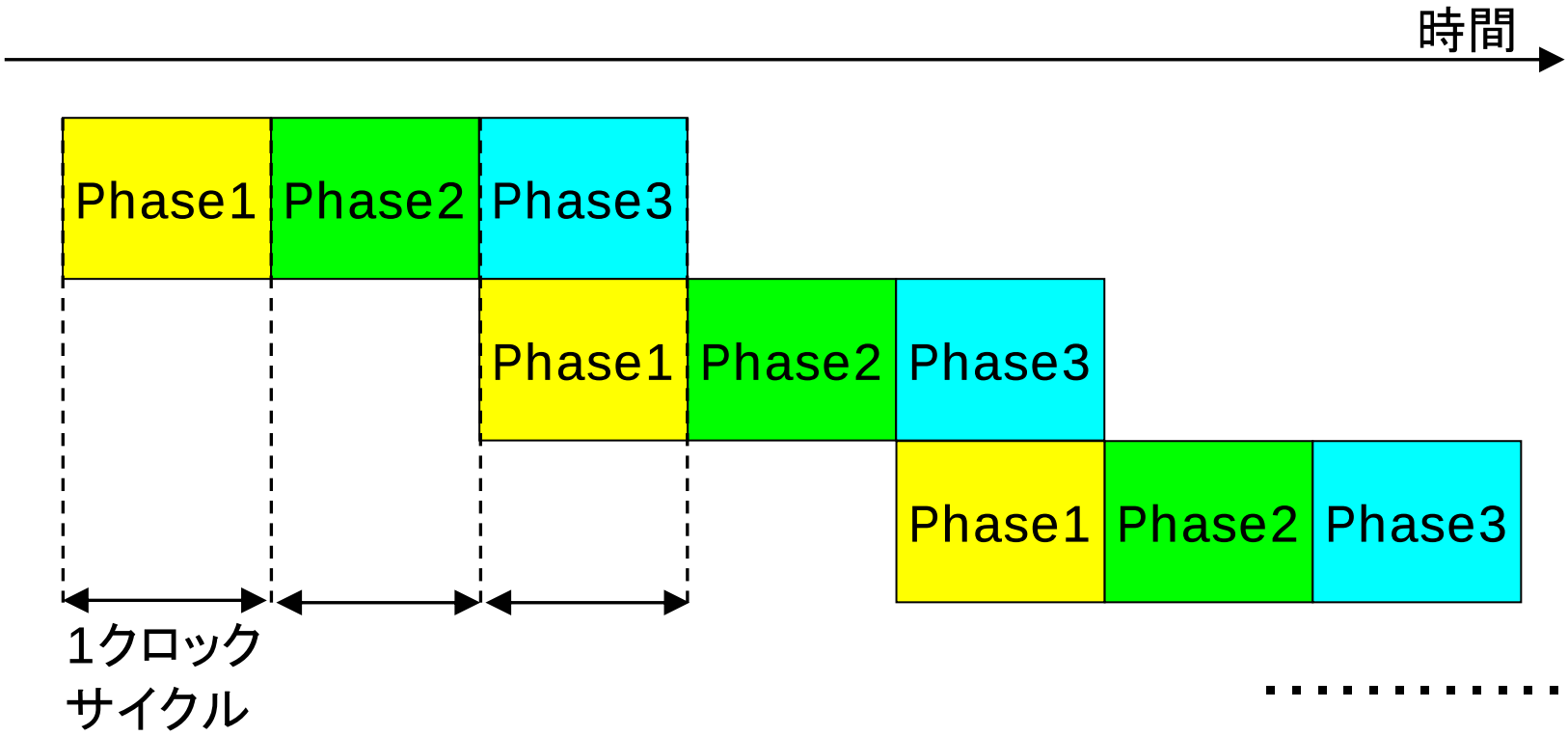
Vulcan実行フロー③
～演算実行～



Vulcan実行フロー③
～演算結果出力～

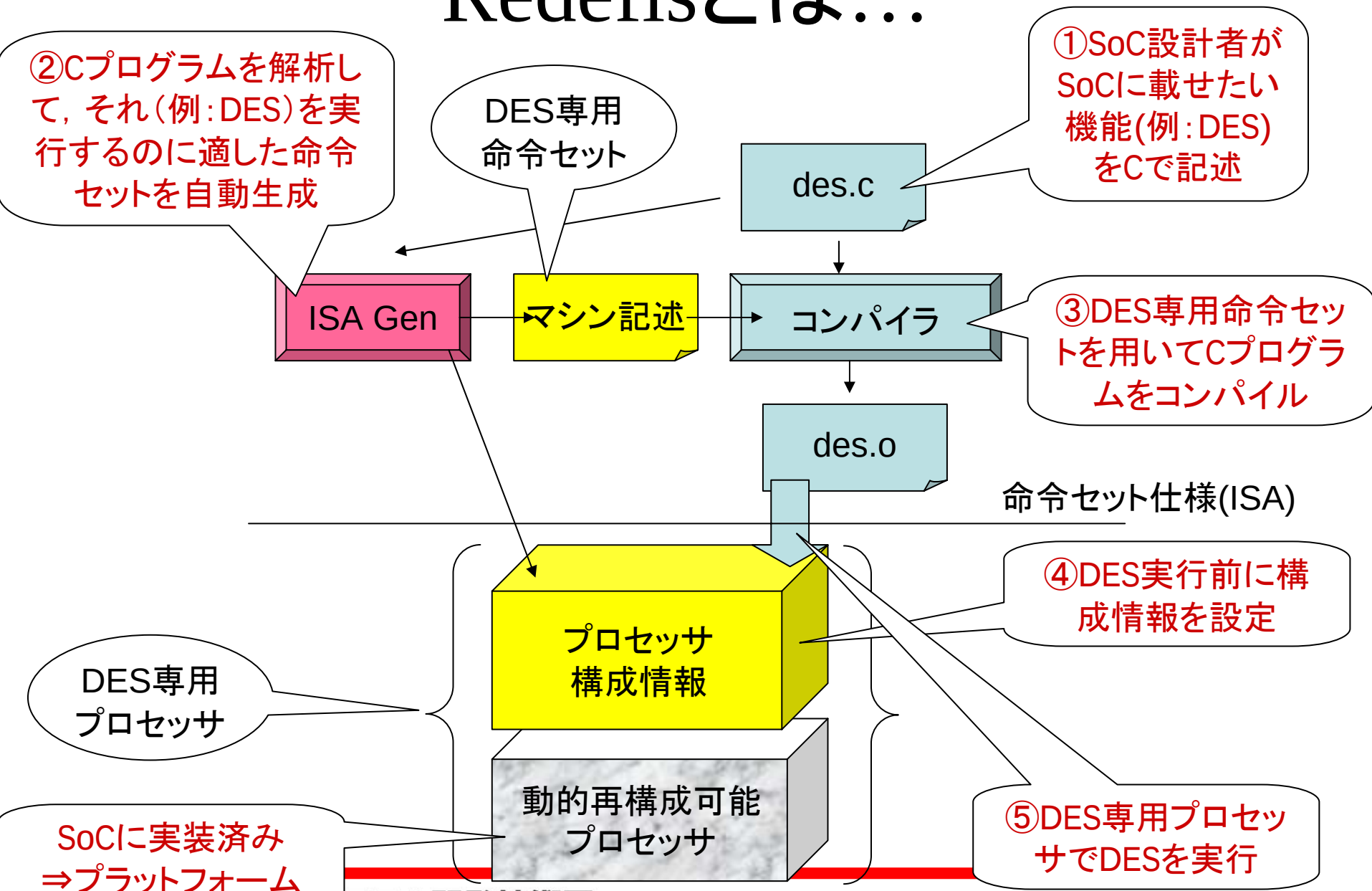


Vulcan1命令のパイプライン実行

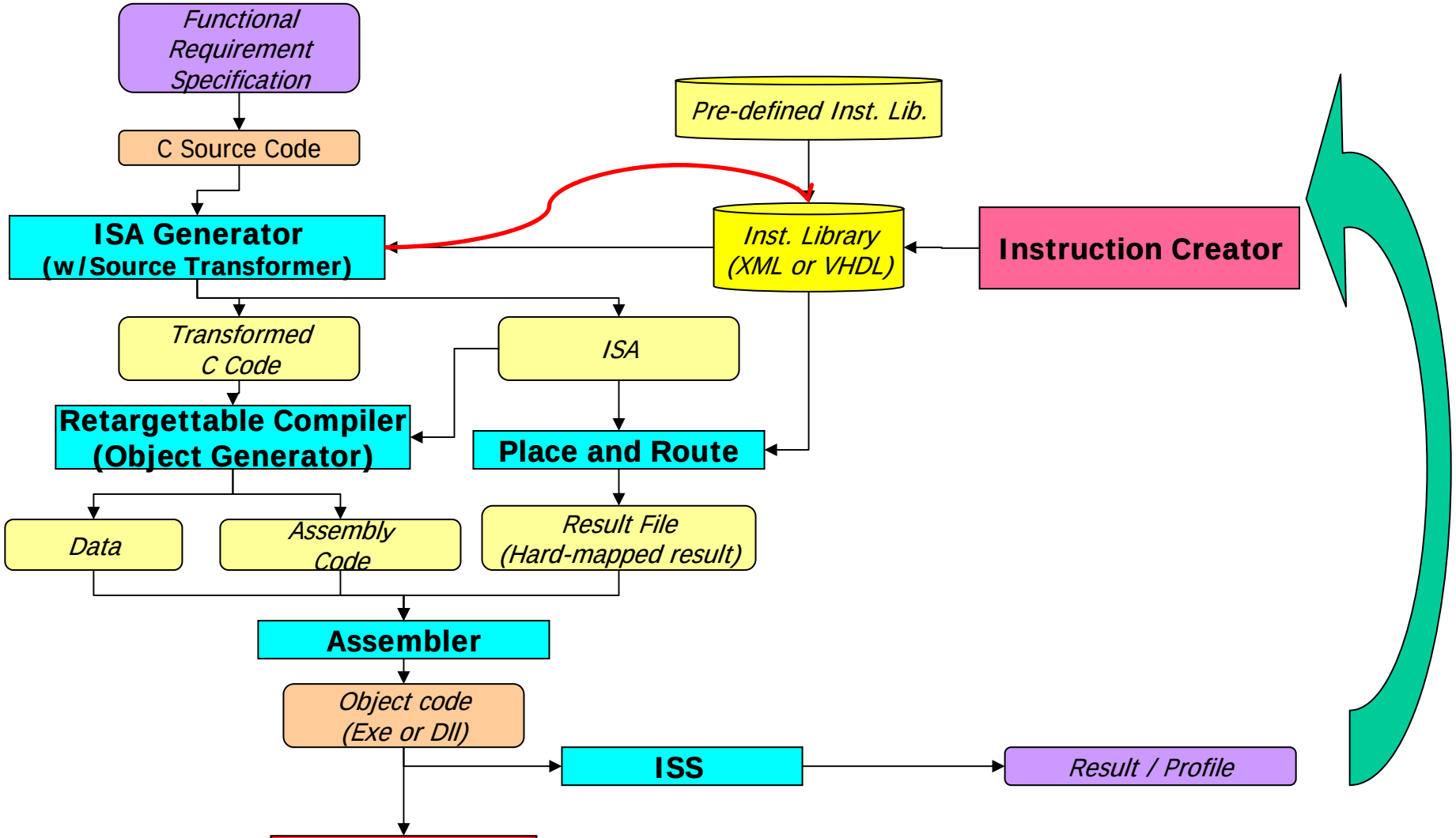


