Orchestrating Multi-Physical Simulations

Michael Lienhardt,

ONERA, Université Paris-Saclay
France

3rd October 2025

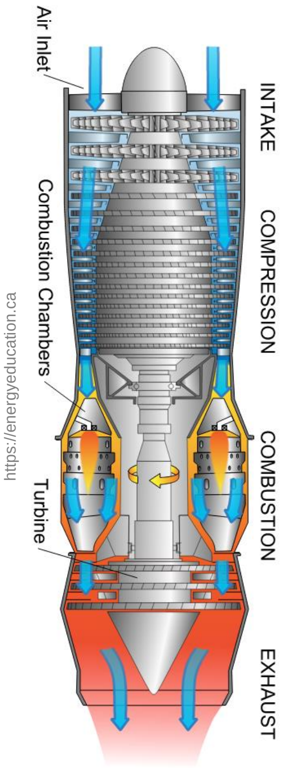
APM

Structure of this Presentation

Usually
Context
↓
Problem (CS) to solve
↓ theories, Tools, …
Solution

This Presentation
Context
↓
Problem Concept
↓ Examples, Analysis, …
Problem (CS) to solve
↓
Solution in progress

Context: Physical Phenomena Simulation



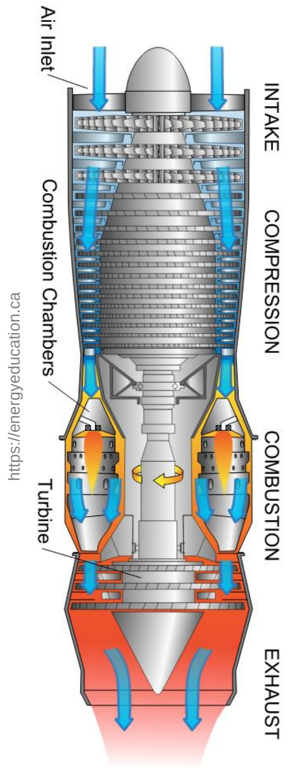
Principle of Simulation:

- Take the equations $f(t, x_i)=0$ encoding the different phenomena at play
- Discretize the space/time of simulation (e.g., using meshes and Δt)
- Implement the equations w.r.t. the discretization done in previous step
- Compute:
  - For all Δt to simulate:
  - Fixpoint to compute the x_i so that the $f(t, x_i)=0$ hold

Possibly many phenomena to take in account:

- Fluid Dynamics of different nature (air/gas/exhaust/rain/…)
- Solid (dilatation/corrosion/fatigue/…)
- Chemistry
- Energetic (e.g., to simulate lightnings or thermal signature)
- …

Context: Physical Phenomena Simulation

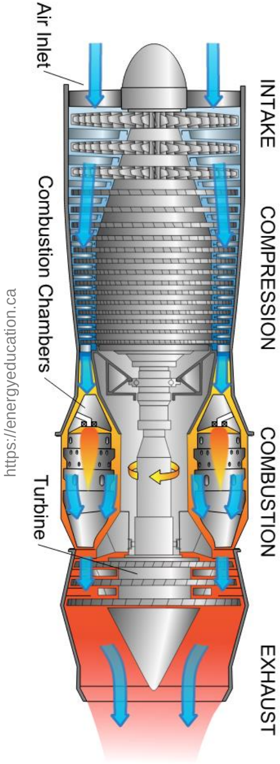


Problem of this approach:
1/3 - needs unrealistic computation space

- Each phenomena has its preferred space discretization
    - e.g., both fluid and energetic use mesh, but with different granularity
    - ➢ either too fine discretization,

      or lots of transfer functions with approximation errors

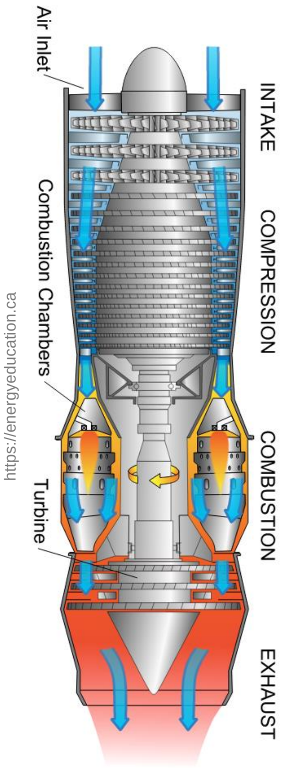
Context: Physical Phenomena Simulation



Problem of this approach:
2/3 - needs unrealistic computation time

- Each phenomena has its preferred time discretization
  - e.g., energetic is instantaneous, fluid is medium, solid is slow
  - ➢ using the greatest common divisor is too fine for most phenomena
  - ➢ also because Δt might change depending on the phenomena at play

