

The slide features a dark blue header bar with the MathWorks logo on the left and "MATLAB&SIMULINK" on the right. Below the header is a large title section with a blue background image of a blue surface. The title text is centered in white.

Heterogeneous Function Composition to Eliminate a Class of Direct Relationships in Software Components of Dynamic Systems

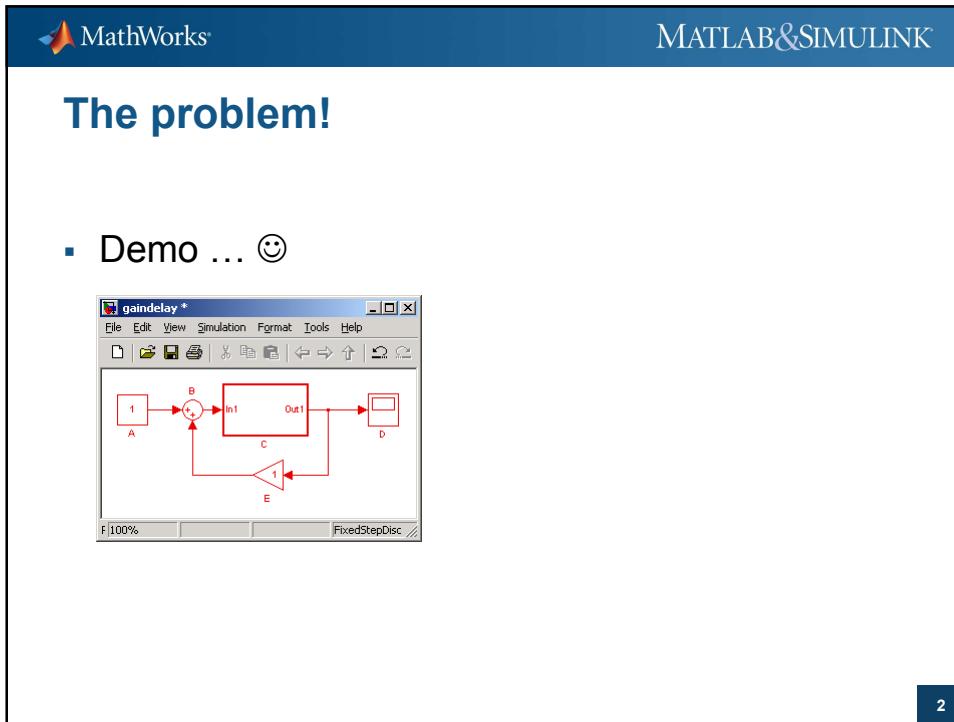
Pieter J. Mosterman

Senior Research Scientist
Design Automation Department

Adjunct Professor
School of Computer Science

  McGill

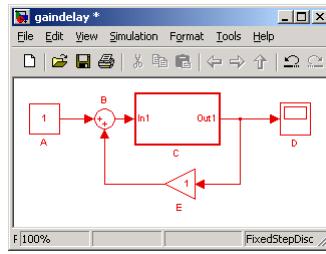
© 2010 The MathWorks, Inc.



The slide features a dark blue header bar with the MathWorks logo on the left and "MATLAB&SIMULINK" on the right. Below the header is a large title section with a white background.

The problem!

- Demo ... ☺



A screenshot of a SIMULINK model window titled "gaindelay*". The model contains several blocks: a unit delay block labeled "In1" and "Out1", a gain block labeled "C", a summing junction labeled "B", a scope block labeled "D", and a fixed-step discrete block labeled "E". The connections show a signal flow from input A through junction B to the summing junction C, then to output Out1, which is connected to the scope D. There is also a feedback path from junction B through block E back to junction B.

2

The problem!

- Demo ... ☺

Block diagram 'multi_vs_singletasking_diff_results_hidden_rt' contains an algebraic loop. The algebraic loop solver is disabled because of the current setting for Algebraic loop option in the Diagnostics page of the Configuration Parameters Dialog

3

Agenda

- - Simulink preliminaries
 - Executing a Simulink model
 - Dealing with hierarchy
 - Heterogeneous composition
 - An implementation in Simulink
 - Conclusions

4

MathWorks® MATLAB&SIMULINK

Basic Simulink syntax

- An example model ...

5

MathWorks® MATLAB&SIMULINK

Basic Simulink syntax

- Network of **blocks** with directed **lines** connected between **ports**

6

The execution hierarchy

- The lines reflect input/output relations
- The (nonvirtual) blocks are dynamic systems

Input, $u(t)$ → **Dynamic System**
State: $x(t)$, Parameters: P → Output, $y(t)$

Output : $y(t) = f(u(t), x(t), P, t)$
Derivative : $\frac{dx(t)}{dt} = g(u(t), x(t), P, t)$

Input, $u(t_k)$ → **Dynamic System**
State: $x(t_k)$, Parameters: P → Output, $y(t_k)$

Output : $y(t_k) = f(u(t_k), x(t_{k-1}), P, t_k)$
Update : $x(t_k) = g(u(t_k), x(t_{k-1}), P, t_k)$

Input, $u(t)$ → **Dynamic System**
Parameters: P → Output, $y(t)$

Algebraic : $f(u(t), y(t), P) = 0$

7

Example block implementation

- A unit delay

$u(t) = x(t)$ → **Z⁻¹** → $y(t) = x(t)$

State, $x(t)$
 $x(0^-) = P_{IC}$

u_x
 $f_{up} \leftarrow x'$
 $f_{op} \longrightarrow x$

```
class UnitDelayBlock : public Block {
public:
    ErrorStatus BlockDrawIcon() {
        // Draw '1/z' on the icon
        .....
    }
    BlockParameterData BlockGetParameterData() {
        // Return initial_condition as block data
        .....
    }
    ErrorStatus BlockOutput() {
        // Implement y(t) = x(t)
        .....
    }
    ErrorStatus BlockUpdate(){
        // Implement x(t) = u(t)
        .....
    }
private:
    double initial_condition;
};
```

8

The basic structure of a discrete time block

- Data dependencies between
 - Input signals, output signals, current state, new state, update function and output function

9

Data dependencies to create a sorted list for efficient execution

- Static dependency analysis

Sorted list:
0:0 sin
0:1 fcn
1:0 f()
0:2 z ⁻¹
0:3 Σ
0:4 out

10

MathWorks® MATLAB&SIMULINK

Algebraic dependencies

- What if we replace the delay block by a gain?

