

ユースケースにもとづく量産コード生成技術の進化

MathWorks Japan

パイロットエンジニアリング部

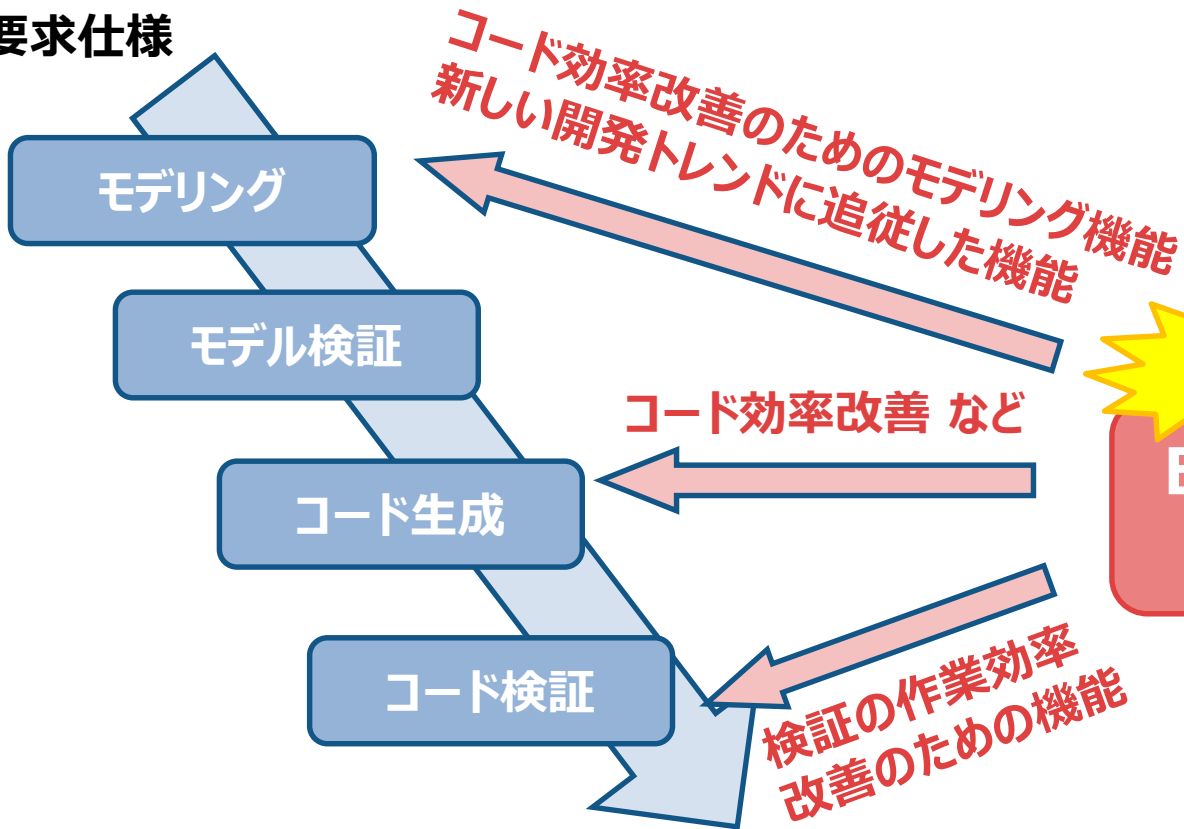
シニアパイロットエンジニア

庄子 晃太郎

本発表のスコープ

Embedded Coder™の進化について紹介します

要求仕様



Embedded Coderの進化によって、日々、開発プロセス上の様々な問題の解決や更なる効率化を実現する機能を提供しています

進化

Embedded
Coder



実装可能なコード

アジェンダ

1. Embedded Coderとは？
2. 自動コード生成ツール適用のメリット
3. Embedded Coderの進化
 - コード生成エンジンの進化
 - 機能の進化
 - 運用フローの進化
4. まとめ

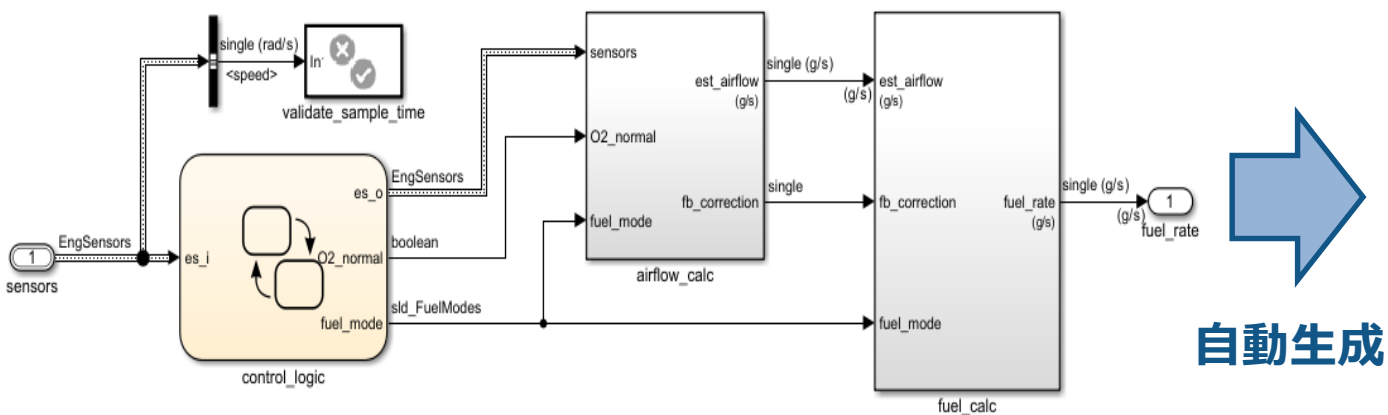
アジェンダ

1. **Embedded Coderとは？**
2. 自動コード生成ツール適用のメリット
3. Embedded Coderの進化
 - コード生成エンジンの進化
 - 機能の進化
 - 運用フローの進化
4. まとめ

Embedded Coderとは？

Simulink®/Stateflow®で作成されたモデルから、組み込み向けコードを自動生成することができる純正ツールです

- **ボタン1発**でモデルから量産コードが生成できます



```

400 void fuel_rate_control_step(void)
401 {
402     real32_T denAccum;
403     real32_T rtb_MultiportSwitch;
404     int32_T sfEvent;
405     real32_T u0;
406
407     /* Outputs for Atomic SubSystem: '<Root>/fuel_rate_control' */
408     /* Chart: '<S1>/control_logic' incorporates:
409      * Inport: '<Root>/sensors'
410      * Lookup_n-D: '<S7>/Pressure_Estimation'
411      * Lookup_n-D: '<S9>/Throttle_Estimation'
412      */
413     /* Block description for '<S1>/control_logic':
414      * Stateflow diagram to determine control system operating mode
415      */
416     sfEvent = CALL_EVENT;
417     if (rtDWork.temporalCounter_i1 < 511U) {
418         rtDWork.temporalCounter_i1++;
419     }
420
421     if (rtDWork.bitsForTID0.is_active_c1_fuel_rate_control == 0U) {
422         rtDWork.bitsForTID0.is_active_c1_fuel_rate_control = 1U;
423         rtDWork.bitsForTID0.is_active_O2 = 1U;
424         rtDWork.bitsForTID0.is_O2 = IN_A;
425         if (rtDWork.bitsForTID0.is_A != IN_O2_warmup) {
426             rtDWork.bitsForTID0.is_A = IN_O2_warmup;
427             rtDWork.temporalCounter_i1 = 0U;
428         }
429
430         if (rtDWork.bitsForTID0.is_active_Pressure != 1U) {
431             rtDWork.bitsForTID0.is_active_Pressure = 1U;
432         }
433
434         rtDWork.bitsForTID0.is_Pressure = IN_normal;
435         if (rtDWork.bitsForTID0.is_active_Throttle != 1U) {
436             rtDWork.bitsForTID0.is_active_Throttle = 1U;
437         }
438
439         rtDWork.bitsForTID0.is_Throttle = IN_normal;
440         if (rtDWork.bitsForTID0.is_active_Speed != 1U) {