Context: Physical Phenomena Simulation

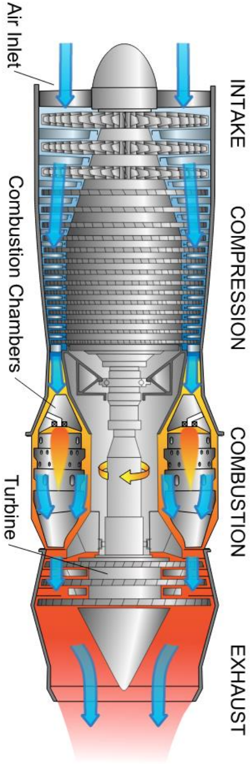


Problem of this approach:
3/3 - ad-hoc tricks specific to each phenomena

- Each phenomena has its specific tricks to make computation quicker
  - e.g., fluid do not simulate turbulence (very fine grain phenomena, chaotic),
    it generates it using functions tailored for the current simulation
  - ➢ requires domain-specific knowledge and implementation

In practice: each phenomena has its dedicated simulator…
and extra work needs to be done to simulate multiple phenomena

Context: Physical Phenomena Simulation

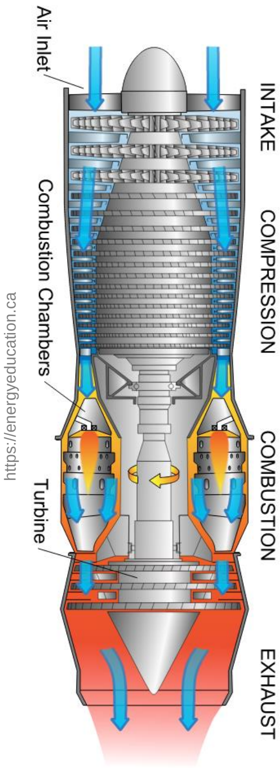


https://energyeducation.ca

Multiple Phenomena Simulation: simulators orchestration

- Problem investigated for since the 70’s
    - studied by apply mathematicians, physicists, numericians, …
    - most solutions are hand-made and one-shot
        - helping tools: cwipi, Palm/OpenPalm
    - dead/live-locks are a real issue
- many computation problems, including
    - data interpolation between meshes of different shapes
    - techniques for different simulators to reach identical values
    - Δt adaptation heuristics to avoid simulators crashing

Context: Physical Phenomena Simulation



Multiple Phenomena Simulation: simulators orchestration

- Need to orchestrate Distributed Programs
  - some simulators are distributed, some not
  - ➢ orchestrator must be distributed to avoid bottle-neck
  - No known publication about the CS aspects of this orchestration

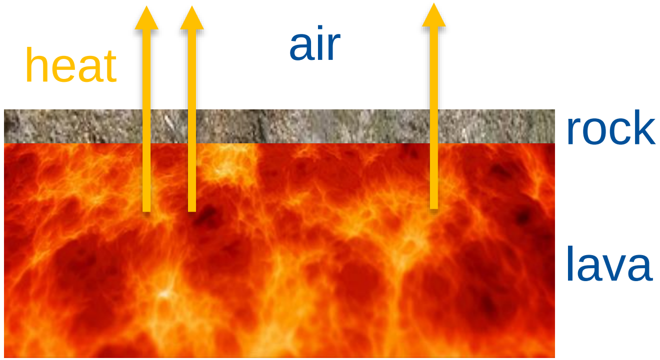
Problem Concept: Example (1/3)

Lava and Heat

- heat is transferred from the lava to the rock, and to the air
- the opposite is technically true, but negligible

➢ need for communications between the different simulators

- the Δt of every simulators are different
- three simulators interact over two “interfaces”:
  - ➢ need to ensure that the computed values on both sides of each interface match



Problem Concept: Example (2/3)