11

MathWorks® MATLAB&SIMULINK

Algebraic dependencies

- Strongly connected components

Algebraic loop solution:
 $y(t) = u(t) + Ky(t)$
 \rightarrow
 $y(t) = u(t)/(1-K)$

Sorted list:

0:0 sin
0:1 fcn
1:0 f()
0:2 AlgLoop
2:0 Σ
2:1 K
0:3 Out

Generate block sorted list

12

MathWorks® MATLAB&SIMULINK

Algebraic dependencies

- Strongly connected components

Algebraic loop solution:
 $y(t) = u(t) + K y(t)$
 \rightarrow
 $y(t) = u(t)$

No embedded
code generation!

Sorted list:

```

0:0 sin
0:1 fcn
1:0 f()
0:2 AlgLoop
2:0 Σ
2:1 K
0:3 Out
    
```

Generate block sorted list

13

MathWorks® MATLAB&SIMULINK

Agenda

- Simulink preliminaries
- Executing a Simulink model
- Dealing with hierarchy
- Heterogeneous composition
- An implementation in Simulink
- Conclusions

14

MathWorks® MATLAB&SIMULINK

Let's assume single-tasking execution

- Allows multi-rate systems
- All blocks run in a single task
 - Single execution time line
 - Base rate at greatest common denominator
 - Blocks execute when they have a sample hit
 - No data integrity issues

15

MathWorks® MATLAB&SIMULINK

A simulation algorithm for networks of dynamic systems

```

graph TD
    Initialize[Initialize] --> Decision{Time < Stop time}
    Decision -- N --> SimulationComplete[Simulation complete]
    Decision -- Y --> OutputMethods[Execute list of active output methods]
    OutputMethods --> UpdateMethods[Execute list of active update methods]
    UpdateMethods --> Integrate[Integrate]
    Integrate --> TimeUpdate[Time = Time + Step size]
    TimeUpdate --> Decision
  
```

Dynamic System

Input, $u(t)$ Output, $y(t)$

State: $x(t)$, Parameters: P

Output : $y(t) = f_o(u(t), x(t), P, t)$

Update : $x_d(t) = f_u(u(t), x(t), P, t)$

Derivative : $\frac{dx_c(t)}{dt} = f_d(u(t), x(t), P, t)$

16

MathWorks® MATLAB&SIMULINK

First generate the sorted list

- Determine data dependencies
 - Based on a direct feedthrough (`df`) flag

0:0	F
0:1	E
0:2	D
0:3	A
0:4	B
0:5	C

17

MathWorks® MATLAB&SIMULINK

Generate execution lists for all of the block methods

Sorted list:

0:0 F
0:1 E
0:2 D
0:3 A
0:4 B
0:5 C

Output execution list:

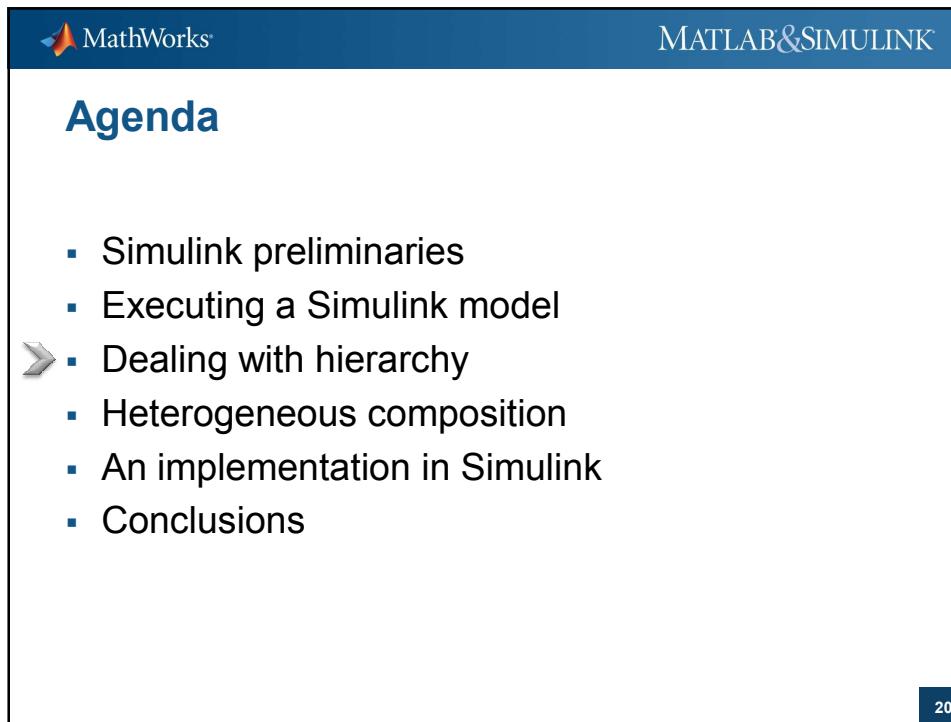
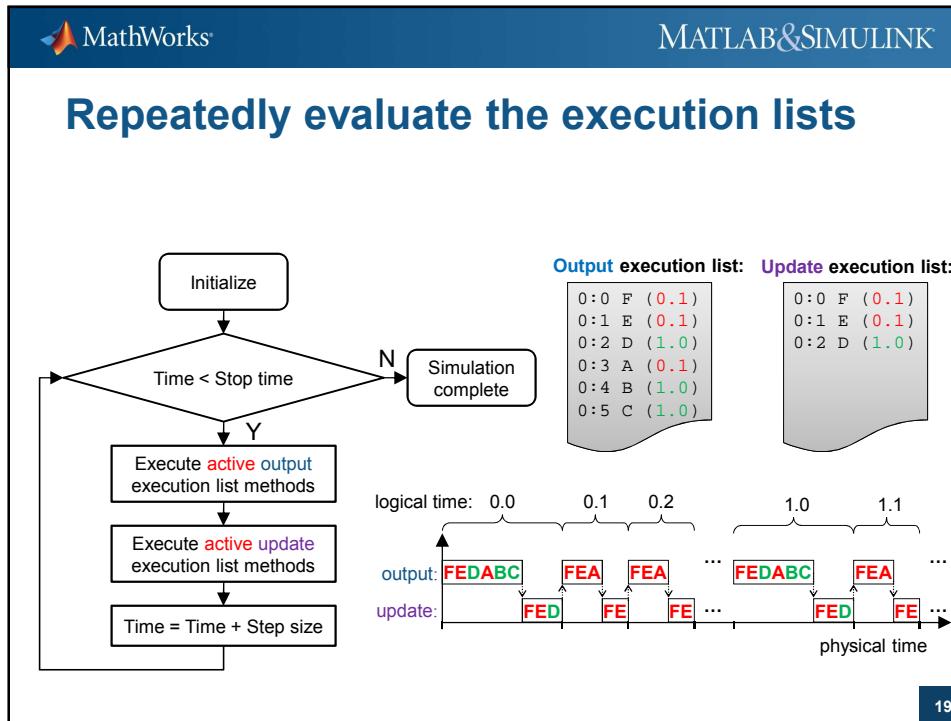
0:0 F (0.1)
0:1 E (0.1)
0:2 D (1.0)
0:3 A (0.1)
0:4 B (1.0)
0:5 C (1.0)

