

Schedulers and Code Size

- For SDF semantics, use Ptolemy's **single-processor SDF schedulers** (**default, Joe's, SJS**) and **multi-processor SDF schedulers** (**Hu level, Sih dynamic level, Sih declustering**)
- **Code size** varies dramatically for different schedulers, especially for systems with large sample rate changes
- **Buffer size** also varies with different schedulers
- **Looping** requires indirect addressing of buffers, more overhead (buffer pointer, indirect addressing)
- **Tradeoff code size vs. buffer size**

```
/* Schedule 1 */
```

```
fire_star_A();  
fire_star_A();  
fire_star_A();  
fire_star_A();  
fire_star_A();
```

```
/* Schedule 2 */
```

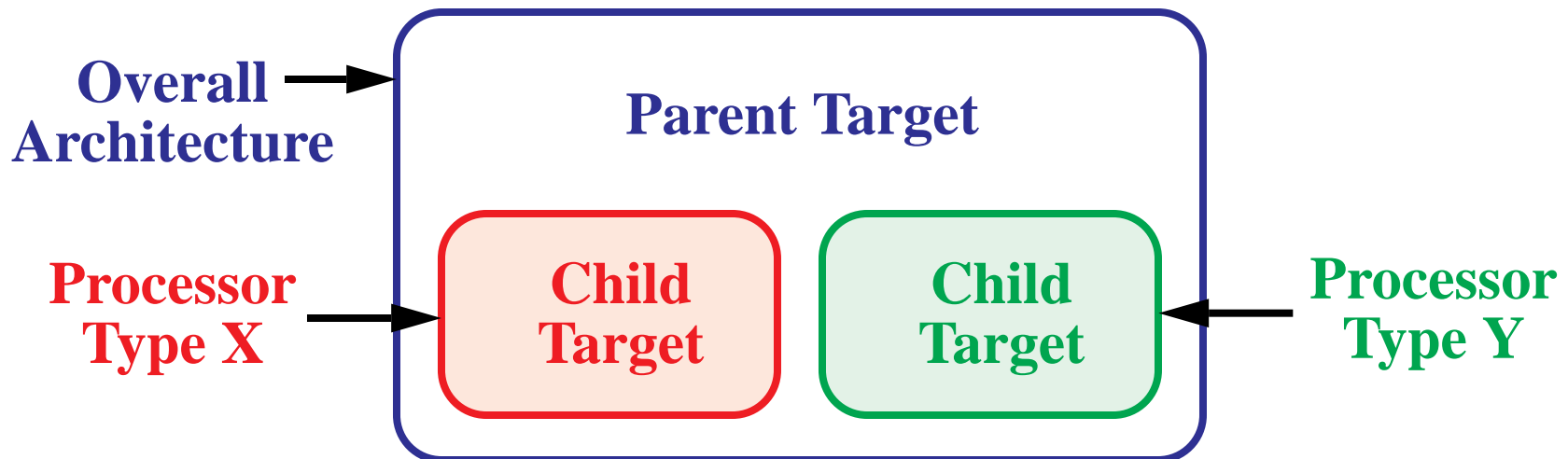
```
repeat(5) {  
    fire_star_A();  
}
```

Methods in Code Generation

- `generateCode()`
- `allocateMemory()`
- `codeGenInit()`
- `mainLoopCode()`
- `go()`
- `wrapup()`
- `frameCode()`
- `writeCode()`
- `compileCode()`
- `loadCode()`
- `runCode()`