```

具体的なEmbedded Coderの量産適用事例



DaimlerChrysler
クルーズコントロール制御



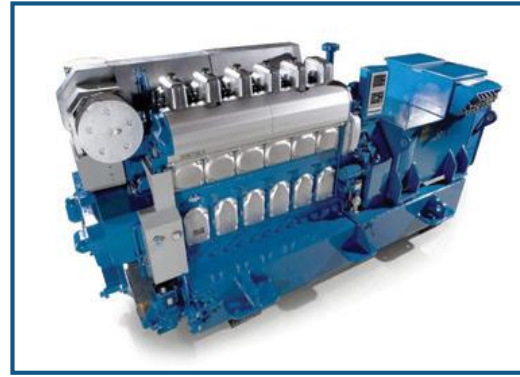
ミツバ
リバーシングワイパー制御



Airbus
空調制御



General Motors
ハイブリッド制御



Wärtsilä
産業用ディーゼルエンジン

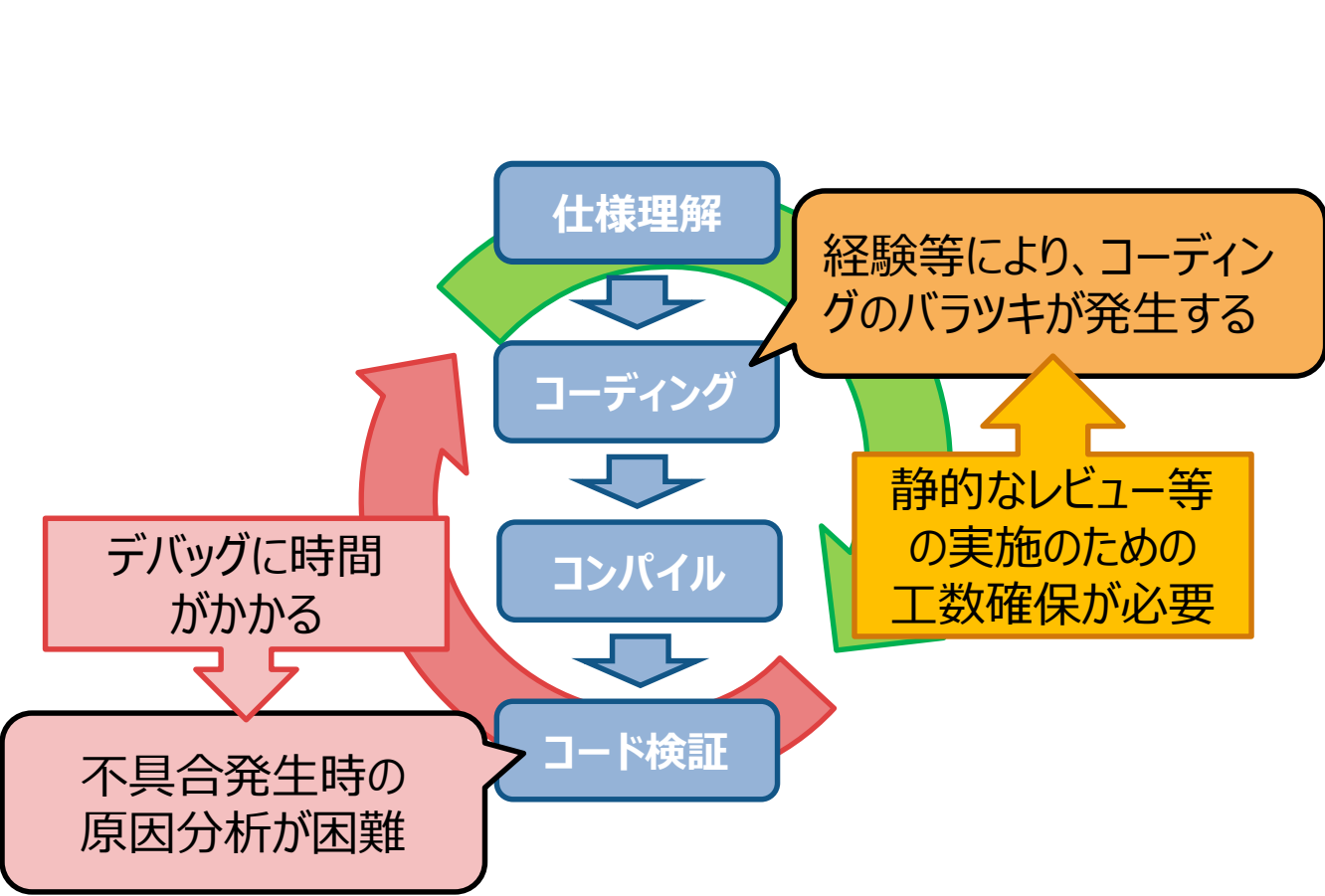


Alstom
鉄道用電力変換システム

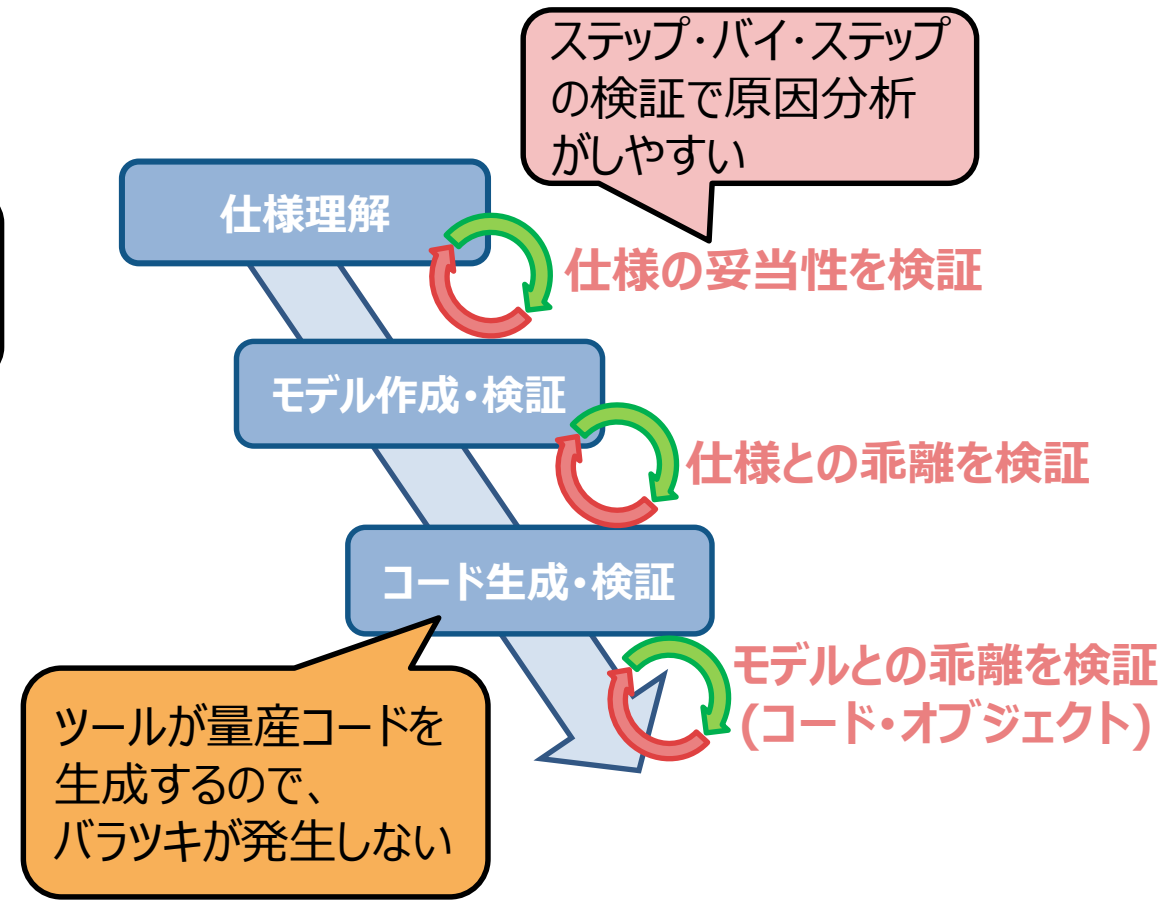
アジェンダ

1. Embedded Coderとは？
2. **自動コード生成ツール適用のメリット**
3. Embedded Coderの進化
 - コード生成エンジンの進化
 - 機能の進化
 - 運用フローの進化
4. まとめ

自動コード生成ツールの適用による工数短縮と高品質の両立



ハンドコードによる従来開発



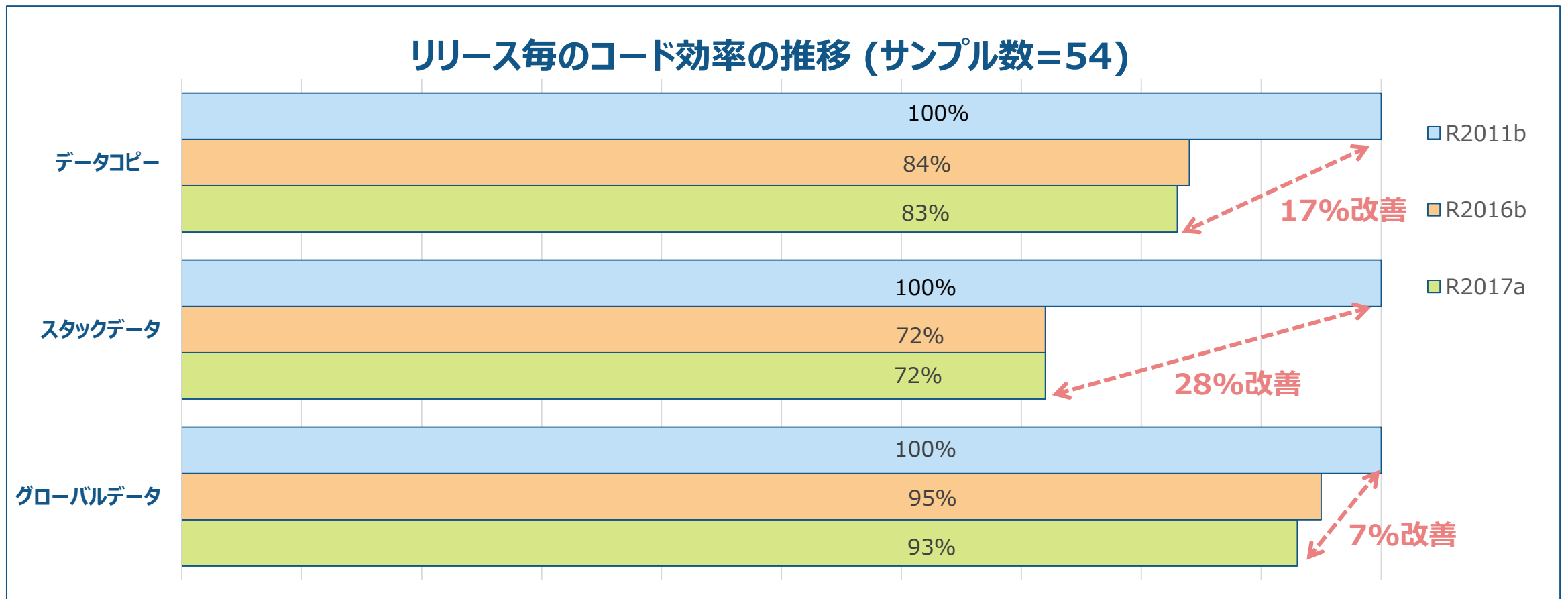
自動コード生成ツールによる開発

アジェンダ

1. Embedded Coderとは？
2. 自動コード生成ツール適用のメリット
3. **Embedded Coderの進化**
 - **コード生成エンジンの進化**
 - 機能の進化
 - 運用フローの進化
4. まとめ

Embedded Coder生成コードの進化

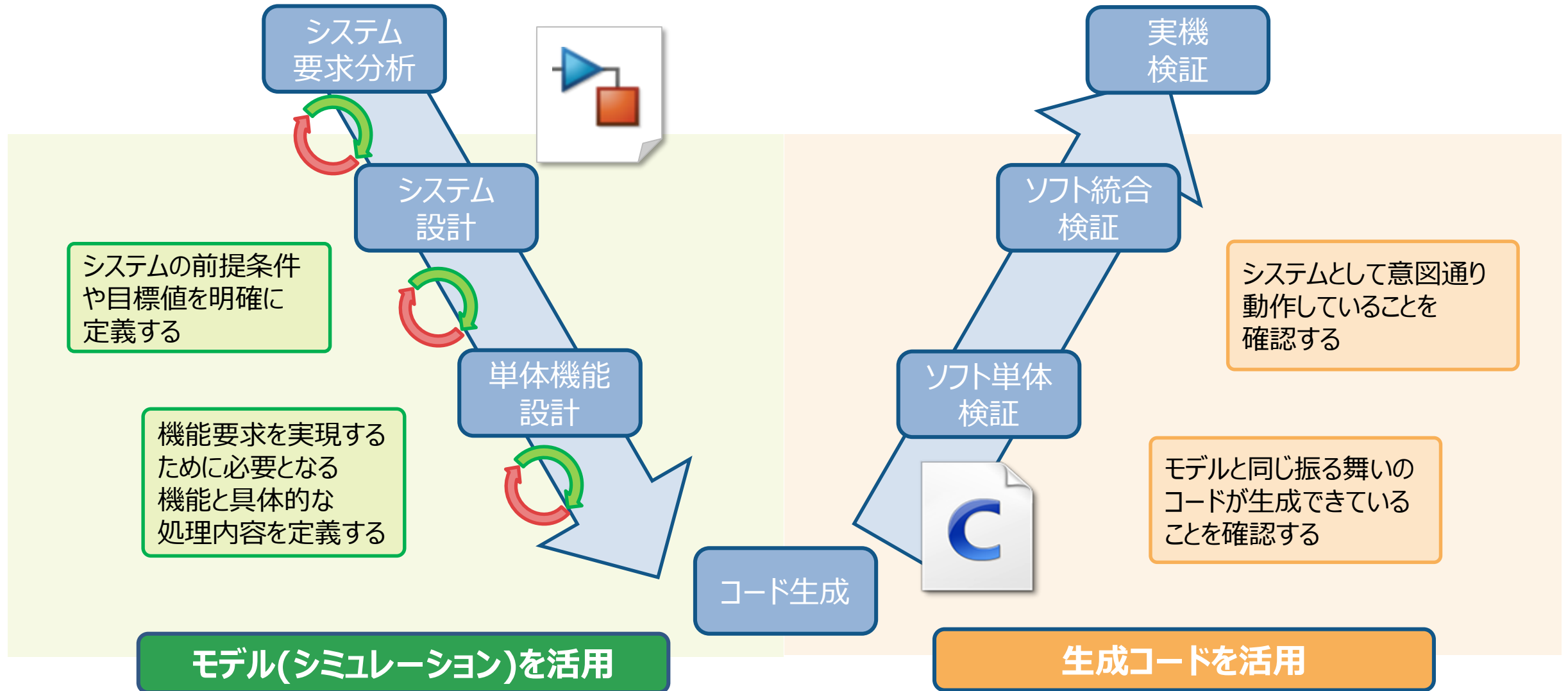
日々改善活動をおこない、バージョンアップの度に更なるコード効率向上を遂げています