- 1命令の実行に3クロック･サイクル必要
- CPI＝2クロック･サイクル

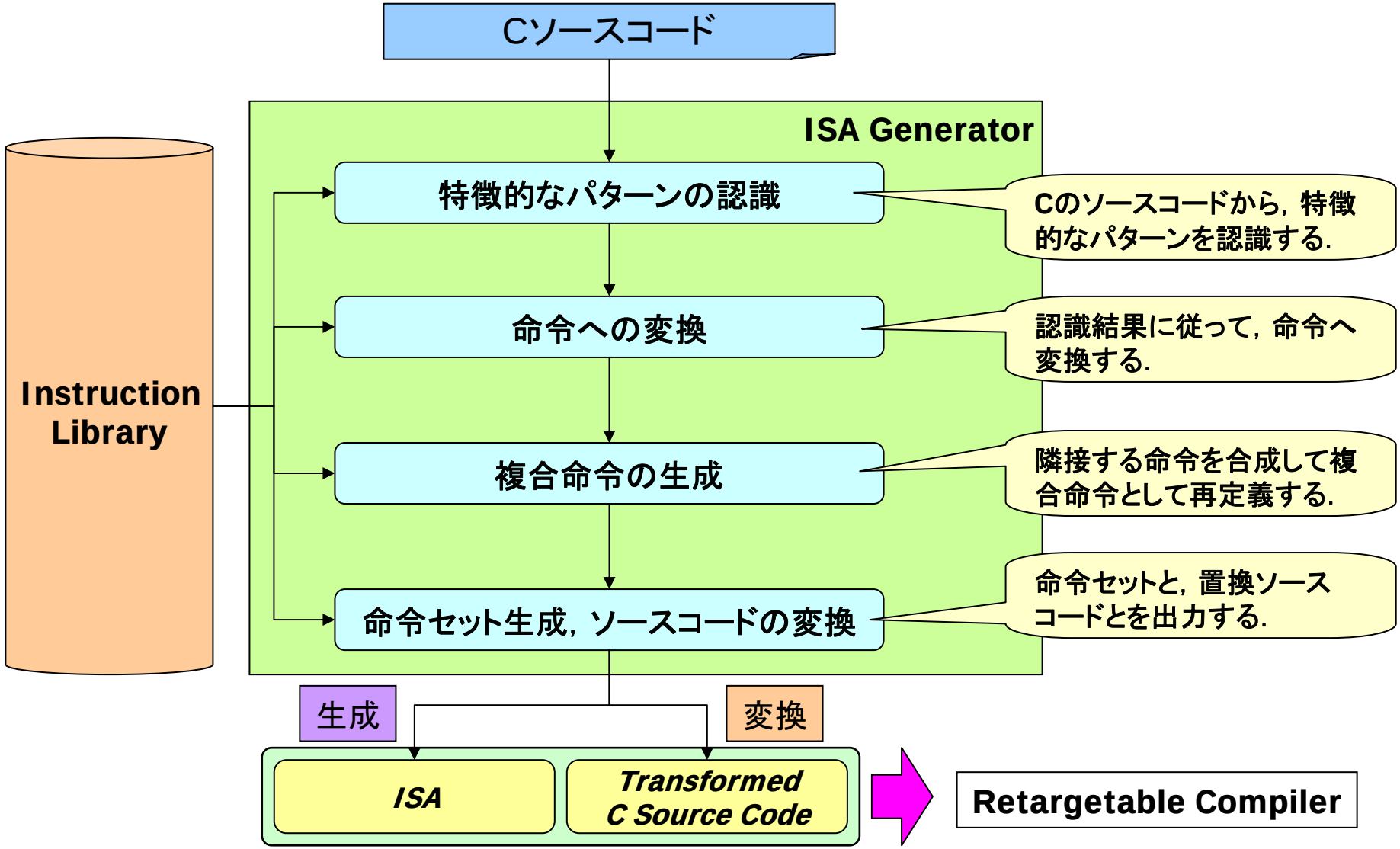
Redefisとは…


②Cプログラムを解析して，それ（例：DES）を実行するのに適した命令セットを自動生成
DES専用
命令セット
①SoC設計者がSoCに載せたい機能(例：DES)をCで記述
des.c
ISA Gen
マシン記述
コンパイラ
③DES専用命令セットを用いてCプログラムをコンパイル
des.o
命令セット仕様(ISA)
④DES実行前に構成情報を設定
プロセッサ
構成情報
DES専用
プロセッサ
動的再構成可能
プロセッサ
SoCに実装済み
⇒プラットフォーム
⑤DES専用プロセッサでDESを実行