Update execution list:

0:0 F (0.1)
0:1 E (0.1)
0:2 D (1.0)

Generate block method execution lists →

18



Hierarchy

- **Virtual** blocks to organize graphical hierarchy
 - Referential transparency; no semantic bearing
- **Nonvirtual** blocks to organize
 - Execution hierarchy
 - Data scope hierarchy

21

A graphical hierarchy

- Group blocks in a **virtual** subsystem
 - Subsystem C does not appear in the sorted list

Sorted list:

```
0:0 Delay
0:1 D
0:2 A
0:3 E
0:4 B
0:5 Gain
```

Output execution list:

```
0:0 Delay
0:1 D
0:2 A
0:3 E
0:4 B
0:5 Gain
```

Update execution list:

```
0:0 Delay
```

Generate block method execution lists

22

MathWorks® MATLAB&SIMULINK

Let's create a component ...

- Make the virtual subsystem **nonvirtual**
 - Becomes a dynamic system in its own right

Input, $u(t)$ → **Dynamic System** State: $x(t)$, Parameters: P → Output, $y(t)$

Output : $y(t) = f_o(u(t), x(t), P, t)$

Update : $x_d(t) = f_u(u(t), x(t), P, t)$

Derivative : $\frac{dx_c(t)}{dt} = f_d(u(t), x(t), P, t)$

- Does that affect sorting?

23

MathWorks® MATLAB&SIMULINK

Let's start with a virtual subsystem ...

- Dependencies derived from **df** flag

24

MathWorks® MATLAB&SIMULINK

Make subsystem an atomic component

- Direct feedthrough moves to component level!

The diagram shows a Simulink model with five blocks labeled A, B, C, D, and E. Block A is a unit delay block. Block B is a summing junction with a gain of 1. Block C is a scope block. Block D is a unit delay block. Block E is a gain block with a value of 1. A signal flows from A to B. The output of B goes to both C and D. The output of D goes to E. The output of E goes back to B. The output of C is labeled 'Out'.

25

MathWorks® MATLAB&SIMULINK

Make subsystem an atomic component

- Now we have a dependency cycle!

The diagram shows a Simulink model with five blocks labeled A, B, C, D, and E. Block A is a unit delay block. Block B is a summing junction with a gain of 1. Block C is a scope block. Block D is a unit delay block. Block E is a gain block with a value of 1. A signal flows from A to B. The output of B goes to both C and D. The output of D goes to E. The output of E goes back to B. The output of C is labeled 'Out'. A red box highlights the subsystem containing blocks B, C, and D. A red arrow points from the output of block D back to its input, indicating a dependency cycle.

26

MathWorks® MATLAB&SIMULINK

Make subsystem an atomic component

- Now we have a dependency cycle!

How did the direct feedthrough flag end up there?! And how can we fix it?!

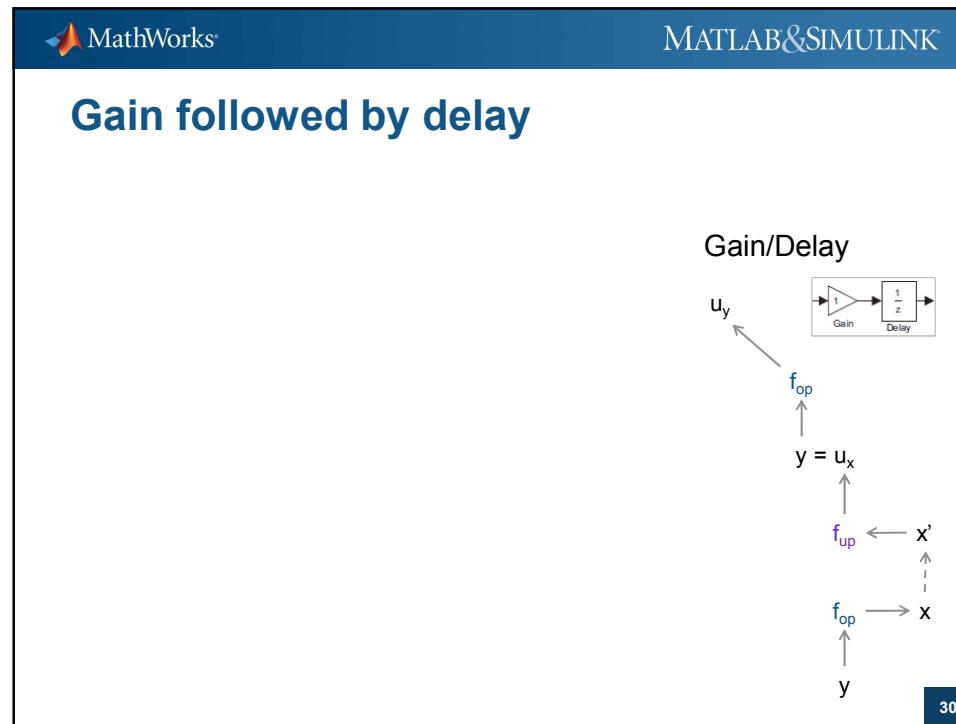
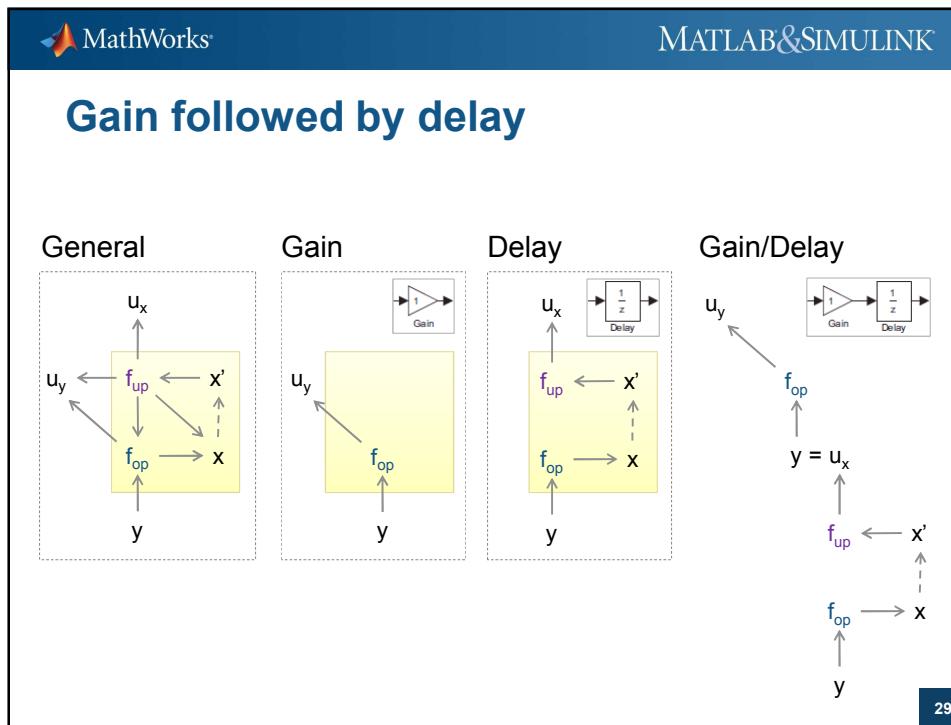
27