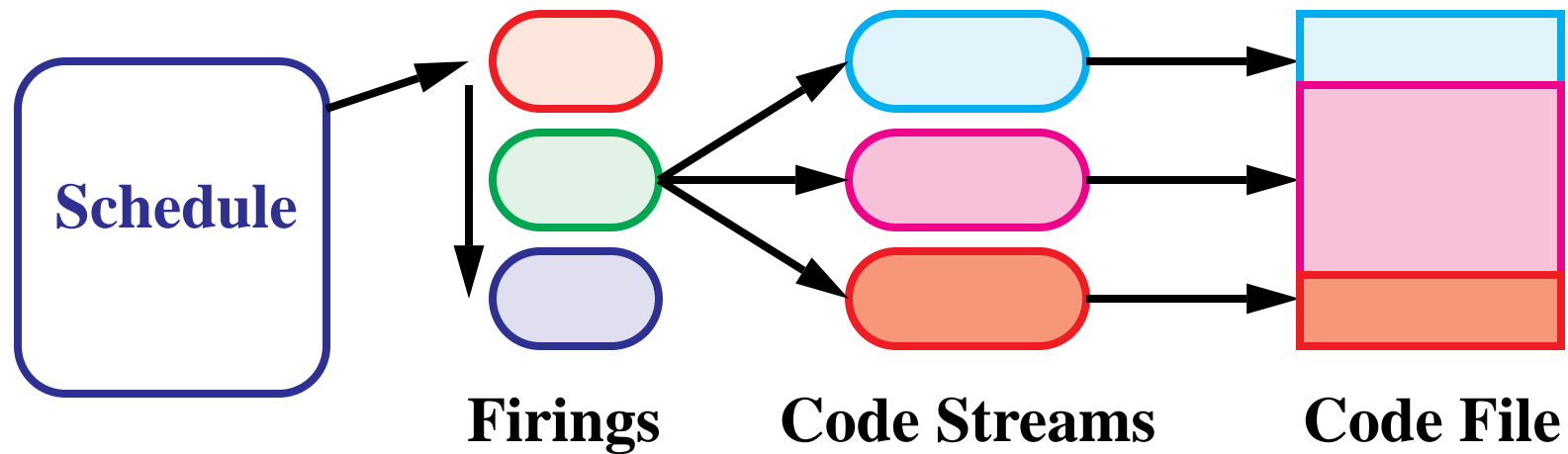
Multitargets

- Multiprocessor architectures
- Use targets hierarchically
- Heterogeneous architectures use heterogeneous targets
- Child targets generate code for their portion of the application
- Parent target coordinates all code generation and interaction with the architecture
- Special stars for synchronization between processors



Code Generation Targets

- **Targets correspond to architectures (single processor, multiprocessor, fully connected, shared bus, etc.)**
- **Multiple targets available for a single code generation domain**
- **Methods to generate schedule, fire individual stars**
- **Methods to generate individual code streams, assemble code streams into code file, generate code file**
- **Methods for compiling code, downloading code, running code**



Attributes

- **State Attributes**

- A_GLOBAL, A_LOCAL
- A_SHARED, A_PRIVATE
- A_CONSTANT, A_NONCONSTANT
- A_SETTABLE, A_NONSETTABLE
- A_CIRC
- A_CONSEC
- A_MEMORY
- A_NOINT
- A_REVERSE
- A_SYMMETRIC
- A_ROM, A_RAM

- **Porthole Attributes**

- P_CIRC, P_SHARED, P_SYMMETRIC, P_NOINT

Macros

- **Base CGStar Macros**

- `$ref(name), $ref(name, offset)`
- `$val(state-name)`
- `$size(name)`
- `$starName()`
- `$fullName()`
- `$starSymbol(name)`
- `$sharedSymbol(list, name)`
- `$label(name), $codeblockSymbol(name)`

- **Additional Macros**

- `$addr(name), $addr(name, offset)`
- `$ref(name, offset)`
- `$mem(name)`

Code Streams

- Distinguish among different sections of code (initialization, main loop, wrapup, declarations, procedures, functions, compiler directives)
- Methods related to `addCode ()` to add code to specific code streams (e.g. `addProcedure ()` to add code to `procedures` code stream)
- Variations of `addCode ()` with arguments to control adding code to an argument code stream
- Check if same code has already been added, avoid redundant code for shared resources

