What is going on with LEMS and jLEMS?

How can we support multicompartmental cells in LEMS?

Executing jLEMS models natively and on exotic hardware.
Where do the numerical methods come in?

Executing LEMS models via LEMS-Lite

- Familiar old problem
  - Published descriptions of models in neuroscience are almost always inadequate for reproducing results.
  - You can't do much science if you can't analyze, extend, reuse or build upon other people's work.

- Familiar old solution
  - Declarative model descriptions using basic design principles form software engineering: separate logic (equations) from data (parameter values); avoid duplication; think about your design and refactor as needed.

    Phiysics, Geometry

    Neuroscience-specific definitions

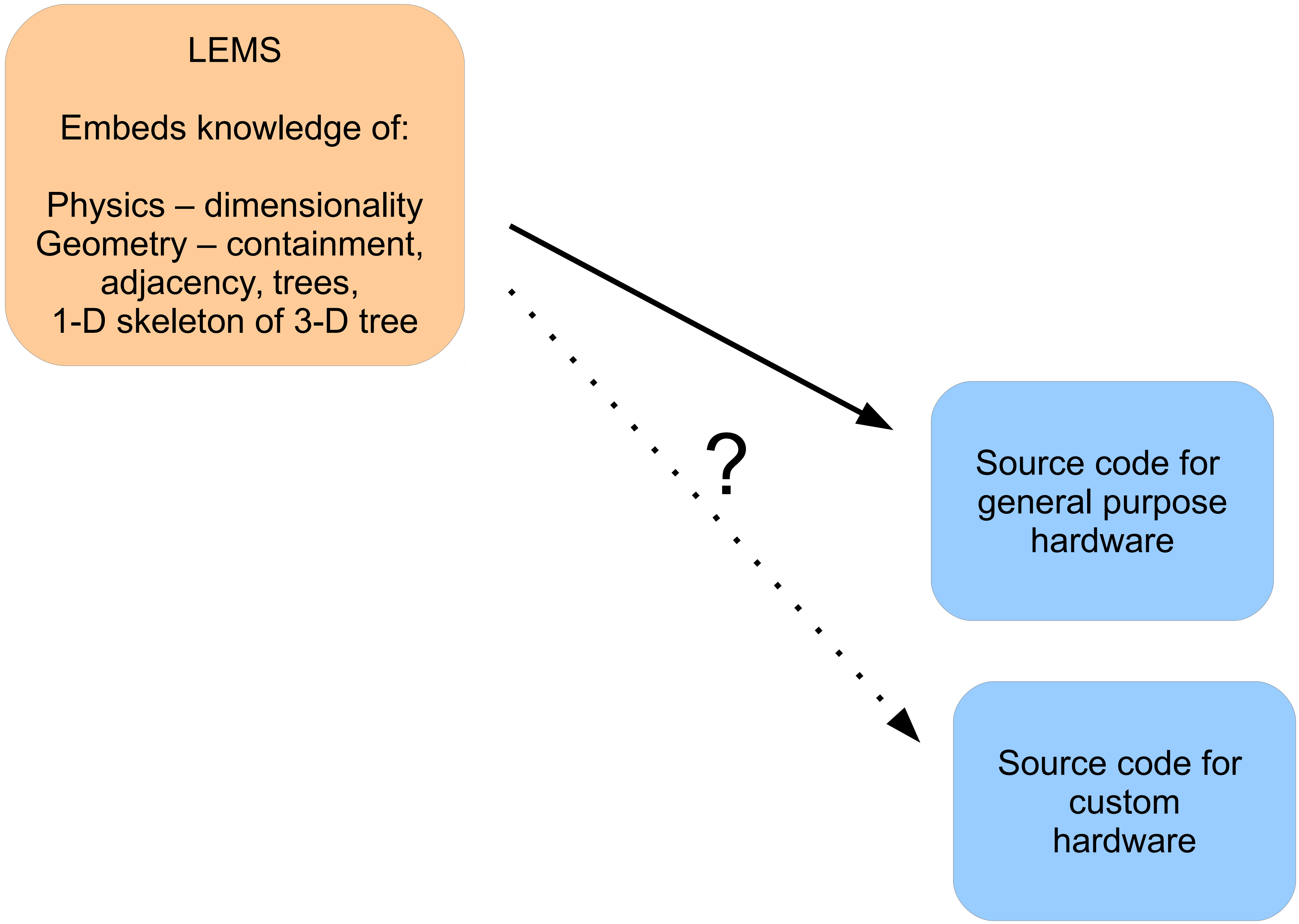
  - Implemented in LEMS/NeuroML. Related efforts include NineML, SpineML, SBML although these focus more on machine-readability, less on human writeability.