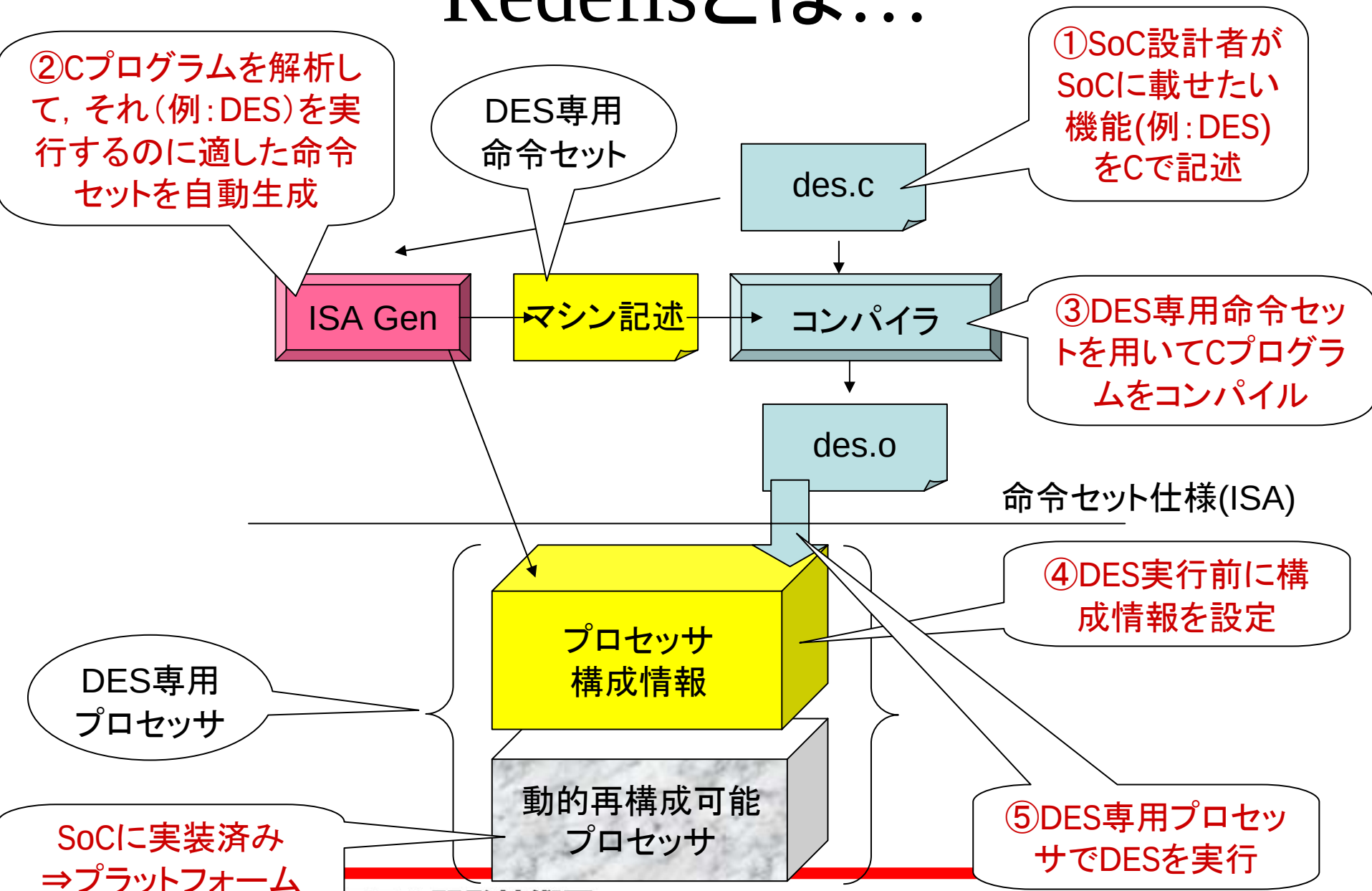
Redefis開発環境
～ツールチェイン～



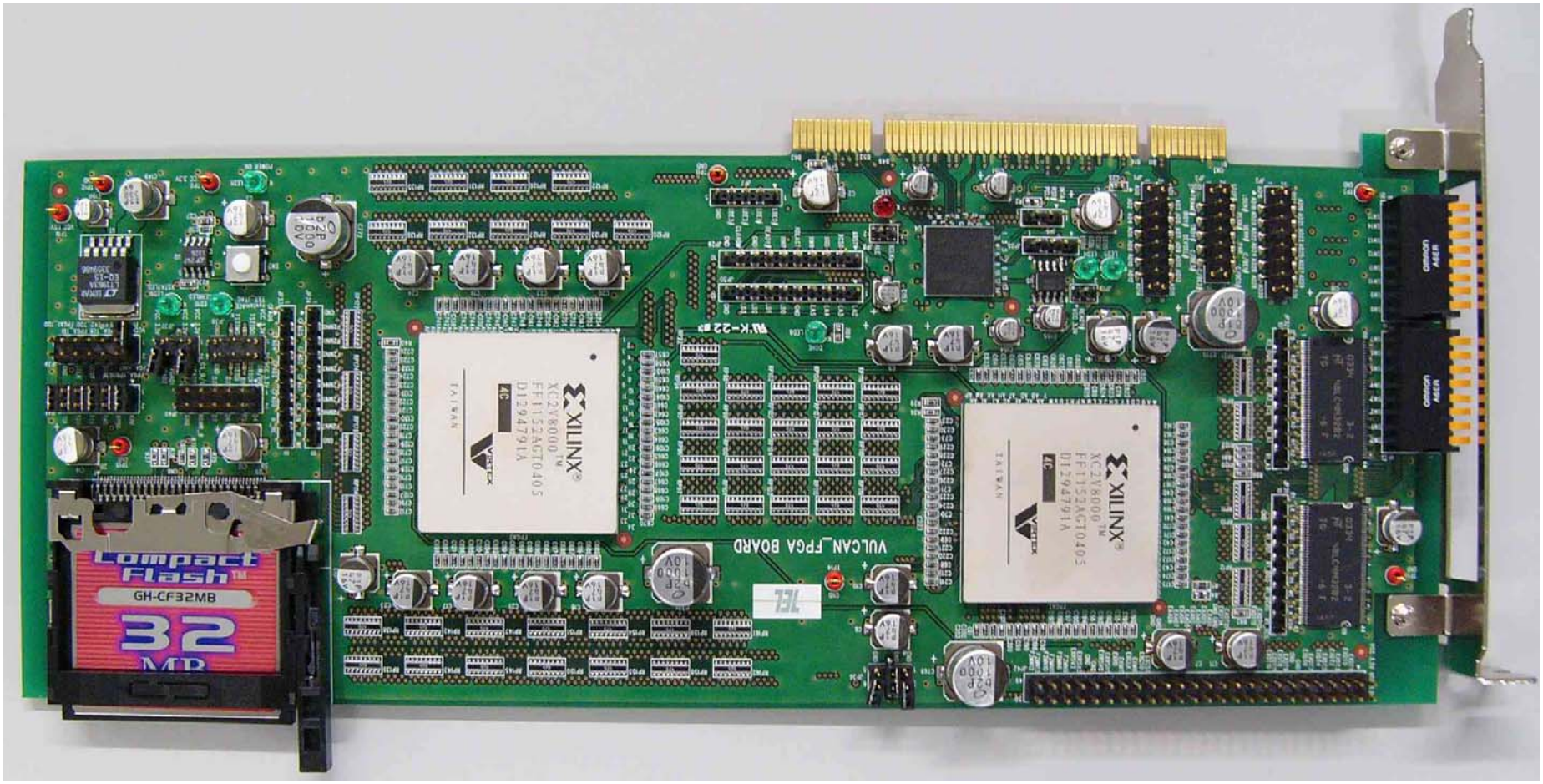
ISA Generator



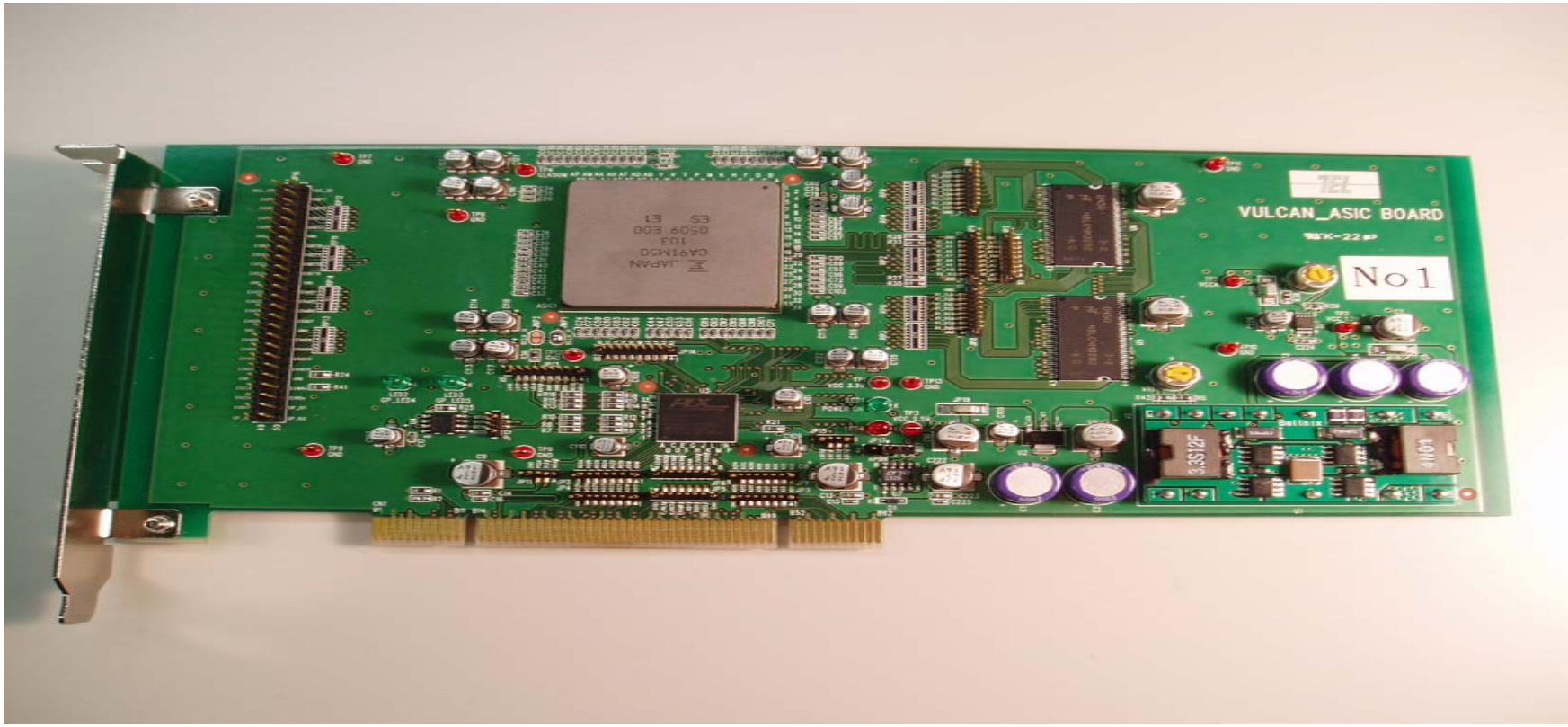
Redefisとは…


②Cプログラムを解析して，それ（例：DES）を実行するのに適した命令セットを自動生成
DES専用
命令セット
①SoC設計者がSoCに載せたい機能(例：DES)をCで記述
des.c
ISA Gen
マシン記述
コンパイラ
③DES専用命令セットを用いてCプログラムをコンパイル
des.o
命令セット仕様(ISA)
④DES実行前に構成情報を設定
プロセッサ
構成情報
DES専用
プロセッサ
動的再構成可能
プロセッサ
SoCに実装済み
⇒プラットフォーム
⑤DES専用プロセッサでDESを実行