Shard and Lightning

- Which wind pressure the Shard must deal with
- Δt is about 10ms

then, a lightning

- "instantaneously", some gas becomes plasma
- if nothing is done, the fluid simulator breaks:

  too big discontinuities because too quick change

  ➢ Need to split its Δt and progressively insert the lightning

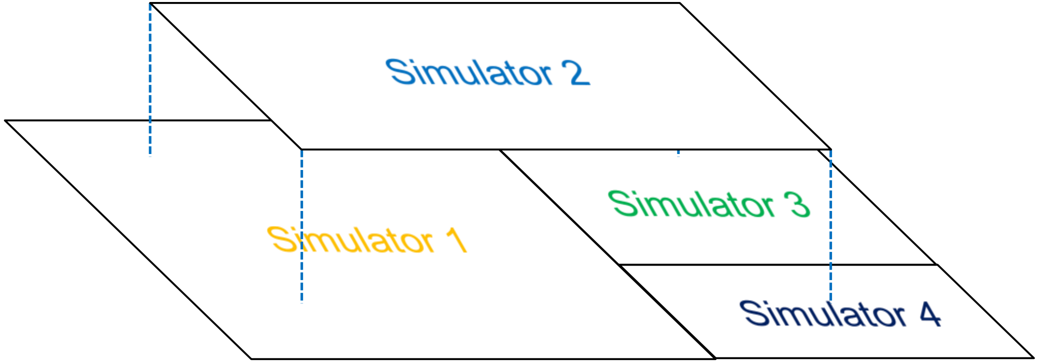
Problem Concept: Example (3/3)

Automn

- Suppose constant wind that pushes the leaves
- The wind pushes the leaves
- The leaves move and impact the air-flow

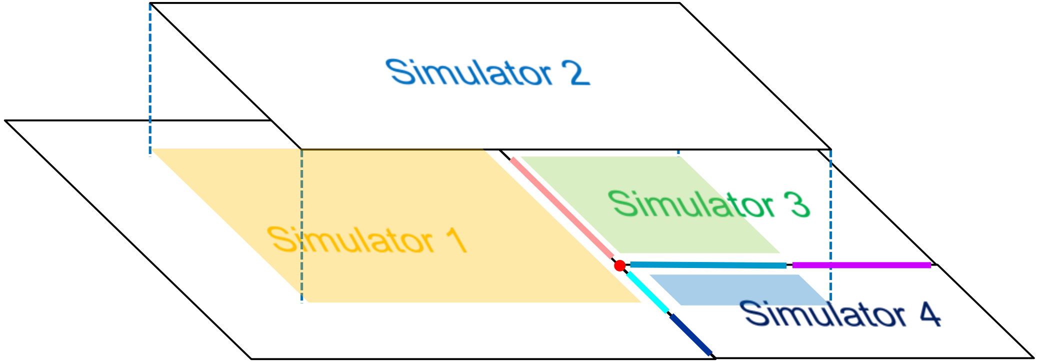
Problem Concept: Analysis

Simulators computation overlaps on the physical space



Problem Concept: Analysis

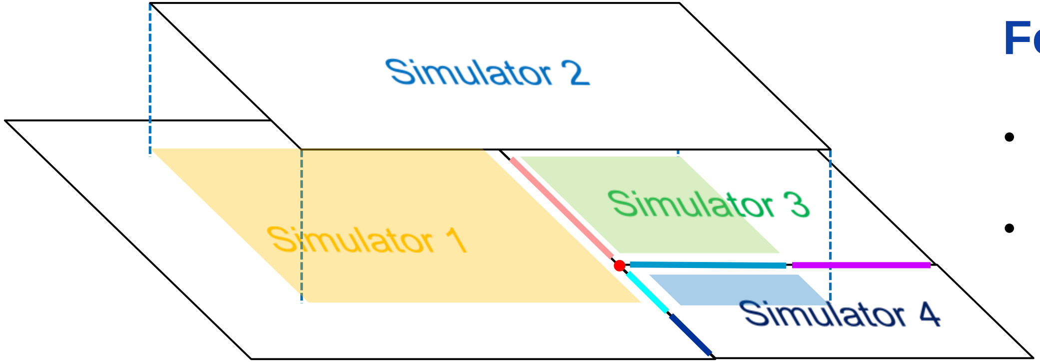
Simulators interact over “interfaces” (i.e., shared spaces)