Codeblocks

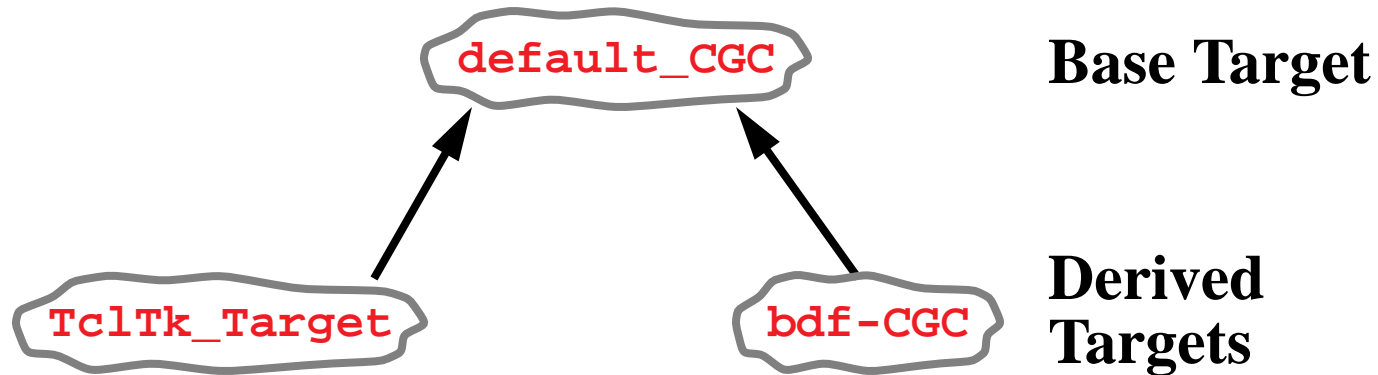
```
codeblock (std) {  
    $ref(output) = $ref(value);  
    $ref(value) += $val(step);  
}  
  
go { addCode(std); }
```

- A section of code which can be generated by a star
- Written in the output code language with interspersed **macros** to be expanded by preprocessing
- Referred to by an identifying name (**std**)
- Added to output code with **addCode()** or related methods
- Decision-making during star firing for conditional code generation

Elements of Code Generation Stars

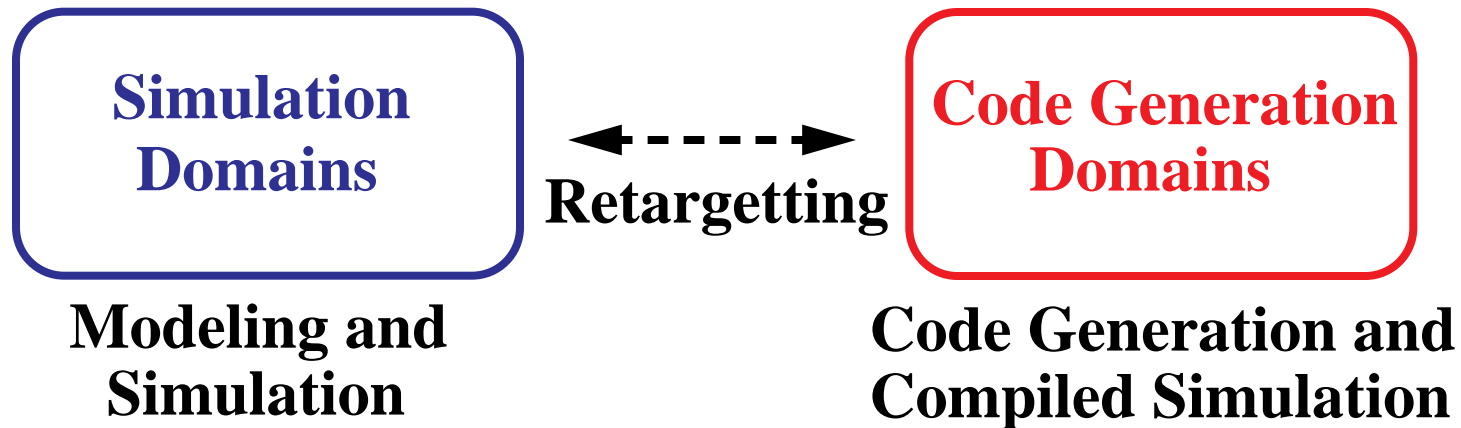
- Similar to simulation stars: `setup()`, `go()`, `wrapup()` methods
- Additional methods: `initCode()`, `execTime()`
- `setup()`: Initialize local variables and states, called before schedule is generated
- `initCode()`: Generate initialization code before the main loop, procedure declaration code outside the main loop, called after schedule is generated
- `go()`: Generate code in the main loop for one firing
- `wrapup()`: Generate code after the main loop
- `int execTime()`: Returns execution time of one firing in main loop in processor cycles or instruction steps; Used primarily in parallel scheduling