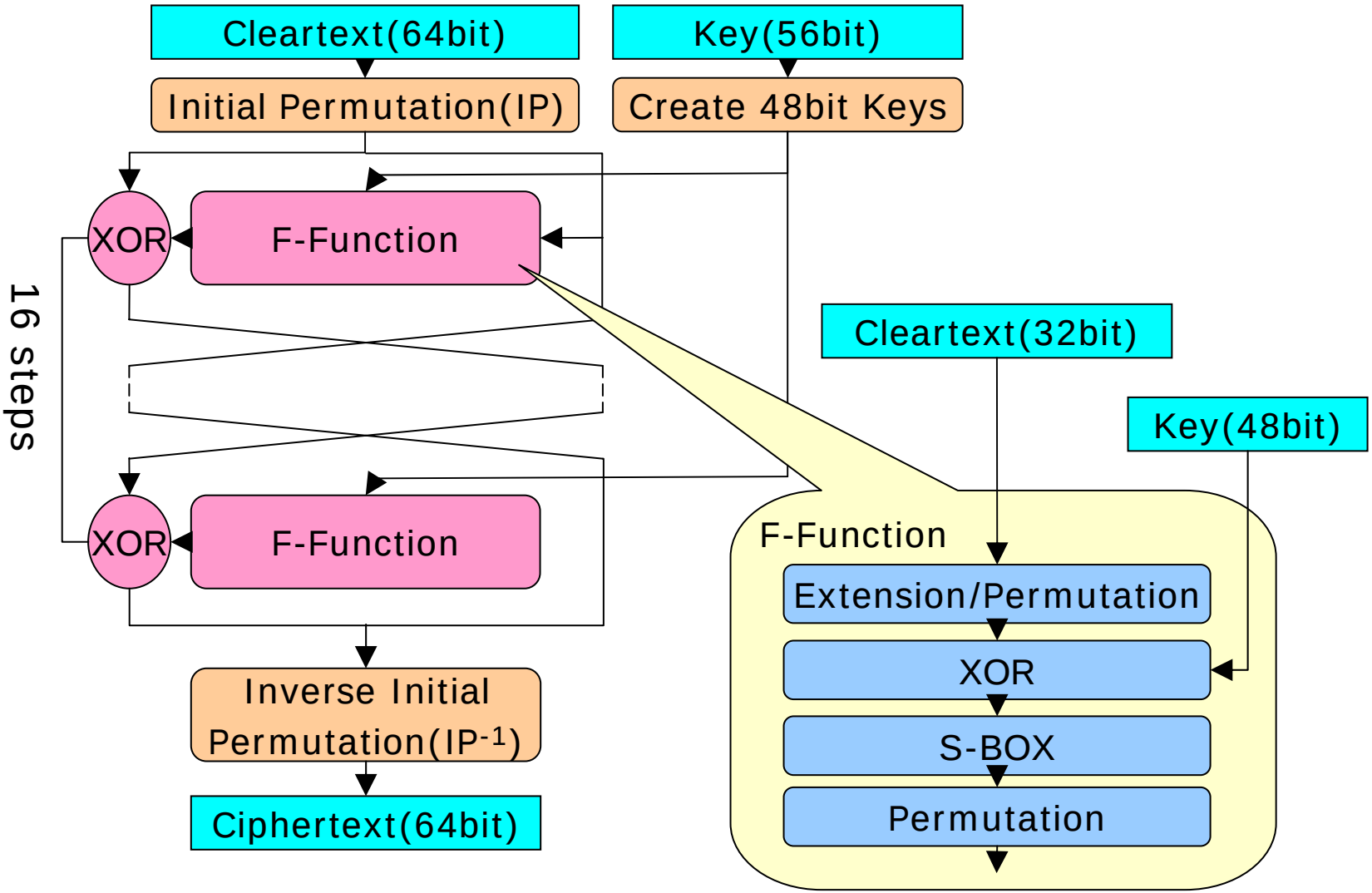
Vulcanを実装したFPGAボード



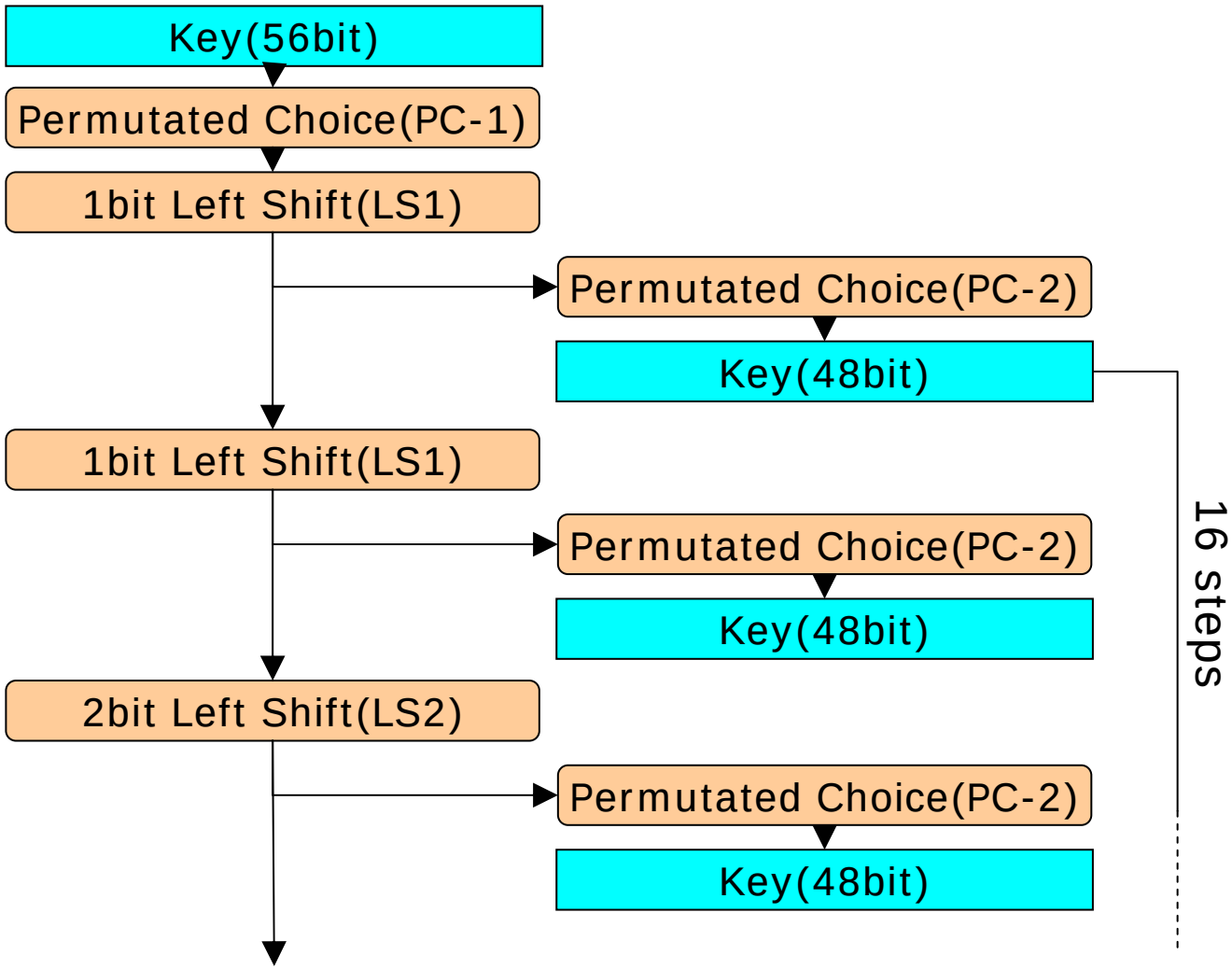
Vulcan LSI搭載ボード



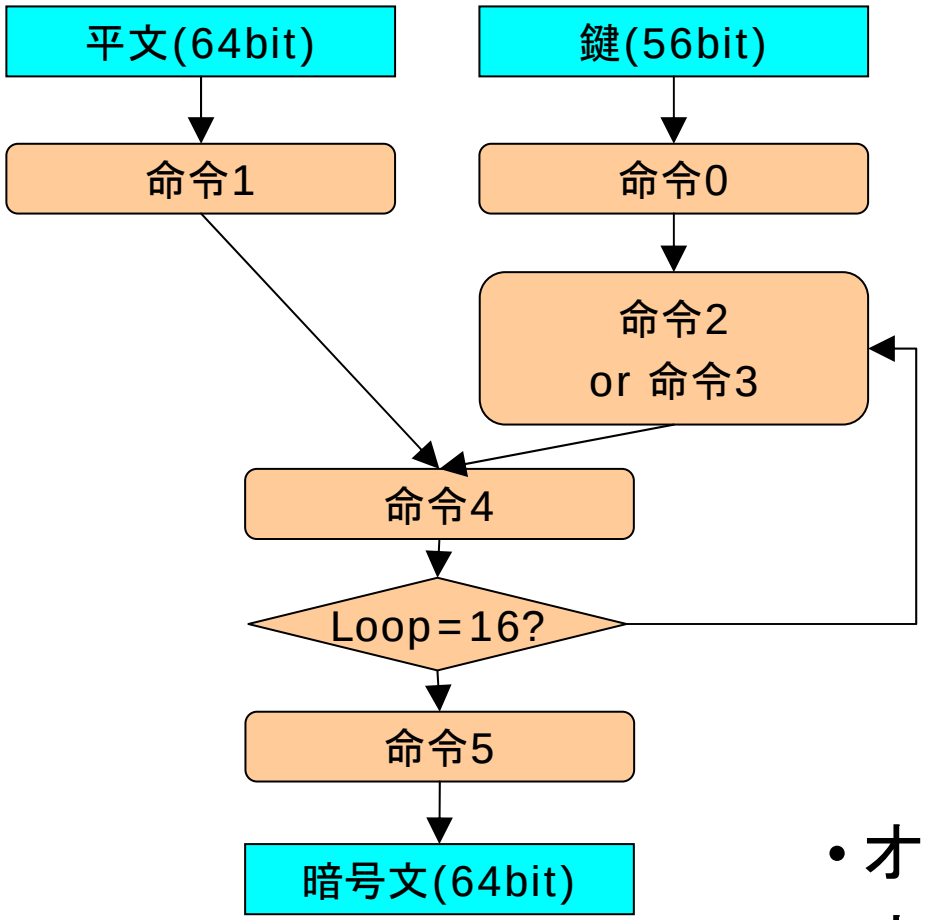
アプリケーション例：DES暗号化



DES暗号化処理
～鍵生成部～



Vulcan上でのDES暗号化処理



命令	内容
0	鍵の読込みと鍵の転置（PC-1）
1	平文の読込みと初期転置（IP）
2	1bit左循環シフト（LS1）
3	2bit左循環シフト（LS2）
4	縮約転置（PC-2）とF関数，および，XOR
5	逆初期転置（IP-1），および，出力

- オブジェクトコード・サイズ：24命令
- 実行命令数：35命令（1回の暗号化）

DES暗号化処理の性能比較