(orange)	Volume: sim1 ⇔ sim2
(light green)	Volume: sim2 ⇔ sim3
(blue)	Volume: sim3 ⇔ sim4
(pink)	Surface: sim1 ⇔ sim2 ⇔ sim3
(red)	Point: sim1 ⇔ sim2 ⇔ sim3 ⇔ sim4
(cyan)	Surface: sim1 ⇔ sim2 ⇔ sim4
(dark blue)	Surface: sim1 ⇔ sim4
(medium blue)	Surface: sim2 ⇔ sim3 ⇔ sim4
(purple)	Surface: sim3 ⇔ sim4

Problem Concept: Analysis

Simulators interact over “Interfaces” (i.e., shared spaces)



For each Interface:

- Need a **consensus** over when to synchronize
- Need a **consensus** over computed values

(at every synchronization points)

Problem Concept: Analysis

Methods to reach a consensus: synchronization points

- Very physics dependent: how quickly every physics is changing the interface
- Related to the number of iterations each simulator need to reach a fixpoint
- Related to the CFL condition [1] that (partially) characterizes Δt
- only solution: fixpoint over reaching a solution
    - hypothesis: progress
    - hypothesis: fault-less network and agents

[1] https://en.wikipedia.org/wiki/Courant%E2%80%93Friedrichs%E2%80%93Lewy_condition

Problem Concept: Analysis

Methods to reach a consensus: computed values (1/2)

1. Use an Oracle that sets the initial values
    - prescriptive: the values are the ones expected at the end of the computation
    - indicative: the values should be close to the ones expected, but could be changed during computation

    - Possible oracles:
        - none (uses the values of the previous computation step)
        - one of the simulator (e.g., in example 1 and 2)
        - ad-hoc functions

Problem Concept: Analysis

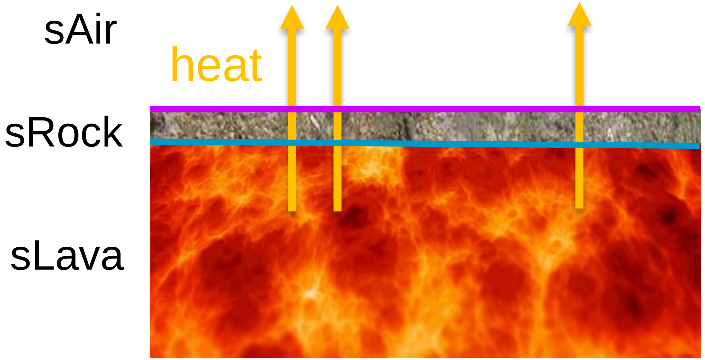
Methods to reach a consensus: computed values (2/2)

2. Use an ad-hoc correction function after computation
    - collect the values computed by every solver, and sets common values
    - prescriptive: the values are the ones expected at the end of the computation
    - indicative: the values should be close to the ones expected, but maybe slightly off
3. Use an additional fixpoint
    - the values produced after the correction is fed as input to the new iteration
        - these values “should” be closer to the expected ones
        - arguments for termination = theorems (in practice, should use a timeout, just in case)

Problem Concept: Analysis

Important details (1/5)

- Simulators as oracles = structural cause of deadlocks
  - Oracles must be executed before the simulators using their values

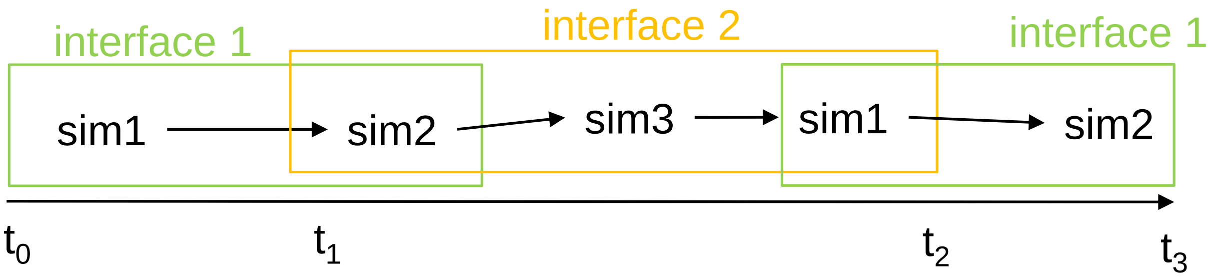


- e.g., if specification error in example 1
  - in interface :
    - Oracle(temperature) = sLava
    - Oracle(pressure) = sRock
  - ➢ Deadlock

Problem Concept: Analysis

Important details (2/5)

- Simulators may start later, and end sooner
  - ➢ Would opposite dependencies in different interfaces also cause deadlocks?