MathWorks® MATLAB&SIMULINK

Agenda

- Simulink preliminaries
- Executing a Simulink model
- Dealing with hierarchy
- Heterogeneous composition
- An implementation in Simulink
- Conclusions

28



MathWorks® MATLAB&SIMULINK

Gain followed by delay

Gain/Delay

The diagram shows a block diagram consisting of two blocks: a 'Gain' block and a 'Delay' block. The 'Gain' block has a value of 1. The 'Delay' block has a transfer function of $\frac{1}{z}$. An input signal enters the 'Gain' block, and its output enters the 'Delay' block. The output of the 'Delay' block is the final output.

u_y

f_{op}

f_{up}

x'

x

y

31

MathWorks® MATLAB&SIMULINK

Gain followed by delay

The diagram shows a block diagram consisting of two blocks: a 'Gain' block and a 'Delay' block. The 'Gain' block has a value of 1. The 'Delay' block has a transfer function of $\frac{1}{z}$. An input signal y enters the 'Gain' block, and its output enters the 'Delay' block. The output of the 'Delay' block is the final output u_y .

u_y

f_{op}

f_{up}

x'

x

y

32

MathWorks® MATLAB&SIMULINK

How do we create a component?

The diagram shows two sub-blocks at the top:

- Left sub-block: A Gain block (Gain = 1) followed by a Delay block ($\frac{1}{z}$).
- Right sub-block: An In block (labeled 'In') followed by a Gain block (Gain = 1), then a Delay block ($\frac{1}{z}$), and finally an Out block (labeled 'Out').

Below these, a flowchart illustrates the composition process:

- Initial State:** Inputs y and x enter the system. y is processed by f_{op} to produce u_y . x is processed by f_{up} to produce x' .
- Intermediate State:** u_y and x' are combined to form a new input u_x .
- Final State:** u_x is processed by f_{up} to produce x' , which is then processed by f_{op} to produce u_y . Simultaneously, y is processed by f_{op} to produce x , which is then processed by f_{up} to produce x' .
- A red question mark indicates the final step of creating the component block.

33

MathWorks® MATLAB&SIMULINK

Homogeneous composition

The diagram shows two sub-blocks at the top:

- Left sub-block: A Gain block (Gain = 1) followed by a Delay block ($\frac{1}{z}$).
- Right sub-block: An In block (labeled 'In') followed by a Gain block (Gain = 1), then a Delay block ($\frac{1}{z}$), and finally an Out block (labeled 'Out').

Below these, a flowchart illustrates the composition process:

- Initial State:** Inputs y and x enter the system. y is processed by f_{op} to produce u_y . x is processed by f_{up} to produce x' .
- Intermediate State:** u_y and x' are combined to form a new input u_y .
- Final State:** u_y is processed by f_{op} to produce u_y , and x is processed by F_{op} to produce x .

34

MathWorks® MATLAB&SIMULINK

Put the component in context

The diagram shows a block diagram of a feedback control system. A reference input 'A' (labeled 1) enters a summing junction 'B'. The output of 'B' is fed into a 'Gain' block, which has a feedback path from its output back to 'B'. The output of the 'Gain' block goes to a 'Delay' block (labeled $\frac{1}{z}$). The output of the 'Delay' block is labeled 'Out'. A feedback signal from 'Out' is fed through a gain of 1 to a summing junction 'C'. The output of 'C' is fed through a gain of 1 to another summing junction 'B'. The output of 'B' is labeled 'D'. The output of 'C' is also fed through a gain of 1 to a final summing junction 'E'. The output of 'E' is labeled 'D'.

Below the diagram, a dependency graph is shown with three nodes A, B, and C. Node A contains the equation $x \leftarrow f_{op}$. Node B contains the equation $u_y \leftarrow f_{op}$ and $f_{up} \leftarrow x'$. Node C contains the equation $F_{op} \rightarrow x$ and $y \leftarrow$. Arrows indicate dependencies: f_{op} in A points to f_{op} in B; x' in B points to f_{up} in B; F_{op} in C points to x in C; and y in C points to f_{op} in B.

35

MathWorks® MATLAB&SIMULINK

We have a dependency cycle!

The diagram is identical to the one above, showing the same block diagram and dependency graph. However, red arrows highlight the cycle in the dependency graph. Red arrows point from f_{op} in node A to f_{op} in node B, and from f_{op} in node B back to f_{op} in node A, forming a closed loop.

36

MathWorks® MATLAB&SIMULINK

Is there another way ... ?