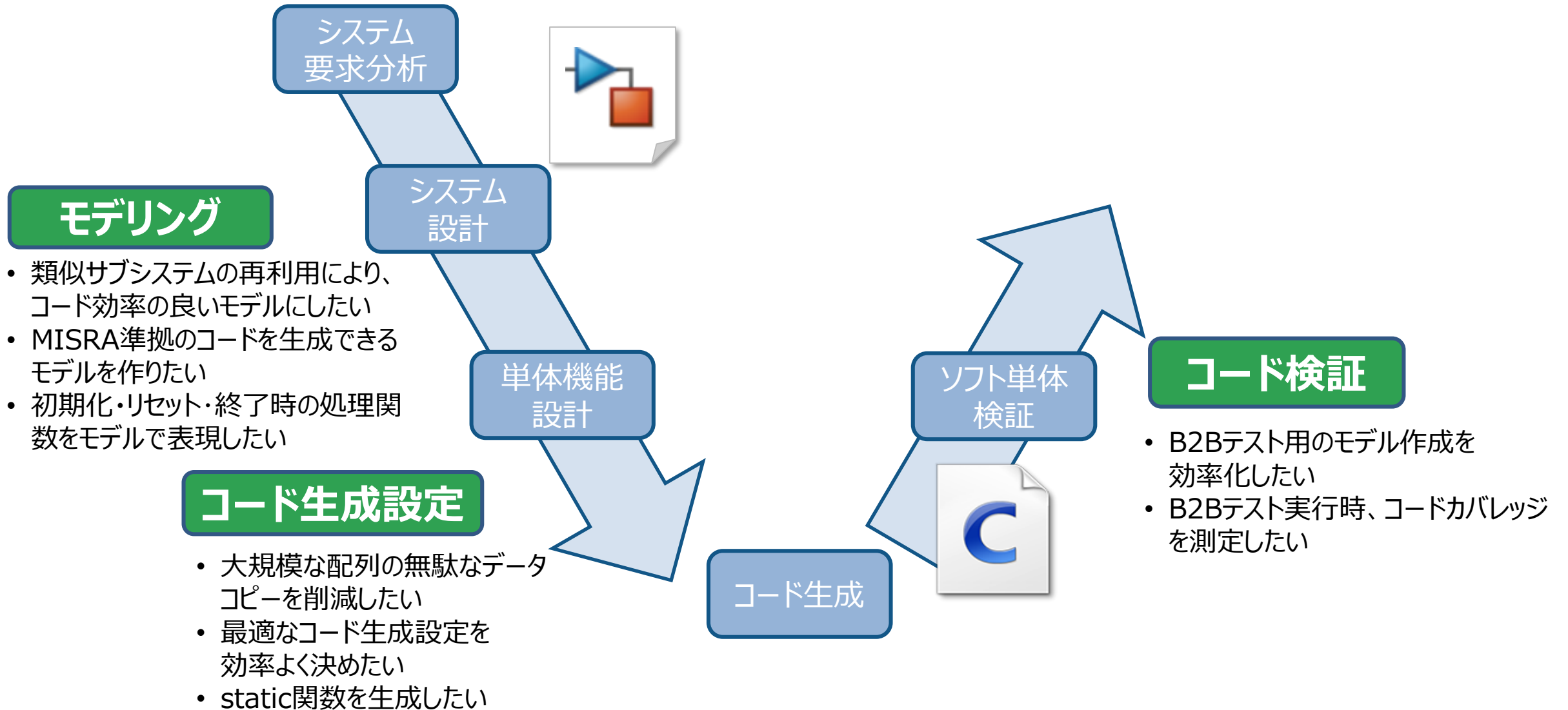
アジェンダ

1. Embedded Coderとは？
2. 自動コード生成ツール適用のメリット
3. **Embedded Coderの進化**
 - コード生成エンジンの進化
 - **機能の進化**
 - 運用フローの進化
4. まとめ

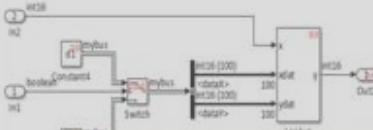




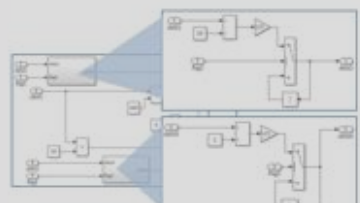
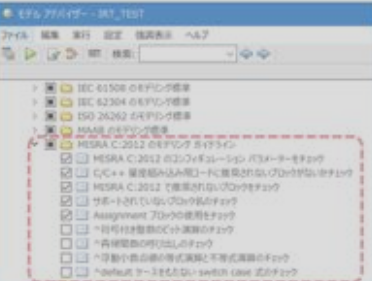
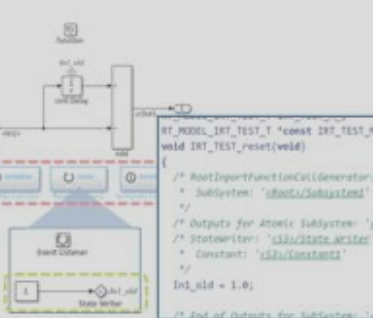
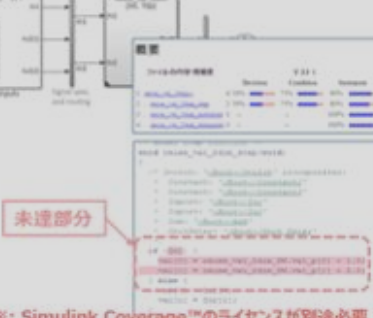
量産 / 組み込み実装向け開発プロセスの流れ



量産コード開発プロセスのよくあるユースケース



ユースケースに対するEmbedded Coderソリューション

<p>大規模な配列の無駄なデータコピーを削減したい</p> <p>最適化が強化され、無駄なmemcpyが減りました</p>  <pre> void rtwDemo_pointer_conversion_1fa_step(void) { const int32_T *rtb_Switch_dataX; const int32_T *rtb_Switch_dataY; if (rtb_data) { rtb_Switch_dataX = (int32_T *)rtb_data[0]; rtb_Switch_dataY = (int32_T *)rtb_data[1]; } else { rtb_Switch_dataX = (int32_T *)rtb_data[0]; } } </pre> <p>Memcpyではなく、ポインタを利用するコードが生成される</p>	<p>最適なコード生成設定を効率よく決めたい</p> <p>Embedded Coder クイックスタートを利用することで、ウィザード形式で最適な設定を決定可能</p> 	<p>static関数を生成したい</p> <p>ファイルパッケージをモジュラー以外かつ“関数宣言に静的キーワードを保持”をONにすることで生成可能</p>  <pre> /* Forward declaration for local function */ static real_T test_initialstatic_onl10real_T_data; /* Add: stop function */ void test_initialstatic_step(void) </pre>	<p>B2Bテスト用のモデル作成を効率化したい</p> <p>テストハーネスの作成機能の利用により、B2B用モデルの作成を効率化できる</p>  						
<p>R2011b~17bまでの間で、約450件以上の組み込みコード生成に関連した機能追加・改善が実施されています</p>									
<p>R2016b</p> <p>類似サブシステムの再コード効率の良いモデル</p> <p>モデリング クローンの特徴</p> <p>再利用可能なサブシステムを生成可能</p>  <table border="1" data-bbox="471 1220 764 1299"> <thead> <tr> <th>Clone Group 1</th> <th>Remarks</th> </tr> </thead> <tbody> <tr> <td><input type="checkbox"/> clone_sample2.Subsystem</td> <td>Virtual subsystem</td> </tr> <tr> <td><input checked="" type="checkbox"/> clone_sample2.Subsystem1</td> <td>Virtual subsystem</td> </tr> </tbody> </table> <p>※: Simulink Check™のライセンスが別途必要</p>	Clone Group 1	Remarks	<input type="checkbox"/> clone_sample2.Subsystem	Virtual subsystem	<input checked="" type="checkbox"/> clone_sample2.Subsystem1	Virtual subsystem	<p>R2015b</p> <p>Guidelines for MISRA C:2012</p> <p>でモデルの準拠状況をチェック可能</p> 	<p>R2016b</p> <p>実現可能</p>  <pre> /* RootExportFunctionCallGenerator: * Subsystem: 'statefulSubsys1' */ /* Outputs for Atomic Subsystem: 'statefulSubsys1' * Statewriter: 'statefulSubsys1' * Constant: 'statefulSubsys1' */ int32_T statefulSubsys1; int32_T statefulSubsys1; } </pre>	<p>R2016a</p> <p>実行時、コードカバレッジを測定したい</p> <p>シミュレーション中に生成コードの測定できる</p>  <p>※: Simulink Coverage™のライセンスが別途必要</p>
Clone Group 1	Remarks								
<input type="checkbox"/> clone_sample2.Subsystem	Virtual subsystem								
<input checked="" type="checkbox"/> clone_sample2.Subsystem1	Virtual subsystem								

ユースケースに対するEmbedded Coderソリューション

<p>大規模な配列の無駄なデータコピーを削減したい</p> <p>最適化が強化され、無駄なmemcpyが減りました</p> <p>Memcpyではなく、ポインタを利用するコードが生成される</p>	<p>最適なコード生成設定を効率よく決めたい</p> <p>Embedded Coder クイックスタートを利用することで、ウィザード形式で最適な設定を決定可能</p>	<p>static関数を生成したい</p> <p>ファイルパッケージをモジュラー以外かつ“関数宣言に静的キーワードを保持”をONにすることで生成可能</p>	<p>B2Bテスト用のモデル作成を効率化したい</p> <p>テストハーネスの作成機能の利用により、B2B用モデルの作成を効率化できる</p> <p>※: Simulink Test™のライセンスが別途必要</p>						
<p>R2017b</p>	<p>R2015b</p>	<p>R2017b</p>	<p>R2015a</p>						
<p>類似サブシステムの再利用により、コード効率の良いモデルにしたい</p> <p>モデリング クローンの特定機能により、再利用可能なサブシステムを検出可能</p> <table border="1"> <thead> <tr> <th>Clone Group 1</th> <th>Remarks</th> </tr> </thead> <tbody> <tr> <td>clone_sample2.Subsystem</td> <td>Virtual subsystem</td> </tr> <tr> <td>clone_sample3.Subsystem</td> <td>Virtual subsystem</td> </tr> </tbody> </table> <p>※: Simulink Check™のライセンスが別途必要</p>	Clone Group 1	Remarks	clone_sample2.Subsystem	Virtual subsystem	clone_sample3.Subsystem	Virtual subsystem	<p>MISRA C 2012準拠のコードを生成できるモデルを作りたい</p> <p>モデルアドバイザーのModeling Guidelines for MISRA C:2012でモデルの準拠状況をチェック可能</p>	<p>初期化・リセット・終了時の処理関数をモデルで表現したい</p> <p>Initialize/Reset/Terminate Function内に処理を記述することで実現可能</p>	<p>B2Bテスト実行時、コードカバレッジを測定したい</p> <p>SILシミュレーション中に生成コードのカバレッジを測定できる</p> <p>未達部分</p> <p>※: Simulink Coverage™のライセンスが別途必要</p>
Clone Group 1	Remarks								
clone_sample2.Subsystem	Virtual subsystem								
clone_sample3.Subsystem	Virtual subsystem								
<p>R2016b</p>	<p>R2015b</p>	<p>R2016b</p>	<p>R2016a</p>						