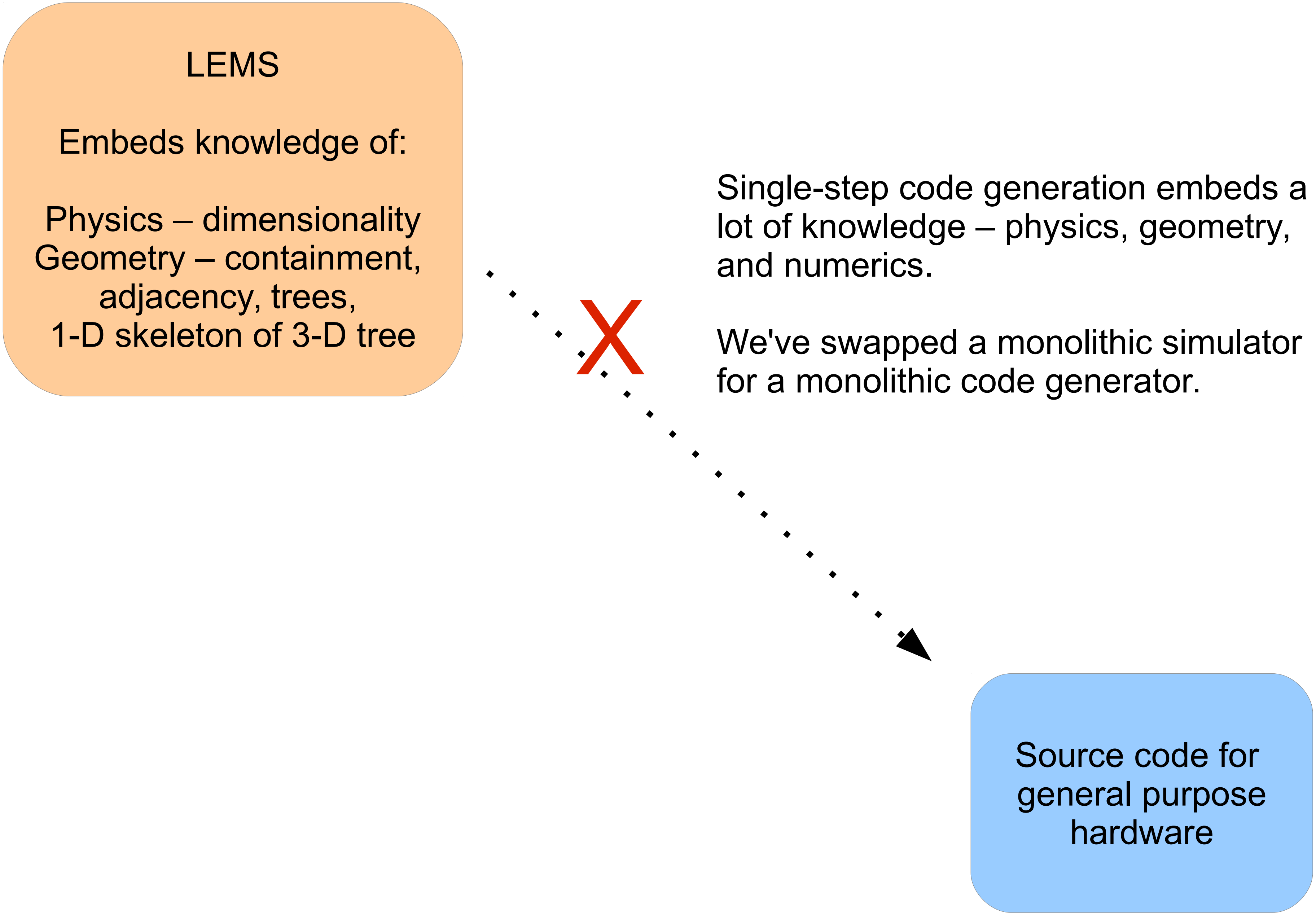
What next?