Block Diagrams:

- Left: Gain (1) —> Delay ($\frac{1}{z}$)
- Right: In —> Gain (1) —> Delay ($\frac{1}{z}$) —> Out

State-Space Models:

- f_{op} : $u_y \leftarrow f_{op} \rightarrow x$
- f_{up} : $u_y \leftarrow f_{up} \leftarrow x' \leftarrow f_{op} \rightarrow x \leftarrow y$

Result:

- $u_x \leftarrow f_{up} \leftarrow x' \leftarrow f_{op} \rightarrow x \leftarrow y$
- $u_y \leftarrow f_{op} \rightarrow x$

?

37

MathWorks® MATLAB&SIMULINK

Heterogeneous composition!

Block Diagrams:

- Left: Gain (1) —> Delay ($\frac{1}{z}$)
- Right: In —> Gain (1) —> Delay ($\frac{1}{z}$) —> Out

State-Space Models:

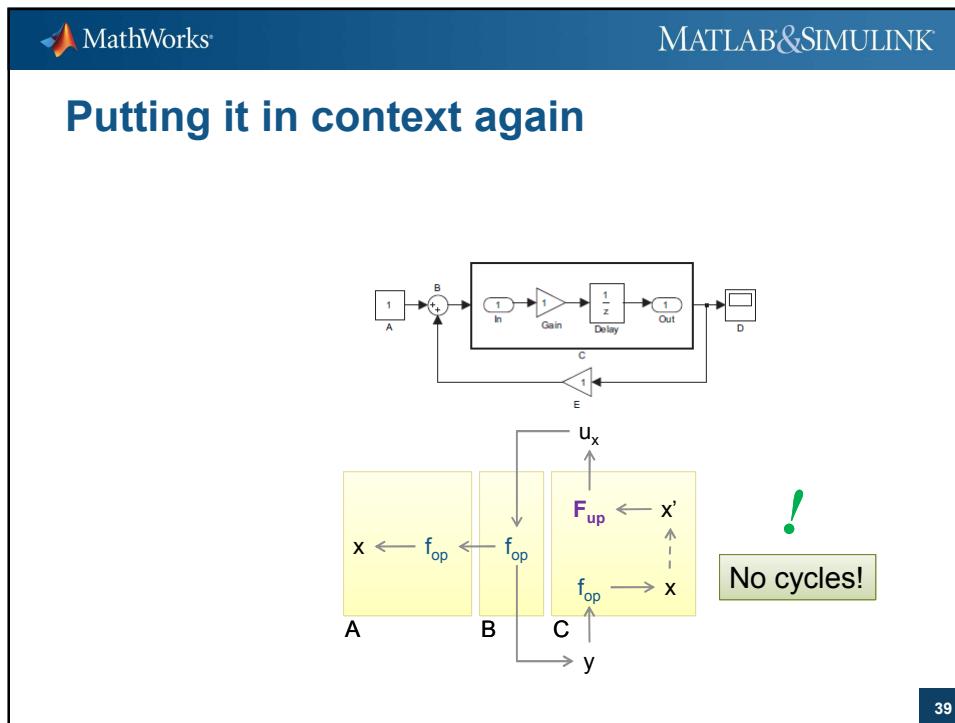
- f_{op} : $u_y \leftarrow f_{op} \rightarrow x$
- f_{up} : $u_y \leftarrow f_{up} \leftarrow x' \leftarrow f_{op} \rightarrow x \leftarrow y$

Result:

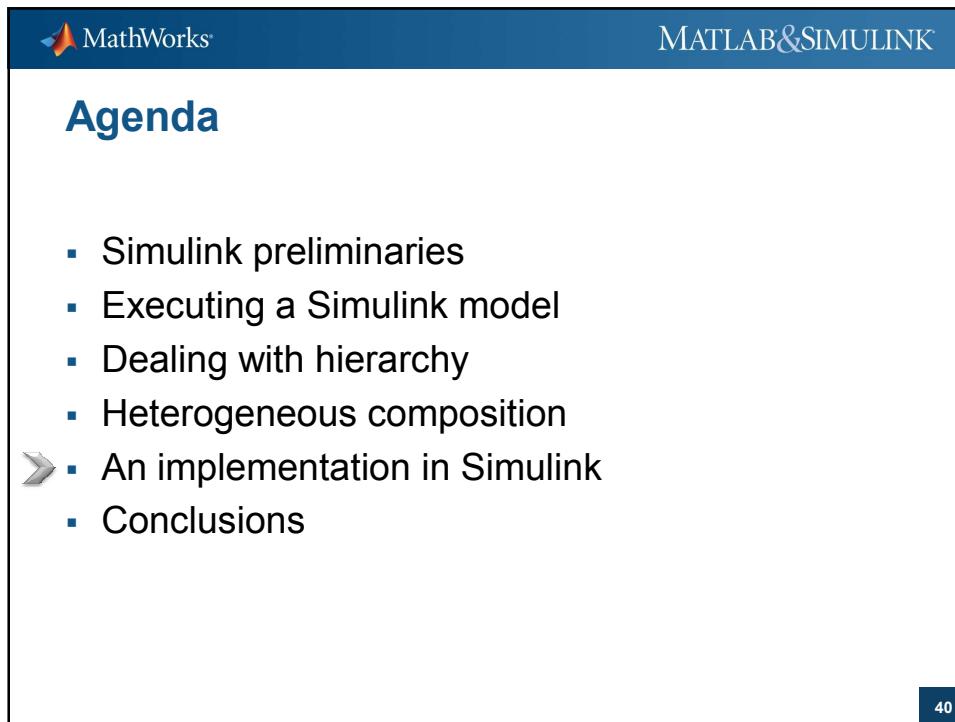
- $u_x \leftarrow F_{up} \leftarrow x' \leftarrow f_{op} \rightarrow x \leftarrow y$
- $u_y \leftarrow f_{op} \rightarrow x$

?

38



39



40

MathWorks® MATLAB&SIMULINK

How do we fit this into the model compilation machinery?

- Create sorted list after clearing **df** flag

41

MathWorks® MATLAB&SIMULINK

How do we fit this into the model compilation machinery?

- Create sorted list after

42

MathWorks® MATLAB&SIMULINK

How do we fit this into the model compilation machinery?