大規模な配列の無駄なデータコピーを削減したい

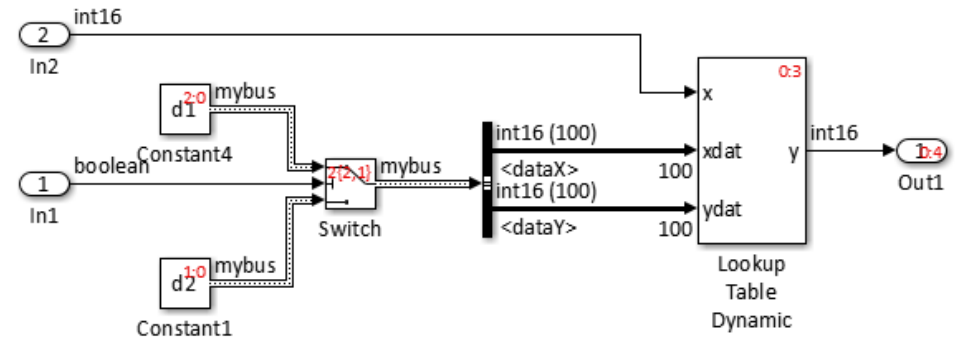
R2017b

最適化が強化され、無駄なmemcpyが減りました

Before

```
void rtwdemo_pointer_conversion_15b_step(void)
{
    int16_T rtb_Switch_dataX[100];
    int16_T rtb_Switch_dataY[100];

    /* Switch: '<Root>/Switch' incorporates:
     * Inport: '<Root>/In1'
     */
    if (rtU.In1) {
        /* Switch: '<Root>/Switch' incorporates:
         * Constant: '<Root>/Constant4'
         */
        memcpy(&rtb_Switch_dataX[0], (&(d1.dataX[0])), 100U * sizeof(int16_T));
        memcpy(&rtb_Switch_dataY[0], (&(d1.dataY[0])), 100U * sizeof(int16_T));
    } else {
        /* Switch: '<Root>/Switch' incorporates:
         * Constant: '<Root>/Constant1'
         */
        memcpy(&rtb_Switch_dataX[0], (&(d2.dataX[0])), 100U * sizeof(int16_T));
        memcpy(&rtb_Switch_dataY[0], (&(d2.dataY[0])), 100U * sizeof(int16_T));
    }
}
```



After

```
void rtwdemo_pointer_conversion_17a_step(void)
{
    const int16_T *rtb_Switch_dataX;
    const int16_T *rtb_Switch_dataY;
    if (rtU.In1) {
        rtb_Switch_dataX = (&(d1.dataX[0]));
        rtb_Switch_dataY = (&(d1.dataY[0]));
    } else {
        rtb_Switch_dataX = (&(d2.dataX[0]));
        rtb_Switch_dataY = (&(d2.dataY[0]));
    }
}

LookIn_S16_S16( &(rtY.Out1), &rtb_Switch_dataY[0],
```

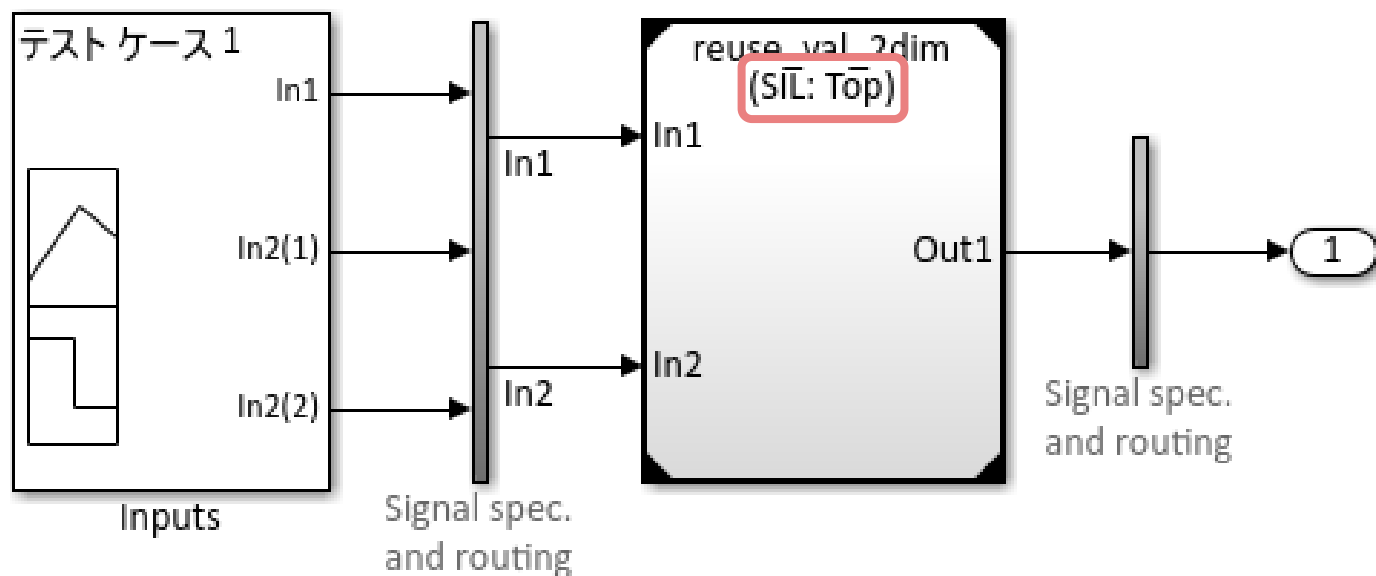

ユースケースに対するEmbedded Coderソリューション

<p>大規模な配列の無駄なデータコピーを削減したい</p> <p>最適化が強化され、無駄なmemcpyが減りました</p> <p>Memcpyではなく、ポインタを利用するコードが生成される</p>	<p>最適なコード生成設定を効率よく決めたい</p> <p>Embedded Coder クイックスタートを利用することで、ウィザード形式で最適な設定を決定可能</p>	<p>static関数を生成したい</p> <p>ファイルパッケージをモジュラー以外かつ“関数宣言に静的キーワードを保持”をONにすることで生成可能</p>	<p>B2Bテスト用のモデル作成を効率化したい</p> <p>テストハーネスの作成機能の利用により、B2B用モデルの作成を効率化できる</p> <p>※: Simulink Test™のライセンスが別途必要</p>
<p>R2017b</p>	<p>R2015b</p>	<p>R2017b</p>	<p>R2015a</p>
<p>類似サブシステムの再利用により、コード効率の良いモデルにしたい</p> <p>モデリング クローンの特定機能により、再利用可能なサブシステムを検出可能</p> <p>※: Simulink Check™のライセンスが別途必要</p>	<p>MISRA C 2012準拠のコードを生成できるモデルを作りたい</p> <p>モデルアドバイザーのModeling Guidelines for MISRA C:2012でモデルの準拠状況をチェック可能</p>	<p>初期化・リセット・終了時の処理関数をモデルで表現したい</p> <p>Initialize/Reset/Terminate Function内に処理を記述することで実現可能</p>	<p>B2Bテスト実行時、コードカバレッジを測定したい</p> <p>SILシミュレーション中に生成コードのカバレッジを測定できる</p> <p>未達部分</p> <p>※: Simulink Coverage™のライセンスが別途必要</p>
<p>R2016b</p>	<p>R2015b</p>	<p>R2016b</p>	<p>R2016a</p>

B2Bテスト実行時、コードカバレッジを測定したい

R2016a

自動生成コードのカバレッジが計測できるようになりました



概要

ファイルの内容/複雑度	テスト1		
	Decision	Condition	Statement
1. reuse_val_2dim.c	4 50%	75%	90%
2... reuse_val_2dim_step	2 50%	75%	80%
3... reuse_val_2dim_initialize	1 --	--	100%
4... reuse_val_2dim_terminate	1 --	--	100%

```

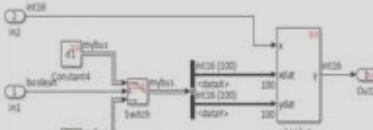




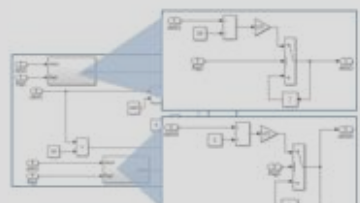
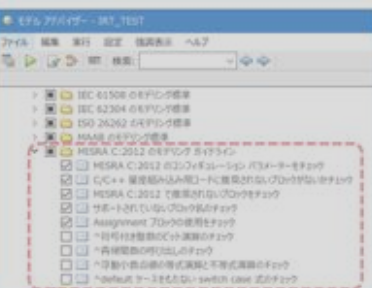
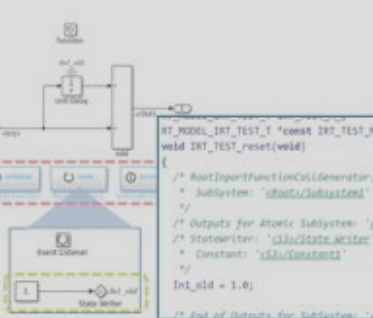

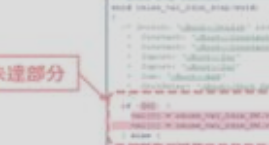
/* Model step function */
void reuse_val_2dim_step(void)
{
    /* Switch: '<Root>/Switch' incorporates:
     * Constant: '<Root>/Constant2'
     * Constant: '<Root>/Constant3'
     * Inport: '<Root>/In1'
     * Inport: '<Root>/In2'
     * Sum: '<Root>/Add'
     * UnitDelay: '<Root>/Unit Delay'
     */
    if (In1) {
        val[0] = reuse_val_2dim_DW.val_p[0] + 1.0;
        val[1] = reuse_val_2dim_DW.val_p[1] + 2.0;
    } else {
        val[0] = In2[0];
        val[1] = In2[1];
    }
}
/* End of Switch: '<Root>/Switch' */