Problem Concept: Analysis

Important details (3/5)

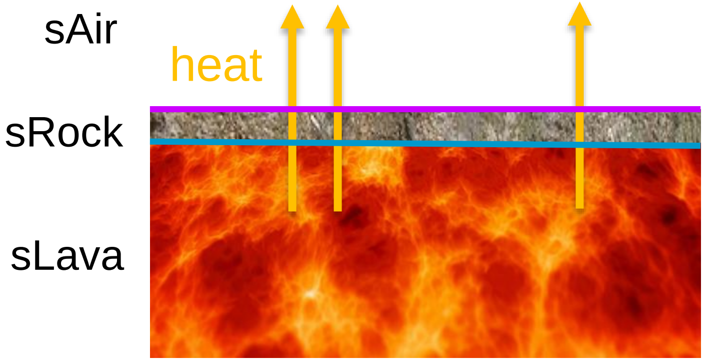
- Consensus can be “asymmetric”

- e.g., in example 2:
  - wind sets the speed of the leaves
  - leaves set the air-flow space

Problem Concept: Analysis

Important details (4/5)

- Fixpoint in interfaces can interact badly
    - Synchronization point = consensus over time advance



- e.g., in example 1:
    - sRock and sAir agree to go to t+Δt( )
    - sLava and sRock cannot agree to go to t+Δt( )
    (Δt( ) < Δt( )) and must restart computation on t
    - ➢ “commit” inconsistency
    - ➢ When sLava and sRock agree, next Δt( ) must ensure progress (sRock is already at t+Δt( ))

Problem Concept: Analysis

Important details (5/5)

- Failure management
  - Most of the time, failure are caused by simulators crashing/not converging
    - bugs
    - bad input values (i.e., the bug is silently from another simulator)
    - bad configuration (e.g., Δt too big or too small)

- Error recovery can be very difficult:
  - need to identify the cause of the problem (difficult)
  - need to roll-back to a safe state (data may be too large to accurately store a safe state)
  - or restart computation (which can take weeks) when the problem is identified

Problem to solve

- **Problem**: model the expected behavior of a simulation orchestration
- Abstraction:
  - Simulators, oracles, ad-hoc corrections = functions
    - in particular, the data updates performed by the correction in the simulators are abstracted away
  - Ordering between functions in an interface = DAG

Solution (in Progress)

Definition

An *OMPS* O is a tuple (F, D, I, L) where:

- F is a set of function names
- D gives the time interval $[f_b, f_e]$ for every function f in F
- I is a set of interfaces $i = (F_i, E_i)$ such that $F_i \subseteq F$ and i is a DAG
- L gives for every interface i the boolean i_l stating if it can loop

In the rest, we suppose given an OMPS $O = (F, D, I, L)$.

Solution (in Progress)

Definition

A *runtime state* is a pair $S = (S_f, S_i)$ where:

- S_f gives, for every function $f \in F$, the pair (f_o, f_u) of its last commited time and its current time
- S_i is a partial function which gives, for every interface $i \in I$, the pair (i_o, i_n) of its last commited time and its next commited time

Solution (in Progress)

Definition

A runtime state $S = (S_f, S_i)$ is *sound* iff:

- S_f gives consistent times for every functions, i.e., $\forall f \in F,\ f_o \leq f_u$
- S_i gives consistent times for every interface, i.e., $\forall i \in I,\ i_o \leq i_n$
- all interface must be considered:
  $\forall i \in I, \bigwedge_{f \in F_i} f_u \in [f_b, f_e[) \Leftrightarrow i \in \mathrm{dom}(S_i)$
- the time of every function must be consistent w.r.t. the interfaces:
  $\forall i \in \mathrm{dom}(S_i),\ \forall f \in F_i,\ i_o \leq f_o \wedge f_u \leq i_n$
- the DAG ordering must be respected:
  $\forall i \in \mathrm{dom}(S_i),\ \forall (f, f') \in E_i, f_u < i_n \Rightarrow f'_u = i_o$

Solution (in Progress)

Definition

The initial runtime state of O is the pair (S_f, S_i) where:

- $S_f(f) = (f_b, f_b)$ for all $f \in F$
- $S_i = \emptyset$

Note: this intial state may not be sound…

Solution (in Progress)