Code Generation Classes



- **Domains correspond to particular languages (C, assembly, Silage, VHDL)**
 - Default target
 - Supported star type
- **Targets correspond to different target architectures**
 - Derived from base class target
 - Control scheduling, compiling, assembling, downloading code
 - Multiprocessor architectures: task partitioning and synchronization

Relationships Between Domains



- **CG domain for comment generation, test schedulers**
- **Domains derived from CG for language-specific code generation**
- **Dataflow semantics supported, other semantics possible**
- **CG kernel: processor-independent actions**
 - allocate memory for buffers, constants, tables
 - generate symbols, labels

Generate System Descriptions for Design Tools

The image displays a design tool interface. At the top, a schematic diagram shows a component labeled 'RampInt' connected to a component labeled 'FIRInt'. Below this, a 'Design Analyzer' window is open, showing a detailed schematic view of the 'FIRInt' component. The schematic view includes a toolbar on the left and a main area with a complex circuit diagram. Below the schematic view, the text 'Current Design: FIRIntTest' and 'Schematic View' is visible. To the left of the Design Analyzer window, a text editor window displays VHDL code for the 'FIRIntTest' design.

```

-- Cluster: FIRIntTest_RampInt_0
entity FIRIntTest_RampInt_0 is
    port(
        FIRIntTest_RampInt5_output_0: IN INTEGER range 0 to 15;
        FIRIntTest_RampInt5_value_1_in: IN INTEGER range 0 to 15;
        FIRIntTest_RampInt5_value_2_in: IN INTEGER range 0 to 15;
    );
end FIRIntTest_RampInt_0;

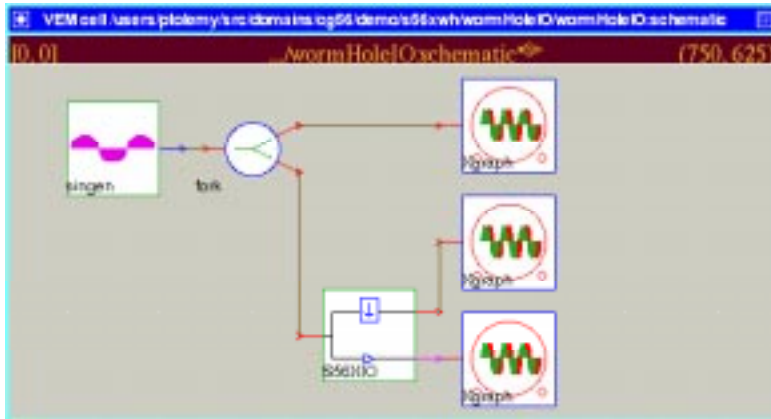
architecture Behavior of FIRIntTest_RampInt_0 is
begin
    process
    begin
        FIRIntTest_RampInt5_value_1_in
    end process;
end Behavior;

-- Cluster: FIRIntTest_FIRInt_0
entity FIRIntTest_FIRInt_0 is
    port(
        FIRIntTest_FIRInt5_output_0: IN INTEGER range 0 to 15;
        FIRIntTest_FIRInt5_output_1: IN INTEGER range 0 to 15;
        FIRIntTest_FIRInt5_output_2: IN INTEGER range 0 to 15;
        FIRIntTest_FIRInt5_output_3: IN INTEGER range 0 to 15;
        FIRIntTest_FIRInt5_output_4: IN INTEGER range 0 to 15;
    );
end FIRIntTest_FIRInt_0;

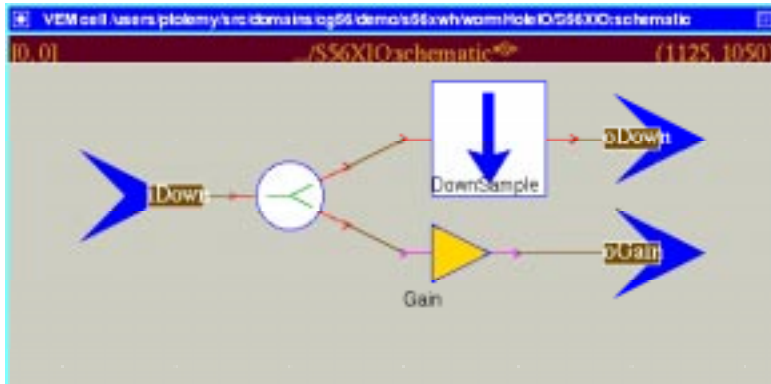
```