```

未達部分

※: Simulink Coverage™のライセンスが別途必要

ユースケースに対するEmbedded Coderソリューション

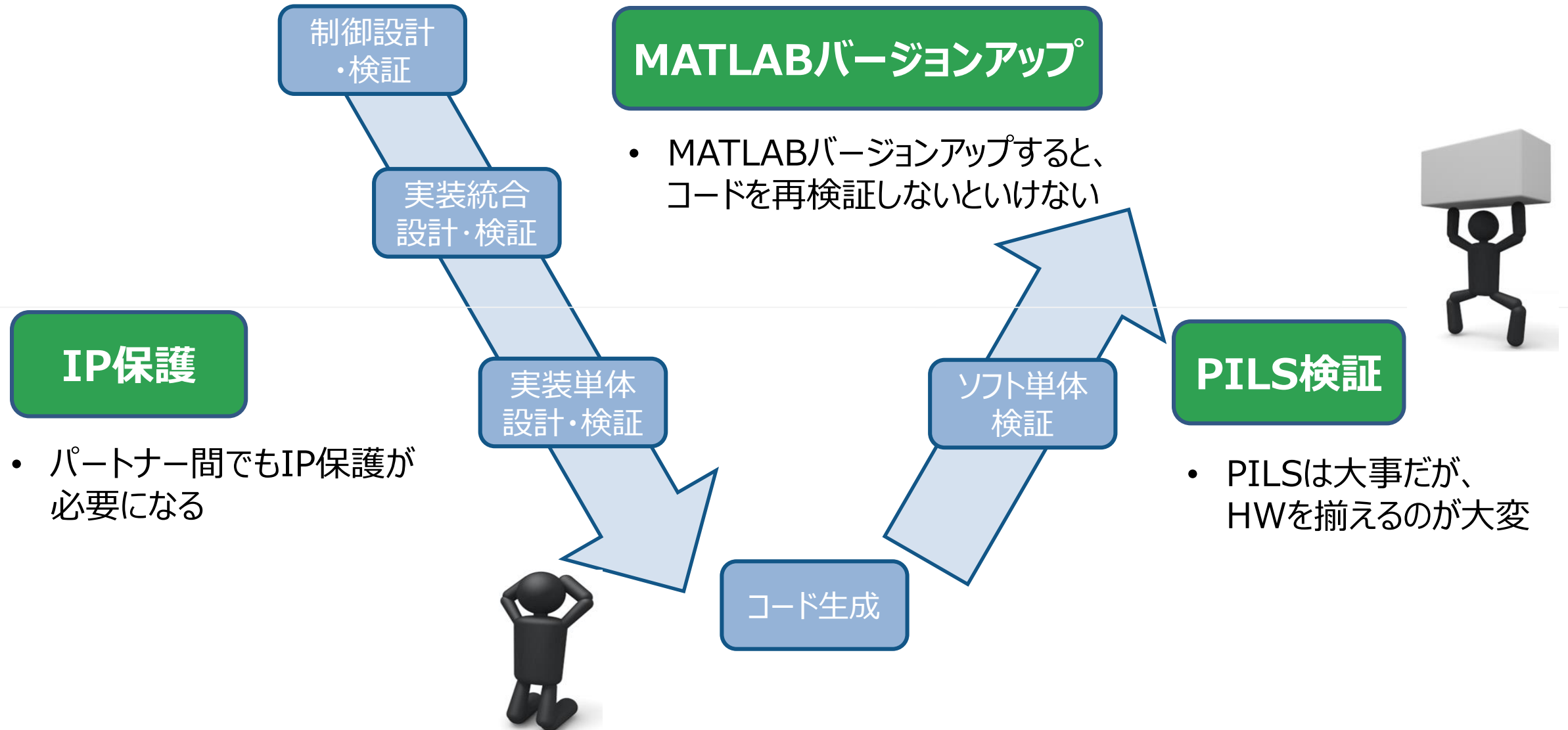
<p>大規模な配列の無駄なデータコピーを削減したい</p> <p>最適化が強化され、無駄なmemcpyが減りました</p>  <pre> void rtwDemo_pointer_conversion_1fa_step(void) { const int32_T *rtb_Switch_dataX; const int32_T *rtb_Switch_dataY; if (rtb_data) { rtb_Switch_dataX = (int32_T *)dataX[0]; rtb_Switch_dataY = (int32_T *)dataY[0]; } else { rtb_Switch_dataX = (int32_T *)dataX[1]; rtb_Switch_dataY = (int32_T *)dataY[1]; } } </pre> <p>Memcpyではなく、ポインタを利用するコードが生成される</p>	<p>最適なコード生成設定を効率よく決めたい</p> <p>Embedded Coder クイックスタートを利用することで、ウィザード形式で最適な設定を決定可能</p> 	<p>static関数を生成したい</p> <p>ファイルパッケージをモジュラー以外かつ“関数宣言に静的キーワードを保持”をONにすることで生成可能</p>  <pre> /* Forward declaration for local function */ static real_T test_initialstatic_onl10real_1_data; /* Add: stop function */ void test_initialstatic_step(void) </pre>	<p>B2Bテスト用のモデル作成を効率化したい</p> <p>テストハーネスの作成機能の利用により、B2B用モデルの作成を効率化できる</p>  						
<p>R2011b</p> <p>類似サブシステムの再コード効率の良いモデル</p> <p>モデリング クローンの特徴</p> <p>再利用可能なサブシステムを生成可能</p>  <table border="1"> <thead> <tr> <th>Clone Group 1 (2 clones, 5 blocks/clone)</th> <th>Remarks</th> </tr> </thead> <tbody> <tr> <td><input checked="" type="checkbox"/> clone_sample2Subsystem</td> <td>Virtual subsystem</td> </tr> <tr> <td><input checked="" type="checkbox"/> clone_sample3Subsystem</td> <td>Virtual subsystem</td> </tr> </tbody> </table> <p>※: Simulink Check™のライセンスが別途必要</p>	Clone Group 1 (2 clones, 5 blocks/clone)	Remarks	<input checked="" type="checkbox"/> clone_sample2Subsystem	Virtual subsystem	<input checked="" type="checkbox"/> clone_sample3Subsystem	Virtual subsystem	<p>R2015b</p> <p>Guidelines for MISRA C:2012でモデルの準拠状況をチェック可能</p> 	<p>R2016b</p> <p>実現可能</p>  <pre> RT_MODEL_INT_TEST_T *const INT_TEST_M; void INT_TEST_reset(void) { /* RootExportFunctionCallGenerator: * Subsystem: 'stateWriter' */ /* Outputs for Atomic Subsystem: 'stateWriter' * StateWriter: 'stateWriter' * Constant: 'stateWriter' */ Init_val = 1.0; } </pre>	<p>R2015a</p> <p>実行時、コードカバレッジ測定できる</p>   <p>未達部分</p> <p>※: Simulink Coverage™のライセンスが別途必要</p>
Clone Group 1 (2 clones, 5 blocks/clone)	Remarks								
<input checked="" type="checkbox"/> clone_sample2Subsystem	Virtual subsystem								
<input checked="" type="checkbox"/> clone_sample3Subsystem	Virtual subsystem								
<p>R2016b</p>	<p>R2015b</p>	<p>R2016b</p>	<p>R2016a</p>						