- Create sorted list after clearing **df** flag

Sorted list:

0:0 C
0:0 Gain
0:1 Delay
0:1 A
0:2 E
0:3 B

- Mark where output for update should be used

Sorted list:

0:0 C
0:0 Gain*
0:1 Delay
0:1 A
0:2 E
0:3 B

Output execution list:

0:0 C(op)
0:0 Delay(op)
0:1 A(op)
0:2 E(op)
0:3 B(op)

Update execution list:

0:0 C(up)
0:0 Gain(op)
0:1 Delay(up)

43

MathWorks® MATLAB&SIMULINK

What does the generated code look like?

```

56 /* Model output function */
57 void gaindelay_output(void)
58 {
59     real_T rtb_E;
60
61     /* Outputs for atomic SubSystem: '<Root>/C' */
62     gaindelay_C();
63
64     /* end of Outputs for SubSystem: '<Root>/C' */
65
66     /* Gain: '<Root>/E' */
67     rtb_E = gaindelay_P_E_Gain * gaindelay_B.Delay;
68
69     /* Sum: '<Root>/B' incorporates:
70      * Constant: '<Root>/A'
71      */
72     gaindelay_B.B = gaindelay_P_A_Value + rtb_E;
73 }

```

Output execution list:

0:0 C(op)
0:0 Delay(op)
0:1 A(op)
0:2 E(op)
0:3 B(op)

Update execution list:

0:0 C(up)
0:0 Gain(op)
0:1 Delay(up)

44

MathWorks® MATLAB&SIMULINK

What does the generated code look like?

```

56 /* Model output function */
57 void gaindelay_output(void)
58 {
59     real_T rtb_E;
60
61     /* Outputs for atomic SubSystem: '<Root>/C' */
62     gaindelay_C();
63
64     /* end of Output */ 37 /* Outputs for atomic system: '<Root>/C' */
65     /* Gain: '<Root>' */ 38 void gaindelay_C(void)
66     /* Gain: '<Root>' */ 39 {
67     rtb_E = gaindel 40 /* UnitDelay: '<S1>/Delay' */
68     rtb_E = gaindelay_B.Delay = gaindelay_DWork.Delay_DSTATE;
69
70     /* Sum: '<Root>' */ 41 gaindelay_B.Delay = gaindelay_DWork.Delay_DSTATE;
71     /* Constant: '<Root>/A' */ 42 }
72     gaindelay_B.B = gaindelay_P.A_Value + rtb_E;
73 }

```

Output execution list:

0:0 C(op)
0:0 Delay(op)
0:1 A(op)
0:2 E(op)
0:3 B(op)

Update execution list:

0:0 C(up)
0:0 Gain(op)
0:1 Delay(up)

45

MathWorks® MATLAB&SIMULINK

What does the generated code look like?

```

44 /* Update for atomic system: '<Root>/C' */
45 void gaindelay_C_Update(void)
46 {
47     real_T rtb_Gain;
48
49     /* Gain: '<S1>/Gain' */ 50 rtb_Gain = gaindelay_P.Gain_Gain * gaindelay_B.B;
50
51     /* Update for UnitDelay: '<S1>/Delay' */ 52 gaindelay_DWork.Delay_DSTATE = rtb_Gain;
52
53 }
54

```

Output execution list:

0:0 C(op)
0:0 Delay(op)
0:1 A(op)
0:2 E(op)
0:3 B(op)

Update execution list:

0:0 C(up)
0:0 Gain(op)
0:1 Delay(up)

46

The general analysis

- For an import block
 - Depth-first search for a **df** path to output
 - If a port **without df** is reached, mark visited nodes for potential move of output method
- If output found
 - Clear **all** blocks visited from the initial input port

47

The general analysis

- For an import block
 - Depth-first search for a **df** path to output
 - If a port **without df** is reached, mark visited nodes for potential move of output method
- If output found
 - Clear **all** blocks visited from the initial input port

48

The general analysis

- For an import block
 - Depth-first search for a **df** path to output
 - If a port **without df** is reached, mark visited nodes for potential move of output method
- If output found
 - Clear **all** blocks visited from the initial input port

49

The general analysis

- For an import block
 - Depth-first search for a **df** path to output
 - If a port **without df** is reached, mark visited nodes for potential move of output method
- If output found
 - Clear **all** blocks visited from the initial input port

50

The synthesis part of the algorithm

- Move marked **df** blocks into an atomic subsystem

Sorted list:

```

0:0 A
0:1 C
1:0 TmpSyn...
0:0 Gain
0:1 Gain1
0:2 Gain2
1:1 Delay
1:2 Delay1
    
```

Output execution list:

```

0:0 A(op)
0:1 C(op)
1:0 Delay(op)
1:1 Delay1(op)
    
```

Update execution list:

```

0:0 C(up)
0:0 TmpSyn...(op)
0:0 Gain(op)
0:1 Gain1(op)
0:2 Gain2(op)
0:1 Delay(up)
0:2 Delay1(up)
    
```

51

Nested cyclic dependencies

Sorted list:

```

0:0 A
0:1 C
1:0 C1
0:0 C2
0:0 Gain3
0:1 Delay3
0:1 B2
0:2 D2
1:1 Delay1
0:2 D
0:3 B
    
```

Output execution list:

```

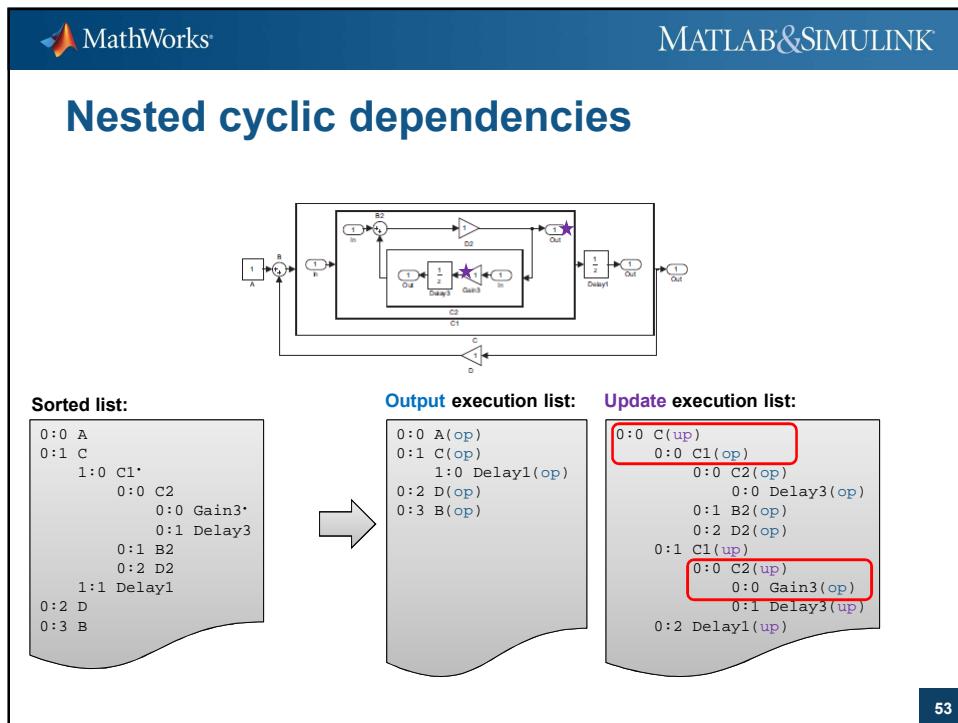
0:0 A(op)
0:1 C(op)
1:0 Delay1(op)
0:2 D(op)
0:3 B(op)
    
```