- **Generate system descriptions for processing by other design tools (Silage-->Hyper, VHDL-->Synopsys)**
- **Domains: Silage, VHDLF, VHDLB**

Generate Code for Co-simulation



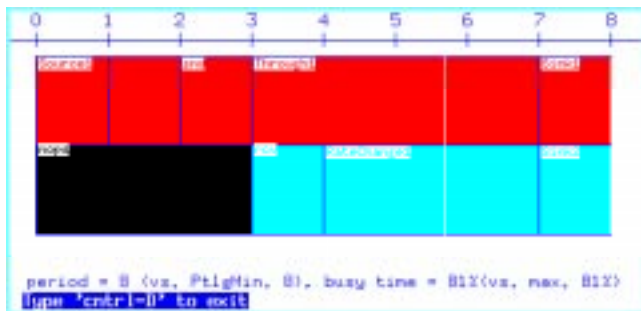
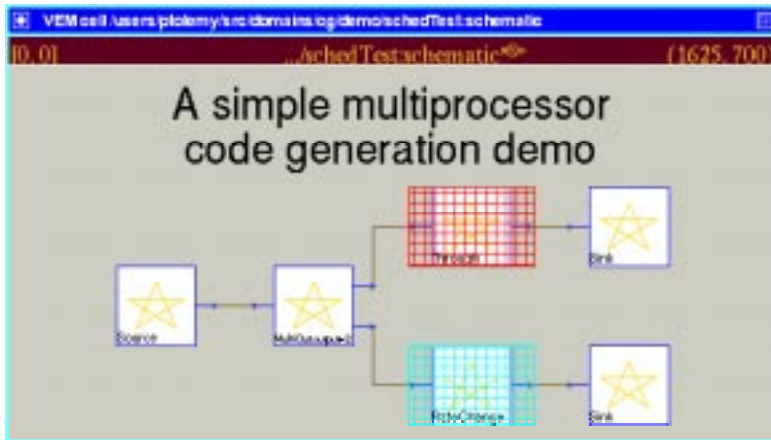
Top-level SDF system



CG56 subsystem

- **Generate code for co-simulation between Ptolemy and code running on hardware connected to a workstation**
- **Targets: S-56X, S-56XWH**

Test Multiprocessor Schedulers

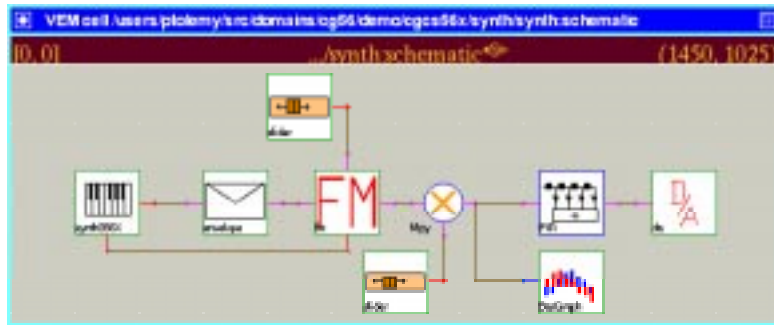


Parameter	Value
directory:	HOMEPTOLEMY_SYSTEMS
file:	
display:	YES
compile?	NO
run?	NO
nprows:	2
inheritProcessors:	NO
seedTime:	1
useStarOverProc:	NO
manualAssignment:	NO
adjustSchedule:	NO
childType:	default-CG
resources:	
refTimeScales:	1
ganttChart:	YES
logFile:	stdout-
serializedComm:	NO
ignoreIPC:	NO
overlapComm:	NO
useCluster:	YES
Use multiple schedulers?:	NO