R2011b~17bまでの間で、約450件以上の組み込みコード生成に関連した機能追加・改善が実施されています

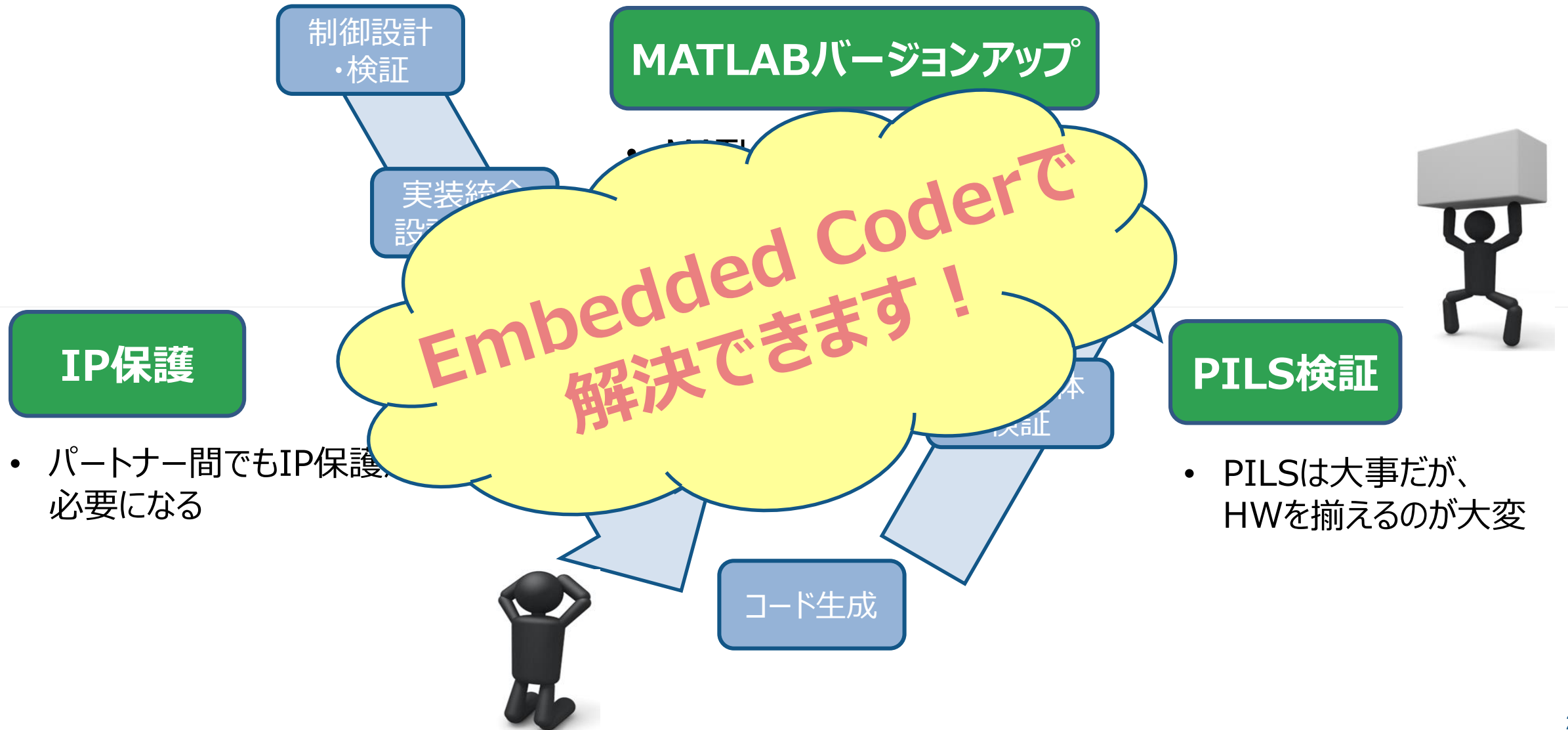
アジェンダ

1. Embedded Coderとは？
2. 自動コード生成ツール適用のメリット
3. **Embedded Coderの進化**
 - コード生成エンジンの進化
 - 機能の進化
 - **運用フローの進化**
4. まとめ

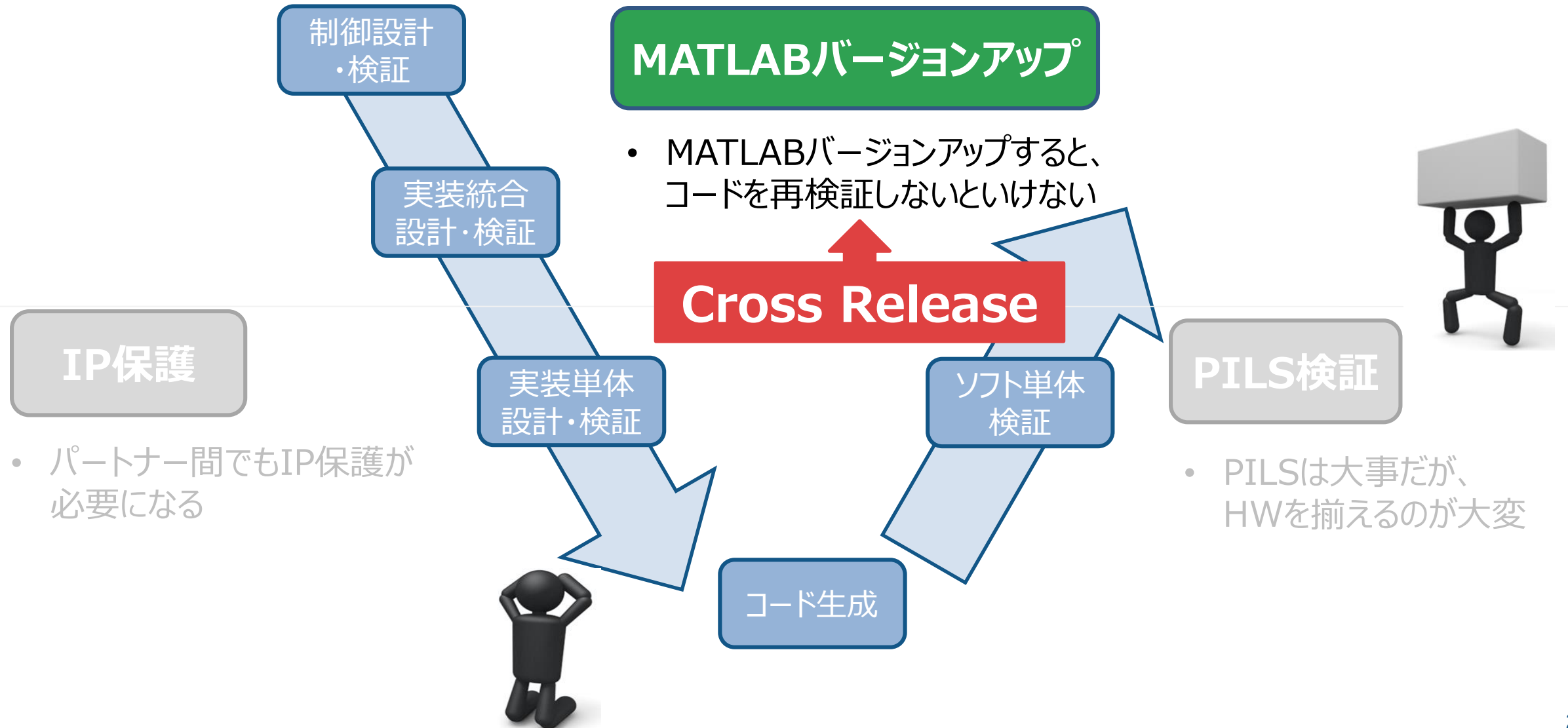
自動コード生成ツールの適用範囲拡大 / 大規模ソフト開発における課題



自動コード生成ツールの適用範囲拡大 / 大規模ソフト開発における課題



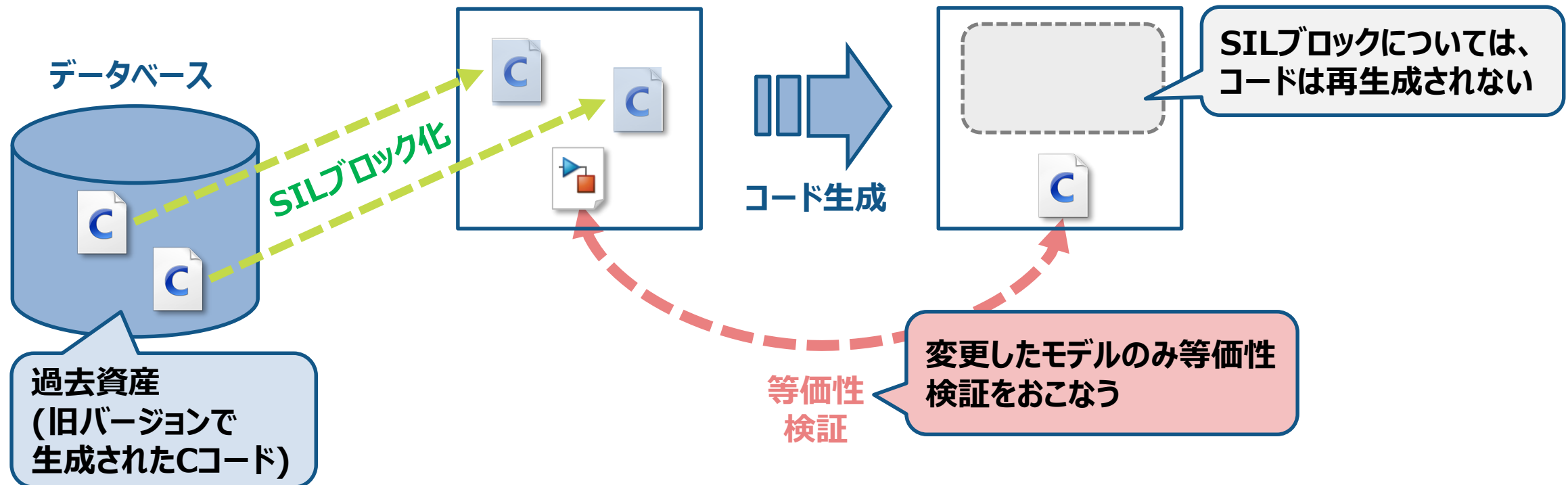
自動コード生成ツールの適用範囲拡大 / 大規模ソフト開発における課題



1. Cross Release

R2016b

- 旧バージョンで生成されたコードをSimulinkモデルに実装し、動作させることができるので、仕様変更が発生したモデルのみの検証に集中できます
- 本機能はR2010a以降で生成されたEmbedded Coderコードに対応しています



デモ: Cross Release

The screenshot displays the MATLAB R2017b environment with a Simulink model titled 'IntegratedModel'. The interface is split into several panes:

- Command Window:** Shows the prompt `fx >>`.
- Workspace:** Lists variables `data1` through `data8`, each with a value of `1x1 Signal`.
- File Explorer:** Shows the current directory `C:\Work\Expo\CrossRelease\Demo\R2017b` containing files like `createCrossReleaseModel.m`, `DataObj_Define.h`, `IntegratedModel.slx`, `romram_imToFile.mat`, `swc001.slx`, `swc002.slx`, `swc003.slx`, and `TestHarness.slx`.
- Simulink Model:** A block diagram showing three sub-models:
 - `swc001`: Receives `data1` and outputs `data2`.
 - `swc002`: Receives `data1` and `data5` as inputs and outputs `data4`.
 - `swc003`: Receives `data4`, `data8`, and `data7` as inputs and outputs `data6`.

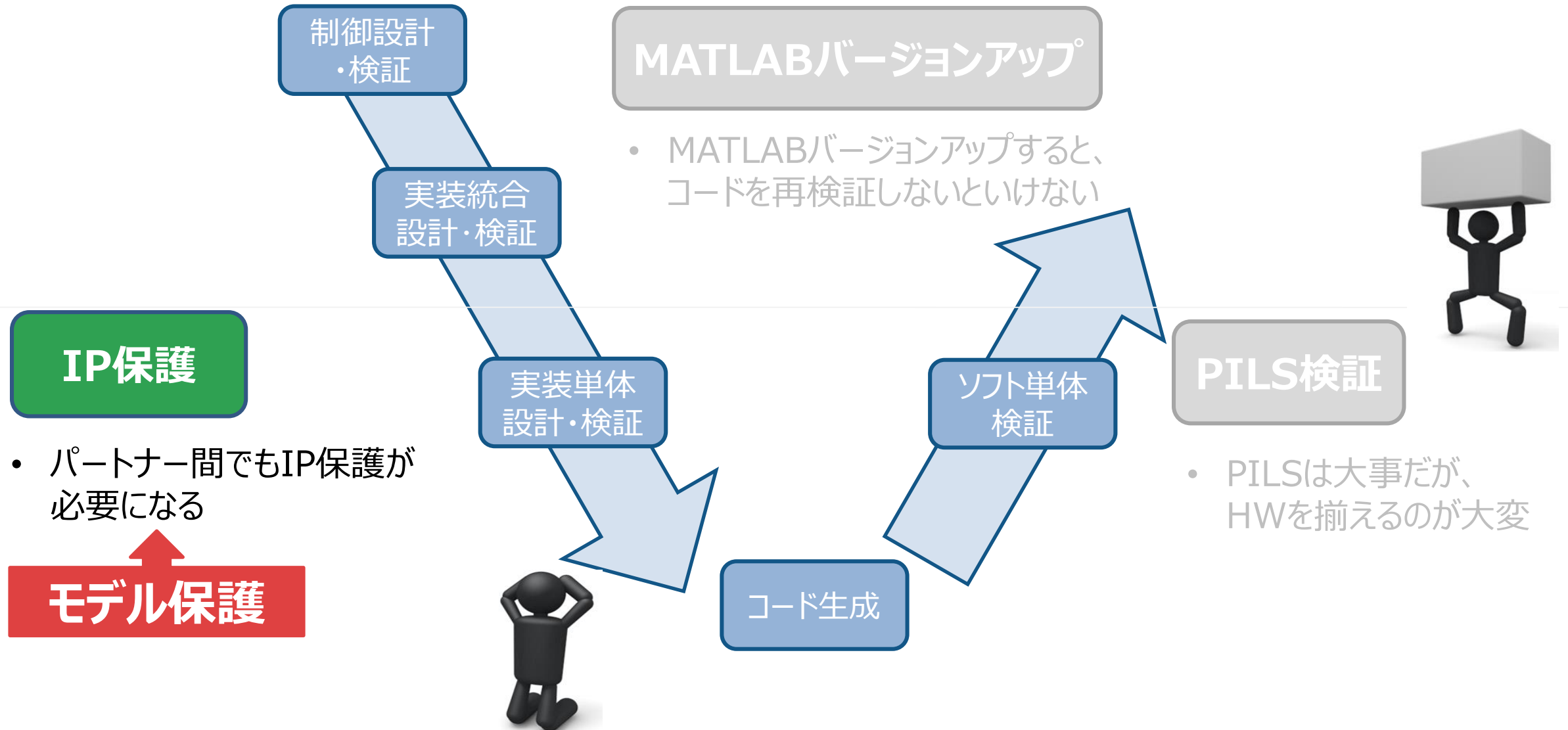
At the bottom, a recording toolbar shows a pause icon, the text '録画中...' (Recording...), a timer at '00:00:00', and icons for stop, refresh, and edit. The status bar at the bottom right indicates '準備完了' (Ready), '90%', and 'FixedStepDiscrete'.

ポイント: Cross Release

- 過去生成コードを最新リリースのMATLABでそのまま再利用できます
- Simulink上で、過去生成コードをそのままシミュレーションできます
- Embedded Coderによるコード生成時、過去生成コードが取り込まれたSILブロックからは、コードは生成されません