Update execution list:

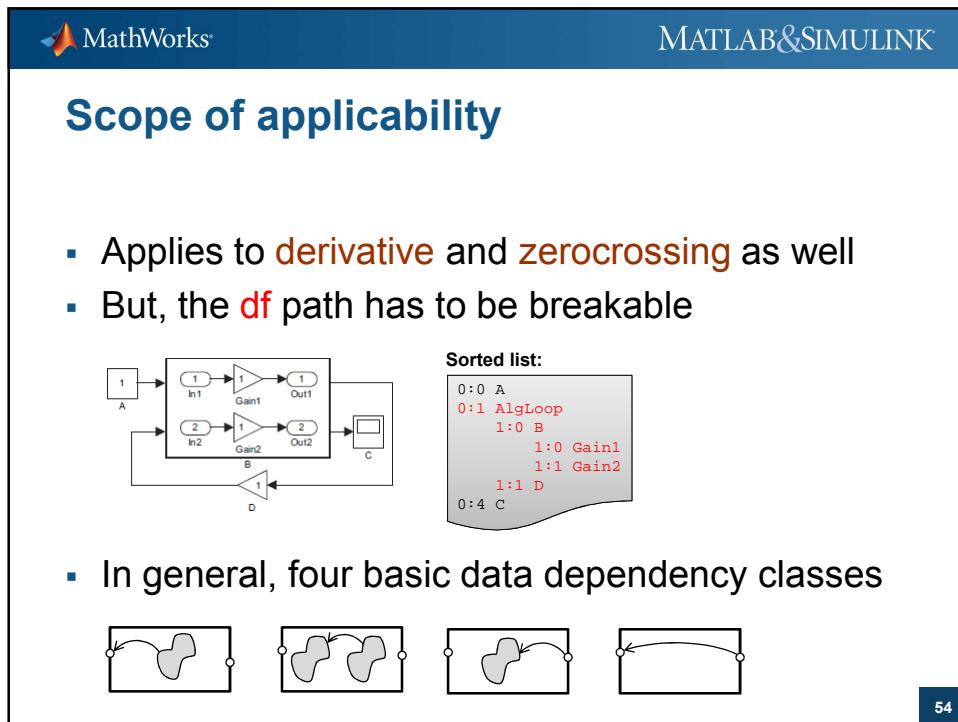
```

0:0 C(up)
0:0 C1(op)
0:0 C2(op)
0:0 Delay3(op)
0:1 B2(op)
0:2 D2(op)
0:1 C1(up)
0:0 C2(up)
0:0 Gain3(op)
0:1 Delay3(up)
0:2 Delay1(up)
    
```