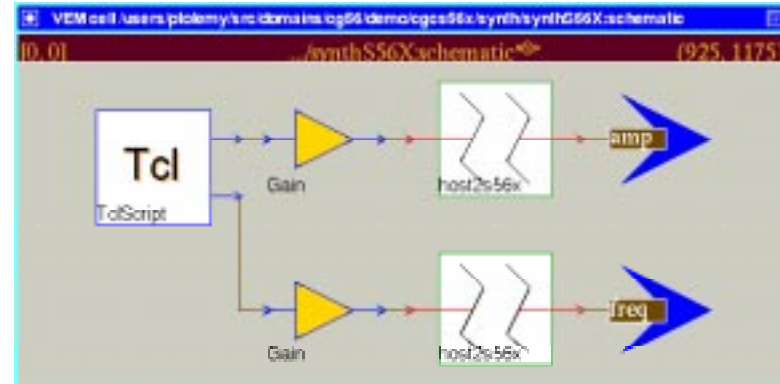
- **Domains: CG; CGTargets: FullyConnected, SharedBus**
- **Other Targets: unixMulti_C, MultiSim-56000**

Generate Code for Heterogeneous Architectures

Top-level CG56 system



CGC subsystem



56000 Code

Run on processor
on S-56X card

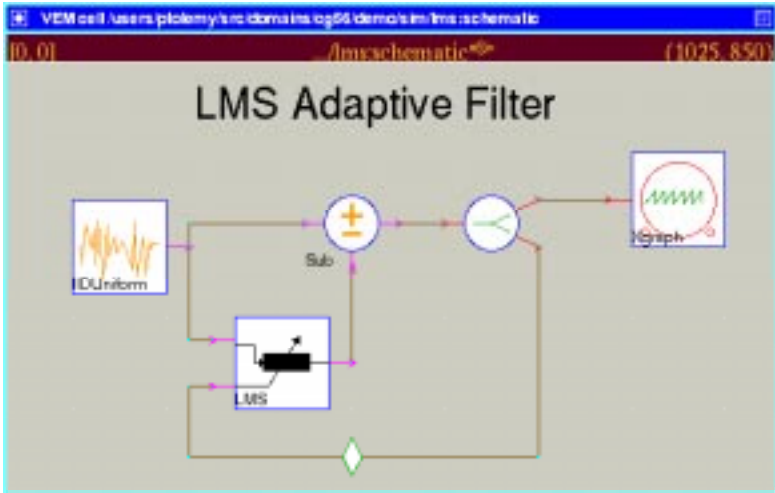
C Code

Run on SparcStation

Target sets up communication

- Domains: CG56
- Targets: CGC-S56X

Generate Code for Special-Purpose Processors



Generated Code

```
...
org p1
...
org p2
...
org p3
...
org p4
...
org p5
...
org p6
...
org p7
...
org p8
...
org p9
...
org p10
...
org p11
...
org p12
...
org p13
...
org p14
...
org p15
...
org p16
...
org p17
...
org p18
...
org p19
...
org p20
...
org p21
...
org p22
...
org p23
...
org p24
...
org p25
...
org p26
...
org p27
...
org p28
...
org p29
...
org p30
...
org p31
...
org p32
...
org p33
...
org p34
...
org p35
...
org p36
...
org p37
...
org p38
...
org p39
...
org p40
...
org p41
...
org p42
...
org p43
...
org p44
...
org p45
...
org p46
...
org p47
...
org p48
...
org p49
...
org p50
...
org p51
...
org p52
...
org p53
...
org p54
...
org p55
...
org p56
...
org p57
...
org p58
...
org p59
...
org p60
...
org p61
...
org p62
...
org p63
...
org p64
...
org p65
...
org p66
...
org p67
...
org p68
...
org p69
...
org p70
...
org p71
...
org p72
...
org p73
...
org p74
...
org p75
...
org p76
...
org p77
...
org p78
...
org p79
...
org p80
...
org p81
...
org p82
...
org p83
...
org p84
...
org p85
...
org p86
...
org p87
...
org p88
...
org p89
...
org p90
...
org p91
...
org p92
...
org p93
...
org p94
...
org p95
...
org p96
...
org p97
...
org p98
...
org p99
...
org p100
...