プロセッサ	動作周波数	スループット
Vulcan	6.25MHz	570KB/sec
Pentium4	2.4GHz	150KB/sec

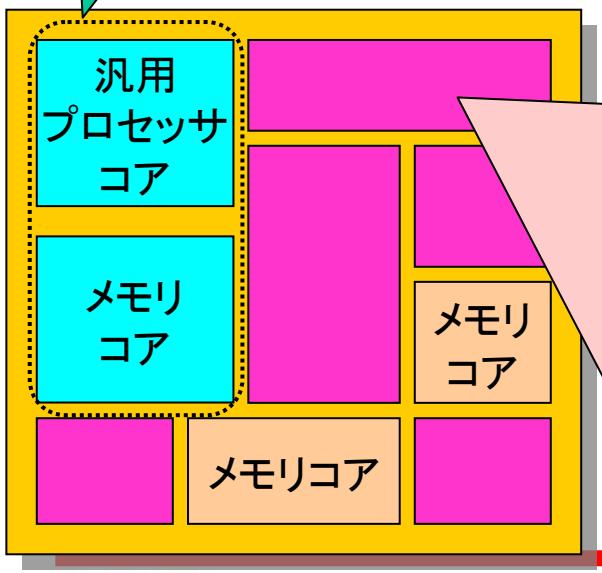
- Pentium4でのDES暗号化処理
  - DES暗号化処理をC言語で忠実に記述したプログラムをコンパイルして実行
  - GCCバージョン：2.95.3
  - 最適化オプション：-O2

おわりに

- SoC設計への諸アプローチ
  - カスタム・ロジックの実現方法
    - プロセッサ＋ソフトウェア
    - コンフィギュラブル・プロセッサ＋ソフトウェア
    - リコンフィギュラブル・プロセッサ＋ソフトウェア
    - リコンフィギュラブル・ハードウェア
- ダイナミック・リコンフィギュラブル・プロセッサの具体例
  - CLUSS Redefis

標準/汎用SoCプラットフォーム

汎用機能の実現法
- 汎用プロセッサ＋ソフトウェア



カスタム・ロジック（専用機能）の実現法
- 「From Scratch」
  - 新規設計のハードウェア
- IPコア・ベース
  - 既存設計のハードウェア
- プラットフォーム・ベース
  - プロセッサ＋ソフトウェア
    - SONY/Toshiba/IBM Cell
    - QuickSilver ACM
  - コンフィギュラブル・プロセッサ＋ソフトウェア
    - Tensilica Xtensa
    - PDI VUPU
  - リコンフィギュラブル・プロセッサ＋ソフトウェア
    - IP Flex DAP/DNA
    - Stretch
    - CLUSS Redefis
  - リコンフィギュラブル・ハードウェア
    - FPGA (Xilinx, Altera, etc.)
    - NEC DRP