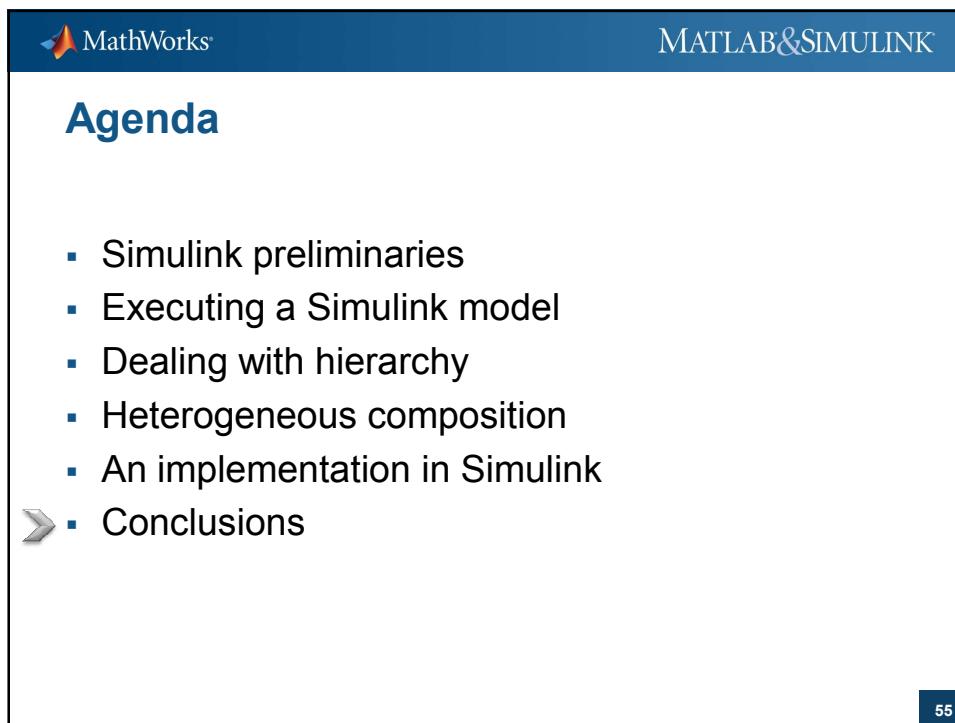
52



53



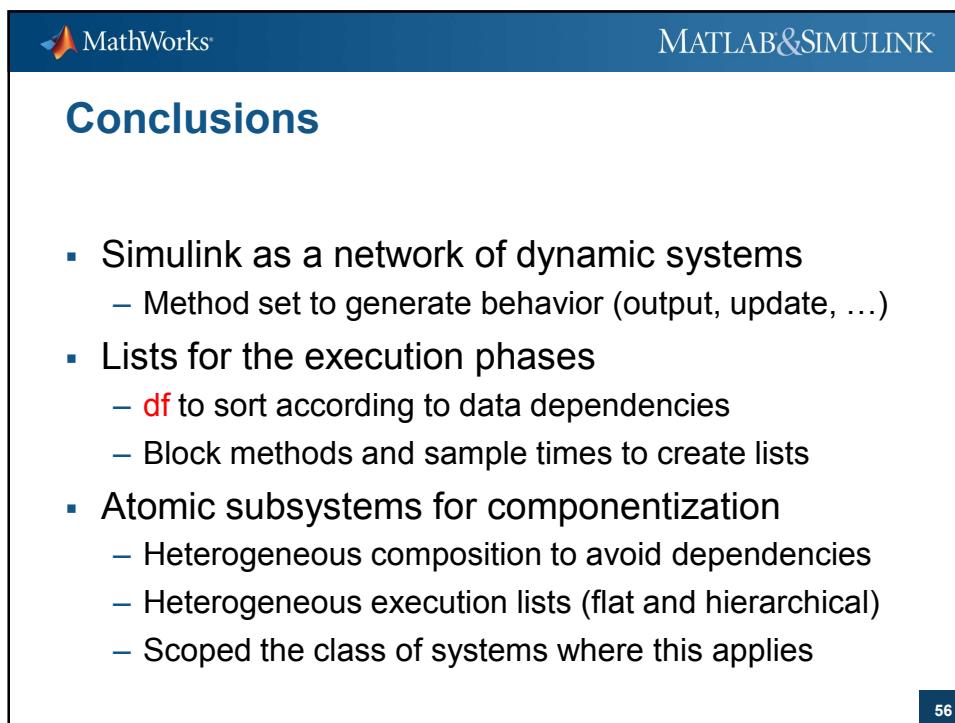
54



The slide has a blue header bar with the MathWorks logo and "MATLAB&SIMULINK" text. The main content area is titled "Agenda" in large blue text. Below it is a bulleted list of topics, with the last item preceded by a grey arrow icon pointing right. A small blue box in the bottom right corner contains the number "55".

Agenda

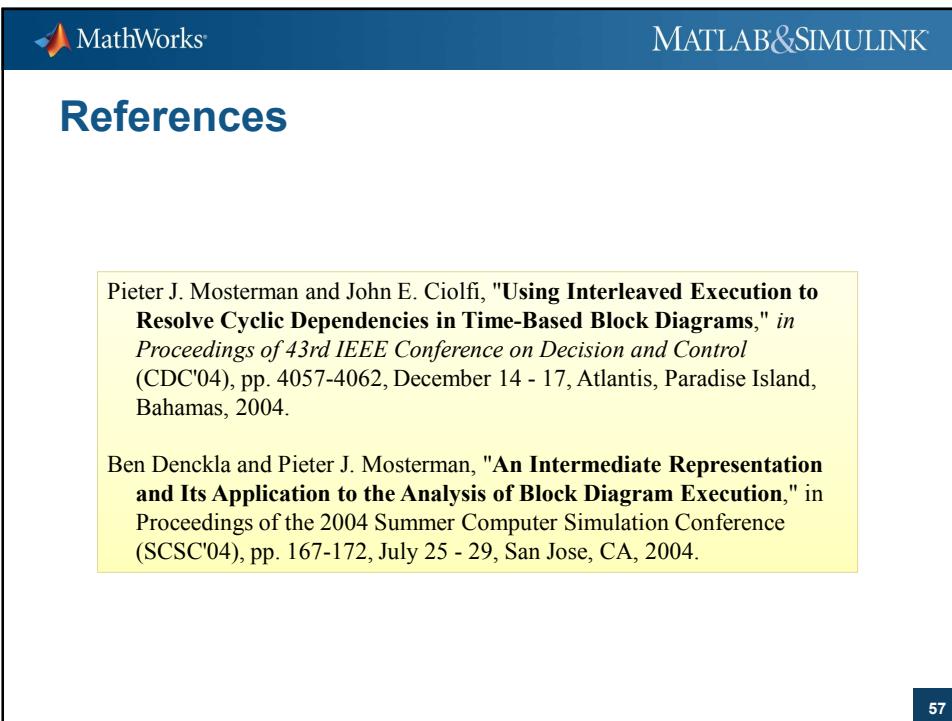
- Simulink preliminaries
- Executing a Simulink model
- Dealing with hierarchy
- Heterogeneous composition
- An implementation in Simulink
- ▶ Conclusions



The slide has a blue header bar with the MathWorks logo and "MATLAB&SIMULINK" text. The main content area is titled "Conclusions" in large blue text. Below it is a bulleted list of points. A small blue box in the bottom right corner contains the number "56".

Conclusions

- Simulink as a network of dynamic systems
 - Method set to generate behavior (output, update, ...)
- Lists for the execution phases
 - **df** to sort according to data dependencies
 - Block methods and sample times to create lists
- Atomic subsystems for componentization
 - Heterogeneous composition to avoid dependencies
 - Heterogeneous execution lists (flat and hierarchical)
 - Scoped the class of systems where this applies



The slide features a dark blue header bar with the MathWorks logo and "MATLAB&SIMULINK" text. Below the header is a white content area with a dark blue border. The title "References" is at the top of the content area. Two yellow callout boxes contain academic citations.

Pieter J. Mosterman and John E. Ciolfi, "Using Interleaved Execution to Resolve Cyclic Dependencies in Time-Based Block Diagrams," in *Proceedings of 43rd IEEE Conference on Decision and Control* (CDC'04), pp. 4057-4062, December 14 - 17, Atlantis, Paradise Island, Bahamas, 2004.

Ben Denckla and Pieter J. Mosterman, "An Intermediate Representation and Its Application to the Analysis of Block Diagram Execution," in Proceedings of the 2004 Summer Computer Simulation Conference (SCSC'04), pp. 167-172, July 25 - 29, San Jose, CA, 2004.

57



The slide features a dark blue header bar with the MathWorks logo and "MATLAB&SIMULINK" text. Below the header is a white content area with a dark blue border. The title "Acknowledgments" is at the top of the content area. A yellow callout box contains the names of two individuals.

John E. Ciolfi
MathWorks

Ben Denckla
Independent Thinker

Many thanks for their continuing collaboration!

58