MATLABバージョンアップ時のコード再検証が不要になります

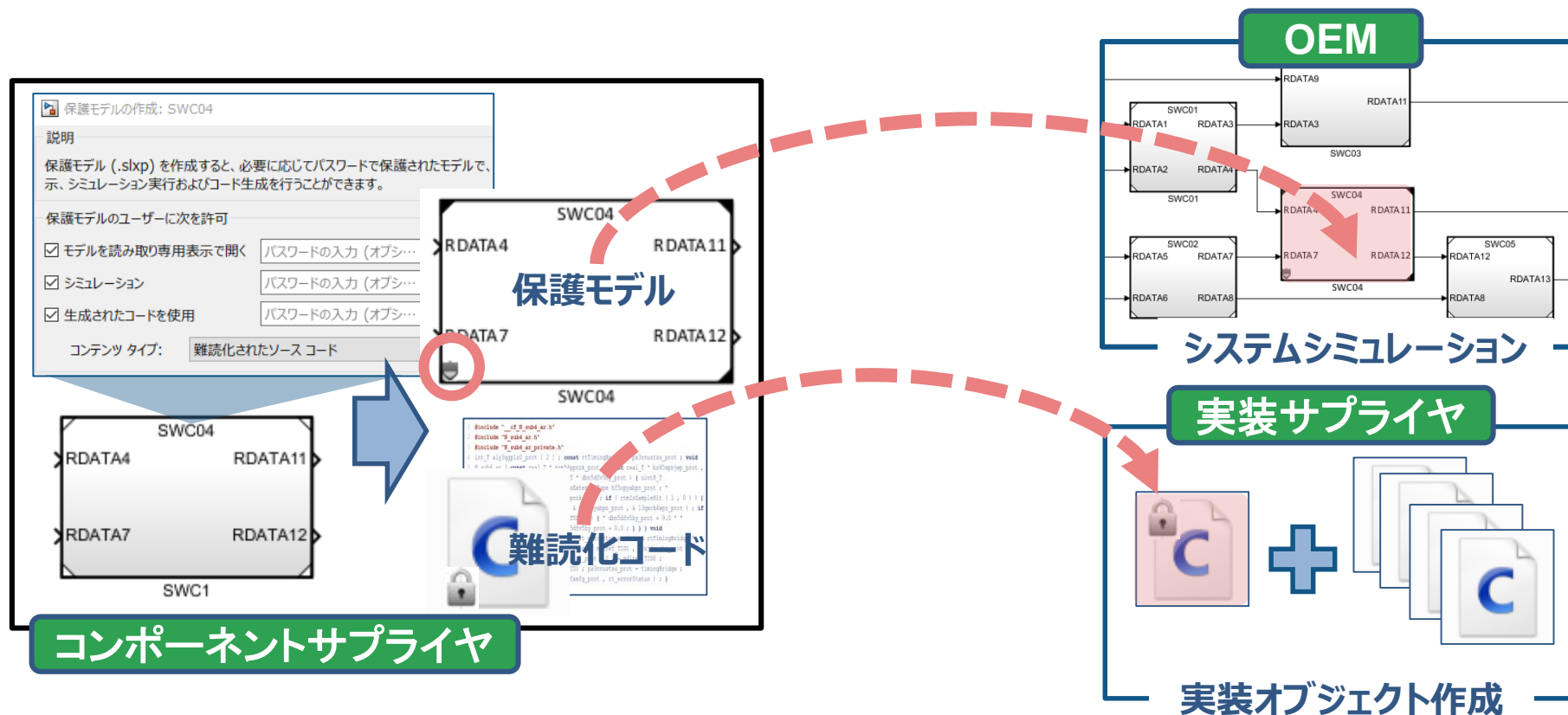
自動コード生成ツールの適用範囲拡大 / 大規模ソフト開発における課題



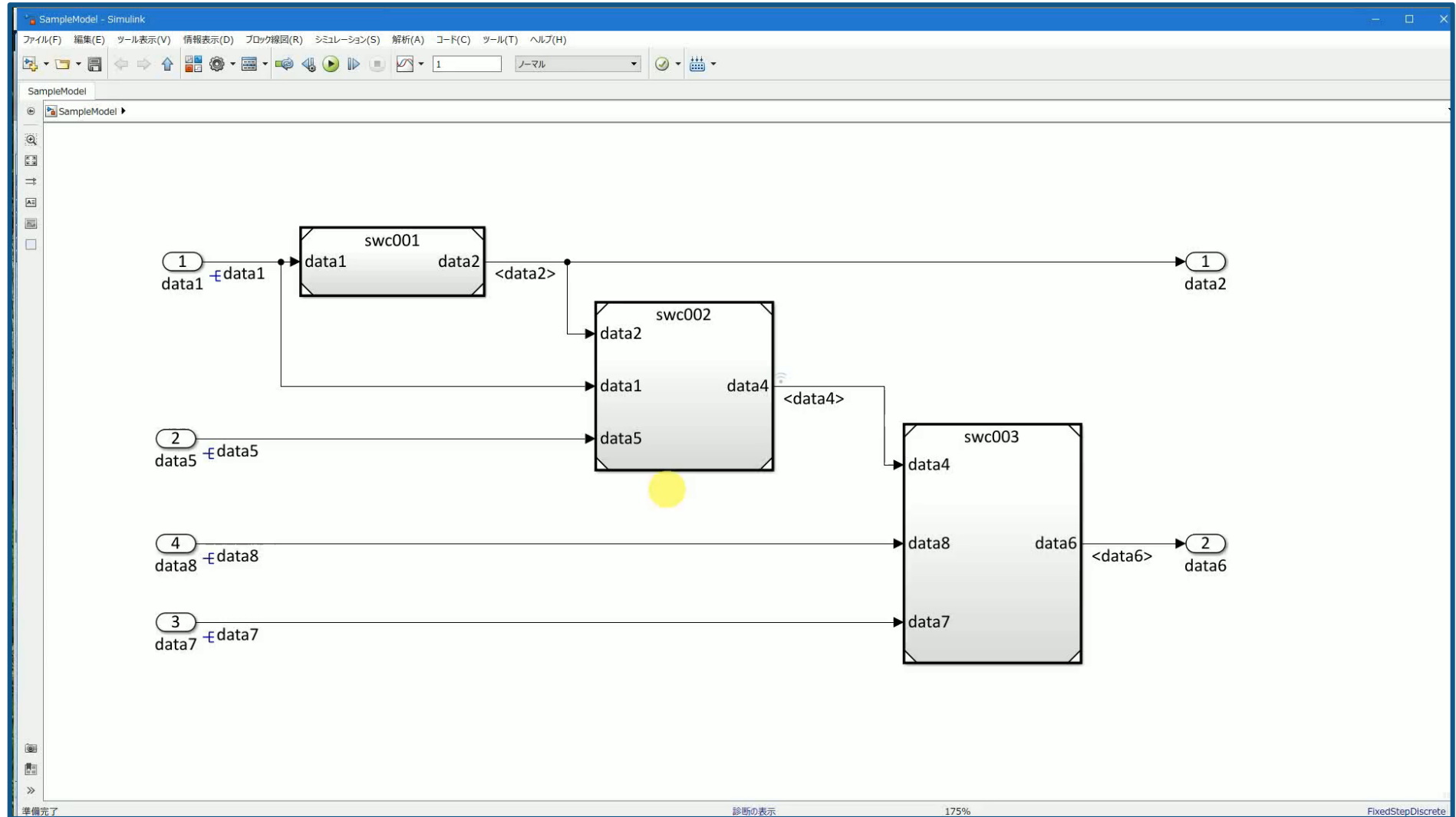
モデル保護

R2012b

- 部品の電制化が進んだことで、コンポーネントサプライヤは部品だけではなく、コントローラモデル・ソフトも開発し、OEMに納品するケースが増えてきています
- コンポーネントサプライヤと実装サプライヤが同一ではない場合は、**モデルとコードのIP保護が重要なキーポイント**となります



デモ: モデル保護

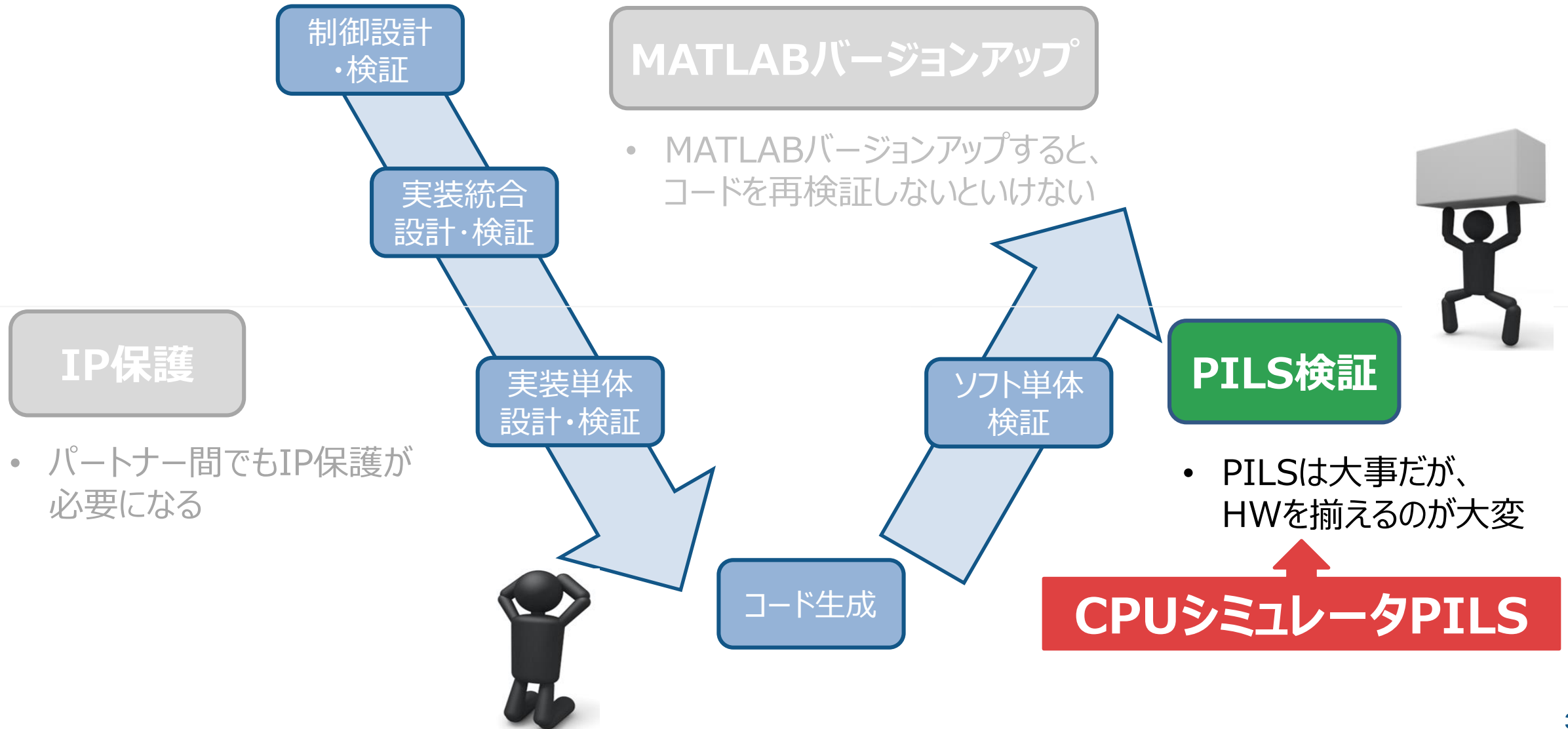


ポイント: モデル保護

- モデルの読み取り/シミュレーション/コード生成の可否をコントロールできます
- Embedded Coderによるコード生成時、生成コードの難読化有無を選択できます
- 本機能は、モデル参照ブロックのみサポートされます