Rule to start an "interface"

$$\frac{i \notin \mathrm{dom}(S_i) \quad \overbrace{\forall f, f' \in F_i, f_u = f'_u}^{\text{all functions are synchronized}} \quad \overbrace{\forall f \in F_i, f_u < f_e}^{\text{all functions can continue}} \qquad \underbrace{(S_f, S_i[i \mapsto (f_u, f_u)]) \rhd_{f_u} (S'_f, S'_i)}_{\text{Compute consensus}}}{(S_f, S_i) \rhd (S'_f, S'_i)}$$

Rule to consensus

$$(S_f, S_i) \rhd_t (S_f\overbrace{[f \mapsto (f_u, f_u), f \in F_i]}^{\text{all functions commit their time}}, S_i[i \mapsto \overbrace{(i_n, \mathrm{consensus}(t, F_i))}^{\text{next rdv for interface given by consensus}}])$$

Solution (in Progress)

Rule to stop an “interface”

$$\frac{i \in \mathrm{dom}(S_i) \quad \overbrace{\forall f, f' \in F_i, f_u = i_n}^{\text{all functions are synchronized at interface's rdv}} \quad \overbrace{\exists f \in F_i, f_u = f_e}^{\text{some function cannot continue}}}{(S_f, S_i) \triangleright (S_f, S_i \setminus \{i\})}$$

Solution (in Progress)

Rule to continue an "interface"

$$\frac{i \in \mathrm{dom}(S_i) \quad \overbrace{\forall f, f' \in F_i, f_u = i_n}^{\text{all functions are synchronized at interface's rdv}} \quad \overbrace{\forall f \in F_i, i_n < f_e}^{\text{all functions can continue}} \quad \overbrace{(S_f, S_i) \triangleright_{i_n} (S'_f, S'_i)}^{\text{Compute consensus}}}{(S_f, S_i) \triangleright (S'_f, S'_i)}$$

Problem to solve

Rule to reset an “interface”

$$\frac{\begin{array}{c} i \in \mathrm{dom}(S_i) \quad \overbrace{L(i)}^{\text{can reset}} \quad \overbrace{\forall f, f' \in F_i, f_u = i_n}^{\text{all functions are synchronized at interface's rdv}} \quad \overbrace{\forall f \in F_i, f_o = i_o}^{\text{all functions can reset to last rdv}} \\ \left.(S_f[f \mapsto (f_o, f_o)], S_i[i \mapsto (i_o, i_o)]) \triangleright_{i_o} (S'_f, S'_i) \quad \right\} \text{Reset timers and compute consensus} \end{array}}{(S_f, S_i) \triangleright (S'_f, S'_i)}$$

Problem to solve

Rule to stop an “interface”

$$\frac{f \in F \quad u \in \left(\overbrace{]f_u, f_e]}^{\text{must finish before its end}} \cap \overbrace{\bigcap_{i\in \mathrm{dom}(S_i), f\in F_i}]f_u, i_n]}^{\text{must finish before end of running interfaces}} \cap \overbrace{\bigcap_{i \in I\setminus \mathrm{dom}(S_i), f\in F_i}^{f'\in F_i, f_u<f'_b}]f_u, f_b]}^{\text{must finish before start of new interfaces}} \right) \quad \left.\bigwedge_{i\in\mathrm{dom}(S_i), f\in F_i} \bigwedge_{(f',f)\in E_i} f'_u \geq i_n \right\} \text{Wait for dependencies}}{(S_f, S_i) \triangleright (S_f[f \mapsto (f_o, u)], S_i)}$$

Note: since this semantics put synchronization points at every function begin/end, opposite dependencies in different interfaces cause deadlocks

Conclusion

This Work:

- Studied the context of Orchestrating of Multiple Physical Simulations
    - From core principle of simulation, to illustrative examples
- Provided a model of interactions
    - Abstracting away data, communication and distribution
    - Focus on time and synchronization

Conclusion

Future Work:

- Study deadlock freedom
- Compare to existing implementations / tools
    - Refine the model
    - Suggest improvements
- Refine the notion of consensus (currently involves all nodes of all simulators)
- Consider recent developments, including
    - temporal interpolation
    - Simulators with local Δt

Thanks

- Jean-Didier Garaud
- Gilles Chaineray
- Karim Anemiche
- Eric Quémerais
- And many others…

Thank You

Questions?