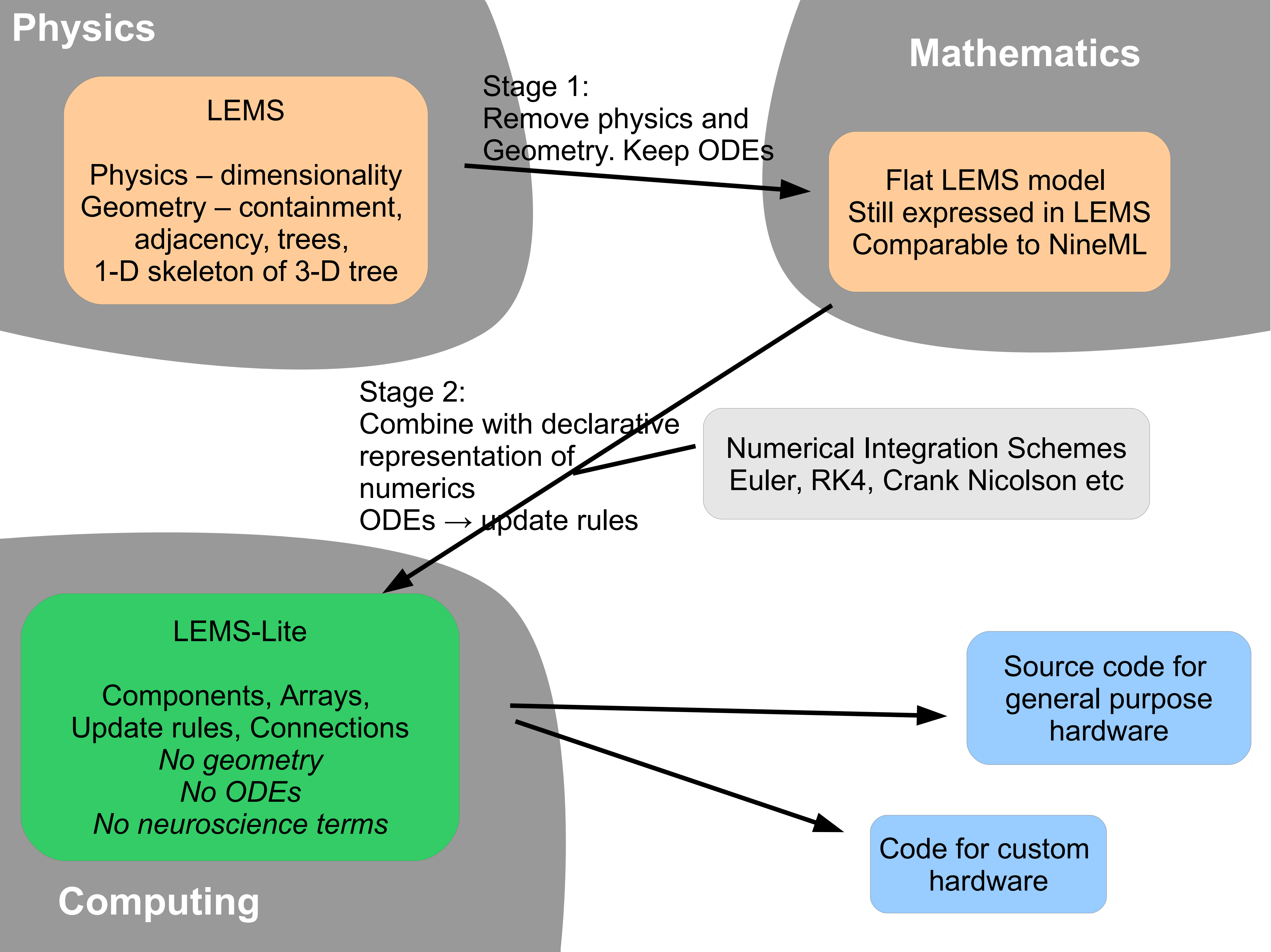
- How do you execute models?
  - Map NeuroML/LEMS models to existing tools, such as Neuron.
  - New simulators designed for the LEMS data model. jLEMS, pyLEMS
  - Generate code for general purpose compilers.
  - Generate code for custom hardware.

Why generate code rather than using Neuron, Moose, Genesis, Brian, PSICS etc?

- Exploit novel hardware including FPGAs and GPUs
- Work with more diverse models
  - Neuron already involves code generation, Moose relies on custom extensions
- Efficiency and memory footprint
  - Eg integerization of models (Mike Hull)
- Why not?
  - Models are like programs. The obvious thing is to compile them, not just run them on a language simulator.
  - The alternative view is that models are like **data**, to be fed into specialist programs. It all depends on the completeness and diversity of your model specifications.