```

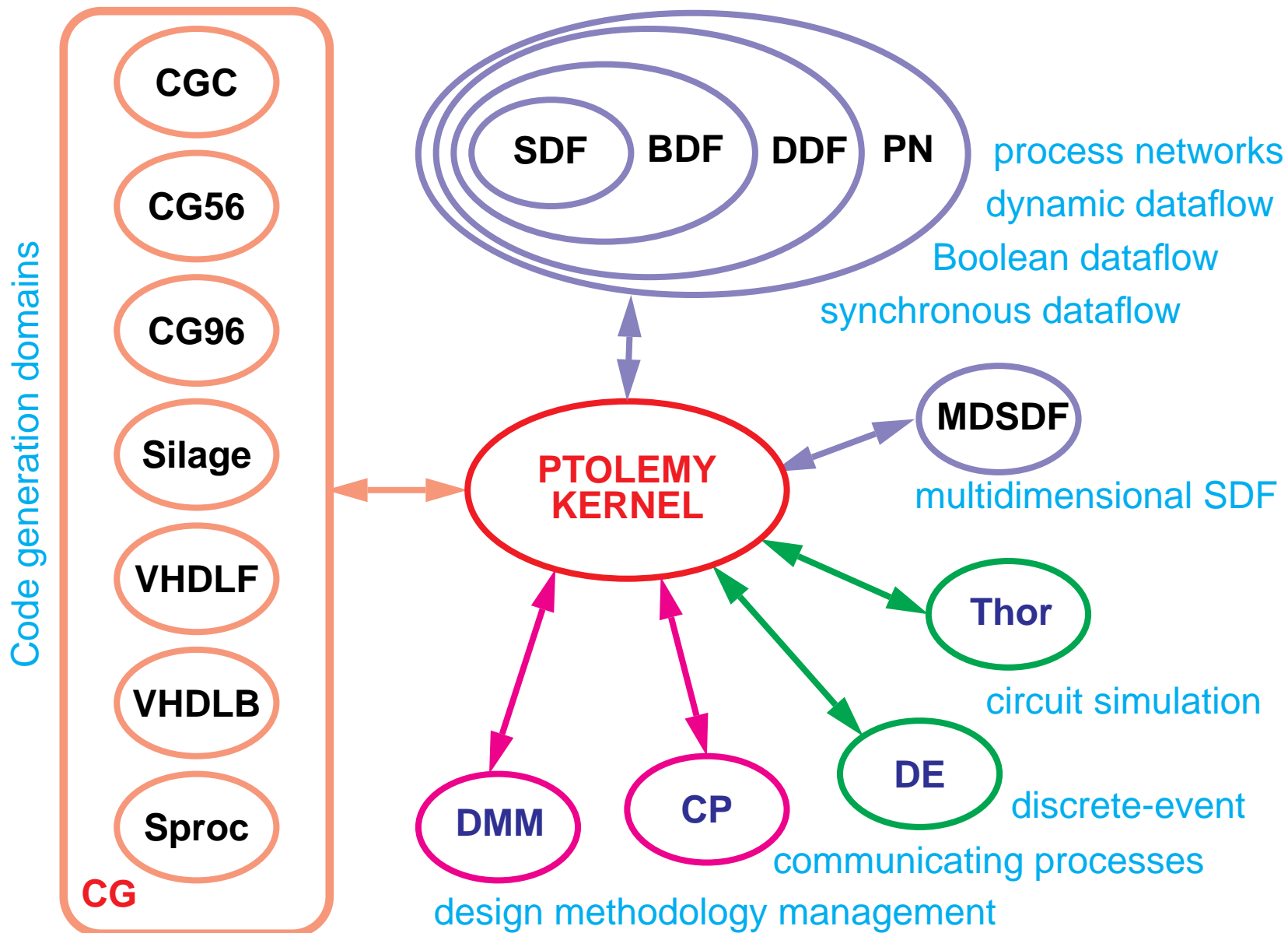
Compiler

Processor Simulator

Processor Architecture

- Domains: CG56, CG96, Sproc

Domains in Ptolemy



Outline

- **Applications of Code Generation**
- **Relationships Between Domains**
- **Code Generation Classes**
- **Elements of Code Generation Stars**
 - **Codeblocks**
 - **Code Streams**
 - **Macros**
 - **Attributes**
- **Code Generation Targets**
- **Multitargets**
- **Methods in Code Generation**
- **Schedulers and Code Size**

Code Generation Concepts



Michael C. Williamson

UC Berkeley