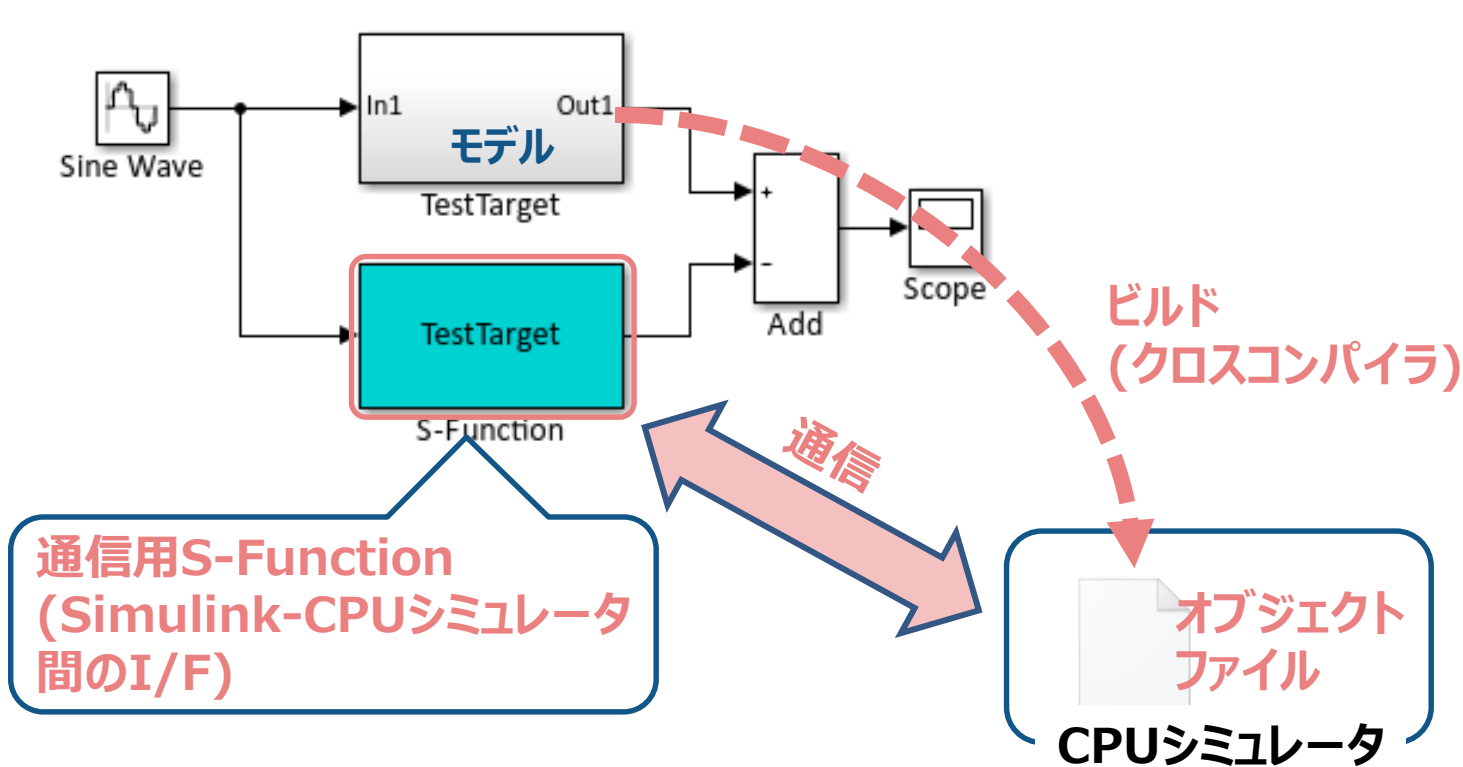
モデル提供時のIP保護を提供先に応じて細かく設定できます

自動コード生成ツールの適用範囲拡大 / 大規模ソフト開発における課題



CPUシミュレータPILS

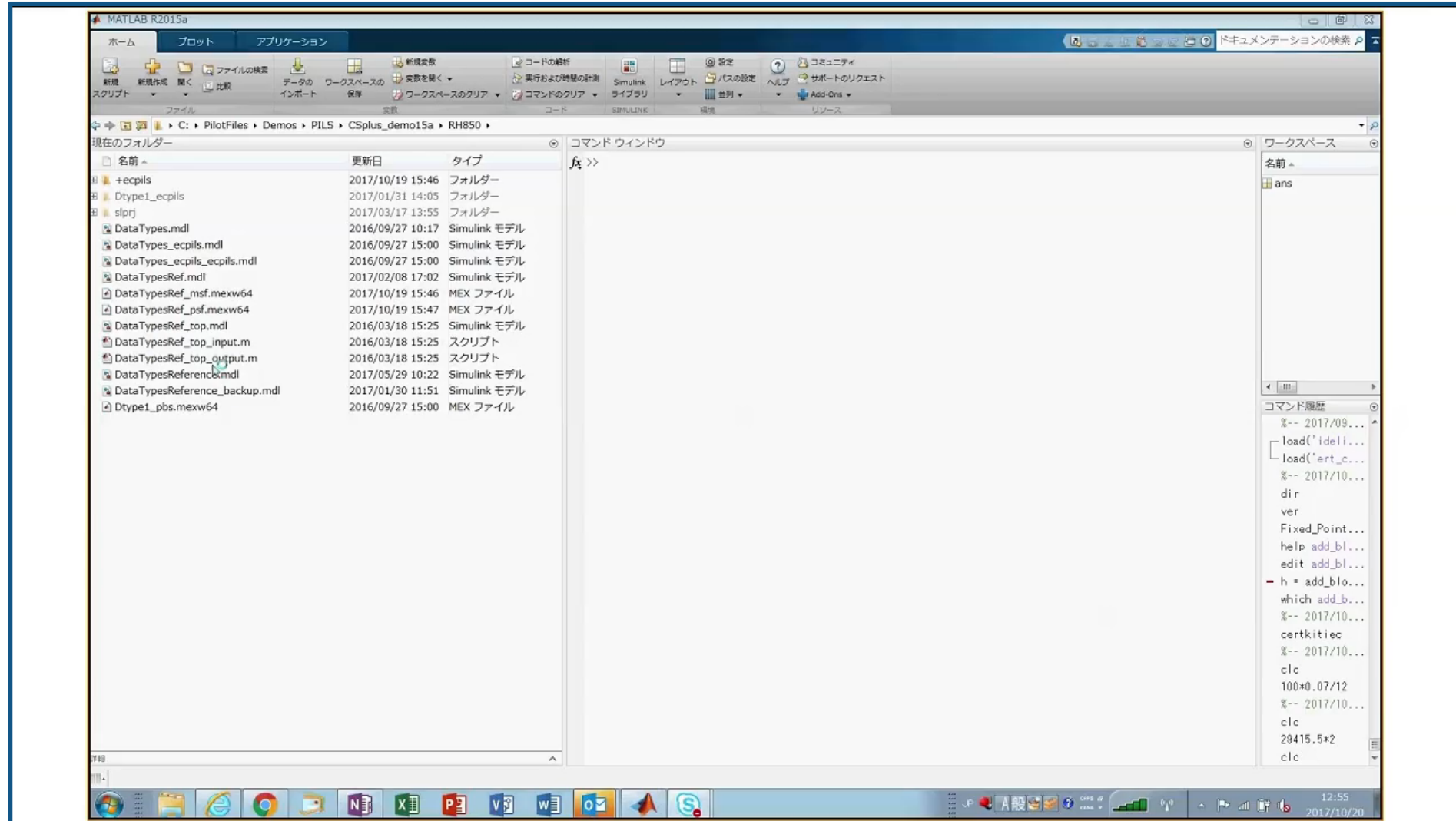
各マイコンベンダが提供するIDEのCPUシミュレータを利用することで、
評価ボードを使用せずにPC上でPILSによる検証が可能になります



対応マイコン	対応IDE, CPUシミュレーター
Renesas RH850, V850 RX/RL78, 78K0R	Renesas CS+ GreenHills MULTI Lauterbach Trace32
Renesas SH2, SH2A	Renesas HEW
ARM Cortex-M	ARM KEIL QEMU ARM エミュレータ
ARM Cortex-A	GreenHills MULTI Lauterbach Trace32 QEMU ARM エミュレータ
NXP MPC55xx	Lauterbach Trace32 NXP Semiconductors Motor Control Development Toolbox

CPUシミュレータPILSによる等価性検証イメージ

デモ: CPUシミュレータPILS

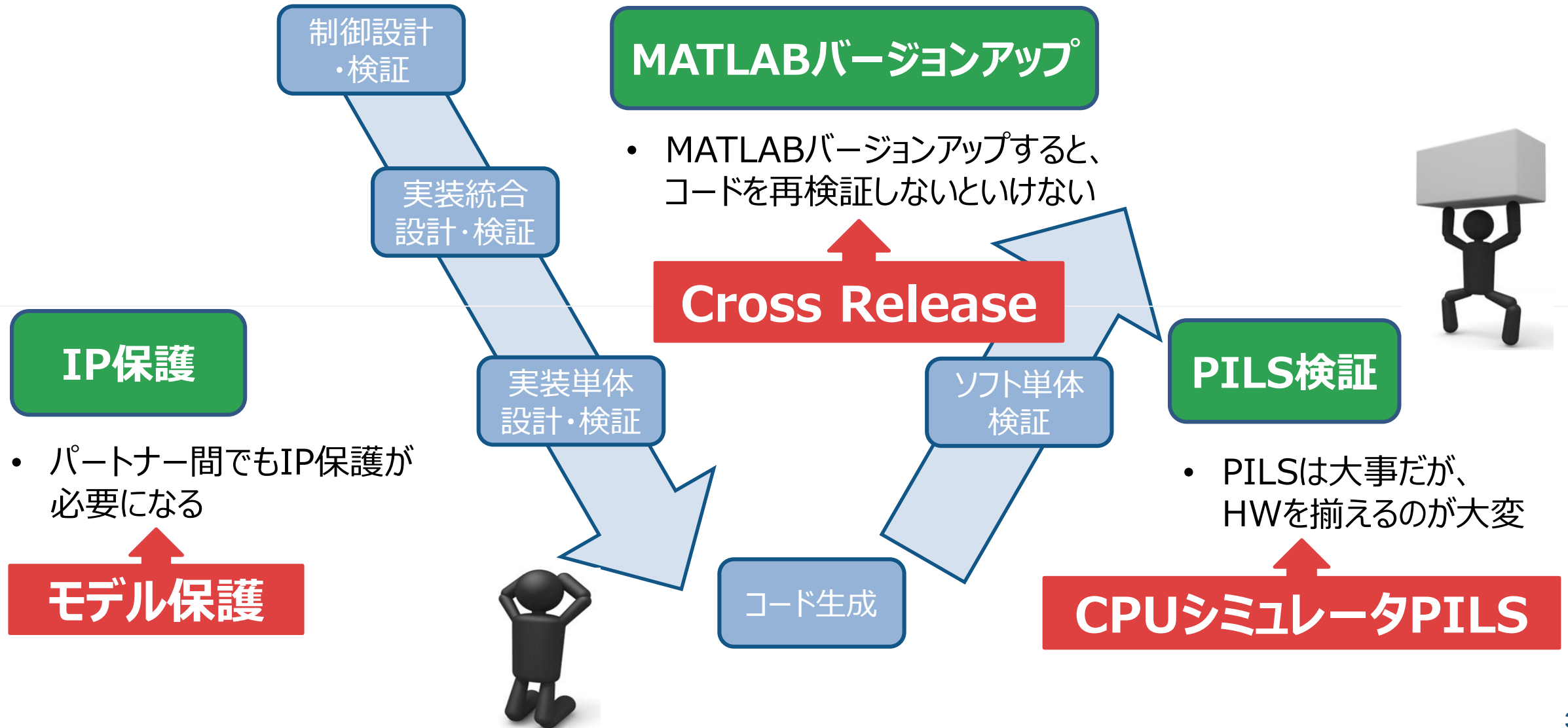


ポイント: CPUシミュレータPILS

- Simulink & Embedded Coderは、各種IDEのCPUシミュレータと連携できます
- この仕組みを利用することで、評価ボードを使用せず、PILS検証が実施できます

H/Wの空き状況に依存しないので、
大人数による開発における検証を効率化できます

自動コード生成ツールの適用範囲拡大 / 大規模ソフト開発における課題



アジェンダ

1. Embedded Coderとは？
2. 自動コード生成ツール適用のメリット
3. Embedded Coderの進化
 - コード生成エンジンの進化
 - 機能の進化
 - 運用フローの進化
4. **まとめ**

まとめ

- Embedded Coderは、様々な分野のお客様の製品開発に適用されています
- MathWorksは、Embedded Coderのバージョンアップ および 関連ツールボックスとの連携により、従来の量産コード生成プロセスに関する課題を解決してきました
- お客様の様々なユースケースにおける課題のソリューションを提供するため、MathWorksは日々、Embedded Coderを進化させつづけています

MathWorksが提供するMBDサポート

トレーニング (参考) 制御設計/組み込み系コースマップ

	制御システム設計	プラントモデル設計	ソフトウェア設計
上級			PSCC/AUTOS
			SLEC/SLHL
		MBC	SLMB/SLDV
中級	SLEX	SLEX	SLEX
		SLPM-S/SLPM-M/ SLPM-P	
	SLCT	SLCT	SLAI
	V-pro		V-pro
初級			SLRT
	SLSF-A		SLSF-A
	SLBE-A	SLBE-A	SLBE-A
	MLBE-A	MLBE-A	MLBE-A

必須 推奨 選択

MathWorks トレーニングコース

<http://www.mathworks.co.jp/services/training/courses/>

コード	コース名
MLBE-A	自動車分野向け MATLAB基礎
SLBE-A	自動車分野向け Simulink 基礎
SLSF-A	自動車分野向け Stateflow 基礎
SLRT	Simulink Coder 基礎
SLCT	MATLAB と Simulink による制御設計
SLPM-S	Simscape によるマルチドメインシステムの物理モデリング
SLPM-M	SimMechanics によるの力学系の物理モデリング
SLPM-P	SimPowerSystems(開発中)
MBC	Model -Based Calibration
SLEX	Simulink とC言語のインタフェース
SLAI	Simulink/Stateflow API
SLMB	Simulink でのモデルの管理と検証
SLDV	Simulink Design Verifier (開発中)
SLEC	Embedded Coderによる量産向けコード生成
AUTOS	AUTOSAR(0.5 day)
SLHL	HDL Coder によるHDLコード生成
PSCC	Polyspace による静的コード検証
V-pro	V-プロセス実習(オンサイトのみ)

ご清聴ありがとうございました



Accelerating the pace of engineering and science

© 2017 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.