LEMS
Embeds knowledge of:
Physics – dimensionality
Geometry – containment,
adjacency, trees,
1-D skeleton of 3-D tree
?
Source code for
general purpose
hardware
Source code for
custom
hardware


LEMS
Embeds knowledge of:
Physics – dimensionality
Geometry – containment,
adjacency, trees,
1-D skeleton of 3-D tree
Single-step code generation embeds a
lot of knowledge – physics, geometry,
and numerics.
We've swapped a monolithic simulator
for a monolithic code generator.
Source code for
general purpose
hardware


Physics
LEMS
Physics – dimensionality
Geometry – containment, adjacency, trees,
1-D skeleton of 3-D tree
Stage 1:
Remove physics and Geometry. Keep ODEs
Mathematics
Flat LEMS model
Still expressed in LEMS
Comparable to NineML
Stage 2:
Combine with declarative representation of numerics
ODEs → update rules
Numerical Integration Schemes
Euler, RK4, Crank Nicolson etc
LEMS-Lite
Components, Arrays,
Update rules, Connections
No geometry
No ODEs
No neuroscience terms
Source code for general purpose hardware
Code for custom hardware
Computing

```xml
<LEMSLite>

  <DiscreteUpdateComponent name="lif_neuron">

    <Interface>
      <Parameter name="bias"/>
      <Parameter name="gain"/>
      <Parameter name="constInput"/>

      <InputEventPort name="spike-in">
        <Parameter name="weight"/>
      </InputEventPort>

      <OutputEventPort name="spike-out"/>

      <Constant name="one_over_rc_float" value="0.0488281"/>
      <Constant name="ptsc_scale_float" value="0.154279"/>

      <OutputVariable name="v"/>
    </Interface>

    <State>
      <StateVariable name="v"/>
      <StateVariable name="inp"/>
      <StateVariable name="ref"/>
    </State>

    <Step>
      <Var name="total" value="(gain * (inp + constInput)) + bias"/>
      <Var name="dv" value="(total-v) * one_over_rc_float"/>
      <Update variable="v" value="v + dv"/>
      <Update variable="inp" value="inp * (1. - ptsc_scale_float)"/>
      <Output variable="v" value="v"/>
    </Step>

    <OnEvent port="spike-in" >
      <Update variable="inp" value="inp + weight * one_over rc"/>
    </OnEvent>

    <OnCondition if="v .gt. 1.0">
      <Update variable="v" value="0"/>
      <Update variable="ref" value="2"/>
      <Emit port="spike-out"/>
    </OnCondition>

    <OnCondition if="ref .gt. 0">
      <Update variable="v" value="0"/>
      <Update variable="ref" value="ref-1"/>
    </OnCondition>

  </DiscreteUpdateComponent>
```

```xml
  <DataSources>
    <File name="mh_conv_level0" id="f_params_pop0" format="csv" shape="(5,3000)"/>
    <Array name="pop0_bias"> <FileSource file="f_params_pop0" column="1"/> </Array>
  </DataSources>

  <ComponentArray name="level0" component="lif_neuron" size="3000">
    <Let parameter="constInput" array="pop0_constInput"/>
    <Let parameter="bias" array="pop0_bias"/>
    <Let parameter="gain" array="pop0_gain"/>

    <Initialize stateVariable="inp" array="pop0_inp" />
  </ComponentArray>

  <EventConnections name="pop0_to_pop1" from="level0" to="level1">
    <EventSource port="spike-out"/>
    <EventTarget port="spike-in"/>

    <SourceTargetConnector>
      <FromArrayConnector pre="conn01_pre" post="conn01_post"/>
    </SourceTargetConnector>

    <ConnectionProperties>
      <Property name="weight" array="conn01_weight"/>
      <Delay value="0"/>
    </ConnectionProperties>

    <EventArguments>
      <Arg name="weight" value="connection.weight"/>
    </EventArguments>

  </EventConnections>

  <Simulation name="handwriting_simulation" dt="1.0e-3" endTime="0.02">
    <OutputFiles>
      <File id="f_out0_csv" name="f_out1.csv" format="csv"></File>
    </OutputFiles>

    <Recording startTime="0" endTime="1" interval="0.1">
      <VariableRecording file="f_out0_csv" componentArray="level0" indices="1,2,3" variable="v"/>
    </Recording>

  </Simulation>

</LEMSLite>
```

Summary: LEMS-Lite

- Acts as a fixed point between the LEMS specification and code generators
- LEMS specification can be revised and extended without affecting downstream implementations provided we maintain the mappings to LEMS-Lite
- lower-level description that nmodl, NineML etc
  - Describes the post-discretization version of the model
  - No ODEs, no neuroscience terminology
  - Reduces uncertainty about what is to be computed or how equations are to be solved

What is going on with LEMS and jLEMS?

How can we support multicompartmental cells in LEMS?

Executing jLEMS models natively and on exotic hardware.
Where do the numerical methods come in?